A Modelling Approach to Multi-Domain Traceability

Masoumeh Taromirad

A thesis submitted for the degree of
Doctor of Philosophy

University of York
Department of Computer Science

August 2014

Abstract

Traceability is an important concern in projects that span different engineering domains. Traceability can also be mandated, exploited and managed across the engineering lifecycle, and may involve defining connections between heterogeneous models. As a result, traceability can be considered to be multi-domain.

This thesis introduces the concept and challenges of multi-domain traceability and explains how it can be used to support typical traceability scenarios. It proposes a model-based approach to develop a traceability solution which effectively operates across multiple engineering domains. The approach introduced a collection of tasks and structures which address the identified challenges for a traceability solution in multi-domain projects. The proposed approach demonstrates that modelling principles and MDE techniques can help to address current challenges and consequently improve the effectiveness of a multi-domain traceability solution.

A prototype of the required tooling to support the approach is implemented with EMF and atop Epsilon; it consists of an implementation of the proposed structures (models) and model management operations to support traceability. Moreover, the approach is illustrated in the context of two safety-critical projects where multi-domain traceability is required to underpin certification arguments.

Contents

List of Figures

List of Tables

Listings

Acknowledgements

I will always be grateful to my supervisor, Prof. Richard Paige, and co-supervisor, Dr. Nicholas Matragkas, for their invaluable guidance, help and encouragement.

I would also like to thank my internal examiner and external examiner, Dr. Fiona Polack and Prof. Antonio Vallecillo, for their comments and feedback, which have greatly improved the quality of this thesis.

I am grateful to Dr. Louis Rose for his advice and expertise throughout my research. I am also thankful to my colleagues and friends in the Enterprise Systems research group for their support and friendship.

I would like to express my gratitude to my parents for their unreserved love, support, and prayers. Finally, I am eternally grateful to my husband Mohammad and my son Mahdi for their love, patience, and understanding all these years.

Author Declaration

Except where stated, all the work contained in this thesis represents the original contribution of the author.

Parts of the work described in this thesis have been previously published by the author in:

- Masoumeh Taromirad, Nicholas D. Matragkas, and Richard F. Paige, **Towards Multi-Domain Traceability: a Model-Driven Approach**, submitted and under review in Journal of Software and Systems Modelling, Springer, 2014.
- Masoumeh Taromirad, Nicholas D. Matragkas, and Richard F. Paige, **Towards a Multi-Domain Model-Driven Traceability Approach**, In Proc. of the 7^{th} International Workshop on Multi-Paradigm Modeling (MPM '13), Miami, Florida, USA, September 2013.
- Masoumeh Taromirad and Richard F. Paige, **Agile Requirements Traceability Using Domain-Specific Modelling Languages**, In Proc. of the 2^{nd} International Extreme Modeling Workshop (XM '12), Innsbruck, Austria, October 2012.

1

Introduction

The success of a software system depends on how well it fits the needs of its users and its environment [Nuseibeh and Easterbrook, 2000]. Recent studies show that eliciting, specifying and managing requirements are essential challenges in developing a software system; errors in requirements are claimed to lead to consistent project failure rates of about 50% [Marasco, 2006].

This issue becomes more crucial with critical applications in which failure of the system could lead to loss of life, environmental damage, or significant financial loss. There are numerous accounts of catastrophic software failures such as the London Ambulance System [Finkelstein and Dowell, 1996], and more recently Boeing 777-200 (registered 9M-MRG) [Australian Transport Safety Burea, 2005]. Post-mortem analyses have laid at least partial blame on the incorrect implementation and management of requirements.

Requirements traceability (or, for short, *traceability*), has been widely studied as a mechanism to deal with software development challenges, such as verification and validation and change impact analysis [Gotel and Finkelstein, 1994; Cleland-Huang et al., 2005a]. In some cases, it is mandated so as to comply with regulations, e.g. DO-178B [RTCA and EUROCAE, 1992] in civil aviation projects. This is because traceability is claimed to help ensure that requirements of the system are completely allocated to system elements and developed correctly. Accordingly, traceability is considered as a key element of any rigorous software development process, which would provide critical support for various development activities [Lago et al., 2009].

Nevertheless, there are substantial challenges associated with traceability in practice, such as identifying the most appropriate artefacts to trace [Kirova et al., 2008] and dealing with artefacts represented in heterogeneous formats [Mäder and Cleland-Huang, 2010]. This makes it difficult to define effective *traceability solutions* – consisting of a Traceability Information Model (TIM), traceability process, and analysis tools – to provide traceability; such solutions are thus still rarely defined and used [Mäder et al., 2009b].

In the following, the scope of the research presented in this thesis is specified and the problems motivating this research are highlighted. Accordingly, the research hypothesis and objectives are outlined and a summary of the results and the main contributions of this research are provided. Then, the research method is briefly introduced and, finally, an overview of the structure of the thesis is provided.

1.1 Motivation: Traceability in Multi-domain Context

In many contexts in which different kinds of traceability are mandated (i.e. developing high-assurance software systems [Cleland-Huang et al., 2012]), projects extend across multiple domains. These could be either engineering domains (e.g. business, software, mechanics, and safety), problem domains (e.g. avionic, medical, and financial), or even technological domains (e.g. embedded systems, web-based systems, and real-time systems). A domain represents a collection of concepts fundamentally associated together for a particular and distinguishable purpose. Therefore, domains are not restricted to the aforementioned ones and could refer to other concepts (possibly in different levels) as long as they coherently serve a single purpose.

Each domain has its specific stakeholders, goals, artefacts, and tools. Stakeholders from any single domain may be concerned with both intra- and inter-domain traceability. For example, a software developer will be interested in traces from system to software requirements, while a safety engineer will want to trace relationships between fault tree analysis, software requirements and verification artefacts. Accordingly, as traceability is usually required throughout the project lifecycle, traceability is a *multi-domain* concern. A traceability solution needs to operate across the project's different domains, effectively deals with various artefacts (documents, models, databases) in different domains and relationships between them.

Nevertheless, software development projects are increasingly becoming more distributed and dependent on different software and techniques. Existing traceability approaches still fall short in tracing scenarios that extend across tool boundaries [Asuncion and Taylor, 2012]. This also motivates the need for traceability solutions which operate across different aspects of a project and, consequently, support such scenarios more easily or effectively.

On the other hand, it is widely accepted that a cost-effective traceability solution is the one that is tailored to a particular project situation and supports traceability goals (e.g. tailoring the granularity and types of trace links). This is because project-specific traceability goals drive traceability activities [Aizenbud-Reshef et al., 2005] and project characteristics are critical in finding the *necessary and sufficient* amount of required information to record [Egyed et al., 2007]. However, due to the lack of practical guidance on how to design, implement, and use project-specific traceability constructs,

such solutions are rarely defined and used in practice [Mäder et al., 2009a]. Particularly, in the context of multi-domain traceability, existing approaches do not consider the fact that a project-wide view of traceability information is just a comprehensive, pervasive, and coherent view of the available information scattered in different domains and none of them provide mechanisms to address this issue explicitly and effectively.

In the following, we introduce a safety-critical systems engineering project and demonstrate general traceability scenarios which motivate the research presented in this thesis. Throughout the thesis, we use examples of this project to illustrate the proposed approach.

1.1.1 Example

We are applying our proposed approach in the context of safety-critical projects, in which traceability is required. In such systems, tracing hazards to mitigation is mandated by certification authorities [RTCA and EUROCAE, 1992]. Traceability from hazards to derived safety requirements and to implemented and verified design solutions provides essential evidence to argue that a system is operationally safe [Lutz, 2000].

The Integrated Altitude Data Display System (IADDS) is responsible for providing pilots with altitude data during flight [Charalambous, 2007]. It is also responsible for issuing audible and visible warnings to pilots whenever pilot-specified altitude limits are reached. In this project, traceability is both a mandatory requirement of safety standards (i.e. DO-178B [RTCA and EUROCAE, 1992]) and essential in addressing the lifecycle management challenges inherent in the safety domain. In particular, the project requires multi-domain traceability: requirements engineering tasks must trace to software engineering tasks and to safety engineering tasks. Largely the project is dominated by requirements to enable certification: if the system cannot be certified then the project fails, and certification requires detailed traceability information (e.g. according to DO-178B).

The process used for developing the software deliverables of the project resembles the XP process following the pipelined iteration model introduced in [Paige et al., 2011]. Each iteration comprises a series of steps that are performed iteratively and each of them consists of a sequence of substeps that may be performed recursively as necessary. Fig. 1.1 shows the conceptual overview of steps, substeps (activities), and artefacts in iteration N of development. It also shows the input and output of activities and the overall sequence in which they are performed.

Generally, *user stories* are derived from *software system requirements*; *development models* are designed according to user stories and implemented by *code*. Meanwhile, safety engineering requires additional artefacts to be produced and traced to the general software development artefacts. *Hazards* are identified through *assessments* (e.g. Functional Hazard Assessment),

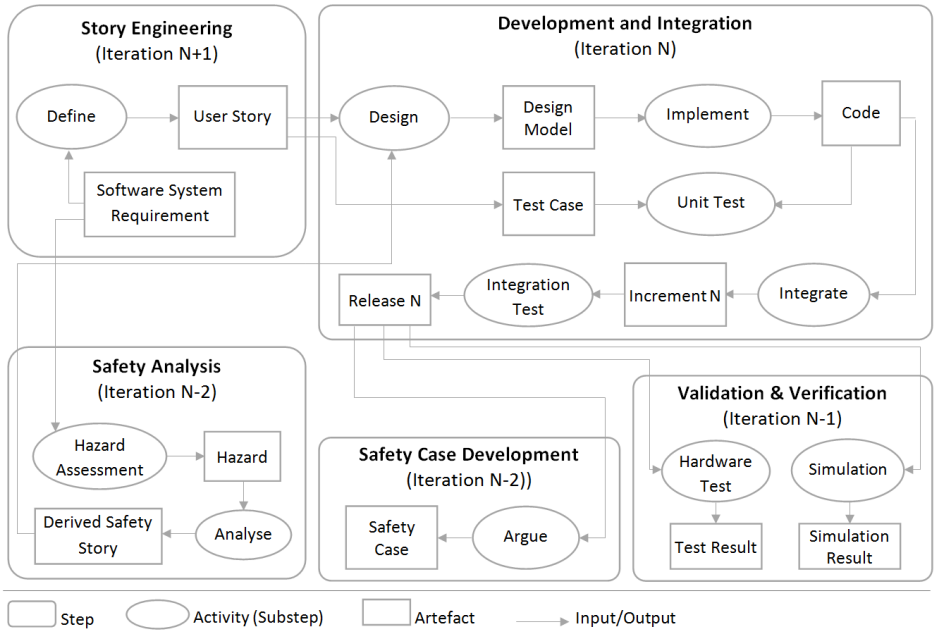


Figure 1.1: Conceptual overview of steps in the development process for IADDS project

which are performed in context of user stories, and documented in *hazard log*. Then, they are examined in more detail using *safety analysis models* and the contribution of various components in the system to these hazards is determined and derived *safety stories* are defined in order to mitigate the hazards. Derived safety stories are similar to user stories with additional properties and will be planned to be implemented in following iterations. Finally, validation and verification (*tests* and *simulations*) and safety assurance activities are performed.

One of the main concerns in safety system development is providing valid and sufficient evidence for the safety argument which aims to show that 'the system is acceptably safe to operate in a particular context' (e.g., civilian airspace) [Hawkins and Kelly, 2009]. Generally, a safety argument consists of a number of strategies which specify how the goals are satisfied. In this context, supporting traceability could be useful to provide required and valid evidence for safety arguments. This is because trace links allow engineers to collect artefacts related to or required for an argument through traversing trace links according to the defined strategy.

In the IADDS project, two main safety arguments are provided in the safety case document: 1. 'implementation is safe' and 2. 'implementation is correct'. In the following, we focus on the first argument and show how it is

supported by evidence and how traceability could help.

To show that the implementation is safe, three strategies are defined:

1. All types of requirements (system, user stories, safety stories) have been satisfied by the implementation
2. All the hazards have been identified
3. All identified hazards have been mitigated (omitted or controlled)

Each of the above strategies need specific evidence. For example, to show that all requirements have been satisfied a safety engineer needs to show that each requirement has been considered, at least, in one design model and the design model has been implemented by code. Also, she has to show that each requirement has been covered by test cases and implementation has passed the related unit tests (test results). In this context, the relationships (trace links) between requirements, design models, code, test cases, test results are essential to prepare the required evidence. Note that a group of requirements are defined in the safety engineering domain as derived safety stories. Thus, to provide the aforementioned evidence, we need to define and keep records of the relationships between derived safety stories (from safety engineering domain) and design models (from development domain): inter-domain traces.

On the other hand, from the traceability perspective, to provide required trace links to answer the above questions (trace queries), traceability information would consist of concepts related directly to them. Therefore, Software System Requirements, User Stories, Safety Stories, Design Models, Code, Test Cases, and Unit Test Results are accumulated in a traceability model. Considering Figure 1.1, it can be observed that most of the required information is already available (except inter-domain traces) and, so, could be extracted from related domain models and put in the traceability model. However, defining and capturing inter-domain links is a challenge in this context.

The above example shows a simple scenario which essentially requires multi-domain traceability. It also briefly outlines some of the challenges of multi-domain traceability.

In chapter 4, examples from this project are used to illustrate the proposed approach in detail to show how the approach helps to provide traceability in order to support safety assurance and certification activities.

1.2 Research Hypothesis

The above discussion and the research context presented in this thesis explore the hypothesis of this work. The definition of the highlighted terms are defined to clarify the hypothesis.

> *In many contexts, traceability is a **multi-domain** concern as it needs to operate across project's different domains. This thesis demonstrates that a **modelling approach** to develop a multi-domain **traceability solution** improves the **precision** and **repeatability** in such solutions. This will be via **automating** the challenging aspects of defining, creating, using, and maintaining **heterogeneous** traceability relationships between various domains in order to support multi-domain **traceability scenarios**.*

The highlighted terms are defined as follows.

Multi-domain Traceability. A traceability solution needs to operate across the project's different domains to effectively deal with various artefacts (documents, models, databases) in different domains and the relationships between them.

Modelling Approach. Using the proposed approach, a model-based traceability solution is developed which uses modelling principles and MDE techniques to carry out related activities and generate artefacts represented as models.

Traceability Solution. The proposed approach will define detailed steps to build a comprehensive traceability solution which consists of a traceability information model, a customised traceability process, and tooling support. The approach considers basic traceability activities (defining TIM, creation, maintenance, and usage of traces) and, therefore, specifies either how the solution support an activity explicitly or how it could be extended with the other techniques.

Traceability Scenarios. Scenarios specify the ultimate purpose of collecting traceability information. They determine traceability goals or requirements. Using the proposed approach, traceability requirements are determined and the solution is developed accordingly; a requirements-driven traceability solution.

Heterogeneous Traceability Relationships. Using the proposed approach, an engineer will be able to manage traceability relationships between different domains' models expressed in heterogeneous formats and contexts.

Precision and Repeatability The proposed approach demonstrates a systematic way which will enable engineers to precisely determine and define traceability information, and capture and maintain it in a repeatable manner rather than in an ad hoc way.

Automation. The proposed approach will provide semi-automatic techniques to build and maintain the solution, in addition to techniques to identify and maintain traceability information.

1.2.1 Research Objectives

The objectives of this thesis are to:

1. Identify and analyse the concept of multi-domain traceability and its specific requirements and challenges.

2. Propose an approach to develop a multi-domain traceability solution which effectively deals with models across different domains.

3. Provide techniques to identify requirements for traceability and develop a project-specific solution accordingly.

4. Support and automate the activity of maintaining the traceability solution to keep it relevant and effective over the time.

5. Provide semi-automatic techniques to specify and capture relationships (trace links) between multiple domains

1.3 Research Results

This thesis proposes a model-based approach to building traceability solutions which effectively operate across multiple engineering domains. The approach introduces a collection of tasks and structures which intend to address the identified challenges and requirements for traceability solutions in multi-domain projects. The proposed approach demonstrates that modelling principles and MDE techniques could help to address current challenges and consequently improve the effectiveness of a multi-domain traceability solution.

A prototype of required tooling to support the approach has been implemented with EMF and atop Epsilon which consists of implementation of the proposed structures (models) and model management operations to support traceability.

The thesis contribution has been validated through using the approach in a large-scale safety-critical project which showed the practicality of the approach and highlighted its benefits in such context. Additionally, a detailed requirements-based evaluation of the approach, presented as a formal argument in the GSN diagram, has demonstrated how the proposed approach fulfils the identified requirements and challenges.

1.4 Research Method

The research presented in this thesis was conducted using the method conceptually illustrated in Figure 1.2, which was effectively based on software engineering processes concepts. The figure shows main phases of the research, activities, and the inputs/outputs of each phase.

In terms of software engineering (SE), there are two levels in a SE process working complementary: *macro* (SE 'in the large') and *micro* (SE 'in the small') levels [Alexander and Maiden, 2004]. Macro level is associated with the overall phases (e.g. exploration and development) and the micro level is concerned with the sequences of low-level activities (e.g. detailed design and implementation).

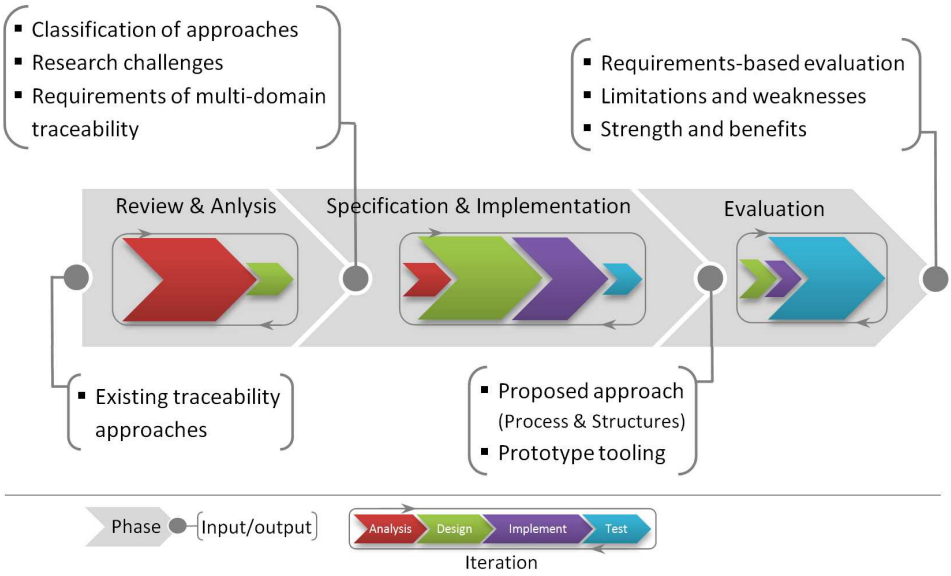


Figure 1.2: Overview of the research method.

As shown in the Figure 1.2, in overall, the research has been done in three phases (the macro level), which mainly specify the primary concern of the research at each time.

1. **Review and Analysis.** In this phase, existing literature for traceability were studied and analysed, to explore the state of traceability approaches in practice, and in multi-domain context, in particular. The analysis led to identification of research challenges and a set of requirements for a practical multi-domain traceability solution addressing the challenges.

2. **Specification and Implementation.** In this phase, based on the result of the analysis phase, a novel approach to develop a multi-domain

traceability solution was proposed, specified and a prototype of the required tooling was implemented accordingly. The set of requirements identified in the analysis phase were used for validating the proposed approach and the implementation.

3. **Application and Evaluation.** In this phase, the proposed approach (process and structures) and the prototype implementation were assessed against the specified requirements. Additionally, the approach was applied in a number of potential contexts (case study) focusing on the feasibility of the approach and its benefits. Accordingly, the strengths and weaknesses of the novel approach were identified.

On the other hand, the research was conducted in an iterative and incremental manner. In this way, the above phases were accomplished through several iterations and their outputs were produced incrementally. In each iteration, the basic software development activities namely analysis, design, implementation, and testing, were carried out sequentially (the micro level). The analysis activity focused on studying existing literature and modelling techniques relevant to this work, and identifying current challenges. Throughout design and implementation activities, the proposed approach was defined and implemented gradually. The output of these activities (a part of the proposed approach) was tested and evaluated in the context of the running example (Section 1.1.1).

The effort and time spent on each activity in each iteration varied depending on the phase in which the research was and the short-term goal. Earlier iterations, which were in the *Review and Analysis* phase, involved intensive analysis and possibly short design activities. In mid iterations (*Specification and Implementation* phase), intensive design and implementation activities were performed as the focus of the research was on defining the approach and implementing a prototype. In later iterations, the approach was applied in case studies and evaluated with respect to the research hypothesis. Therefore, these iterations focused on test activities in comparison to previous iterations, though based on the result of tests, short design and implementation activities might have been carried out too (in the next iteration).

The iterative and incremental approach also allowed us to manage unforeseen issues (e.g. problems, obstacles, missing points) within the time frame of the research plan, through defining a new iteration planned accordingly.

1.5 Thesis Structure

Chapter 2 provides a thorough review of the literature related to this research divided into two parts: software traceability (Section 2.1) and MDE (Section 2.2). Section 2.1.1 describes principles of traceability and Section 2.1.2 outlines the benefits of traceability in terms of its application

for different purposes. In Section 2.1.3, main traceability activities, namely planning and managing, trace creation, trace maintenance, and using traces, as well as the existing literature for each part, are described. Section 2.1.4 introduces the most widely used traceability tools in industry and research-based tools. In Section 2.1.5 limitations and challenges which have hindered the adoption of traceability approaches in practice are identified. Moreover, a short review of traceability in specific domains and different engineering approaches are provided in Section 2.1.6 and Section 2.1.7. Finally, Section 2.2 introduces basic concepts in modelling including models and modelling languages. Then, model management operations are explained in Section 2.2.3 and the concept of *modelling in the large* is introduced and discussed in Section 2.2.4.

Chapter 3 summarises the findings of the review conducted in Chapter 2. First, Section 3.1 provides an of overview existing tracing approaches and a detailed categorisation of them based on the introduced set of parameters (Figure 3.1). Section 3.2 analyses existing traceability approaches regarding the context and scope of this research. Based on this analysis, Section 3.3 synthesises research challenges for traceability in multi-domain context, highlighting those challenges to which this thesis contributes. Finally, Section 3.4 outlines the requirements of a multi-domain traceability solution.

Chapter 4 presents the main contributions of this thesis. Section 4.1 introduces a three-step method to define a project-specific TIM which includes determining traceability goals, identifying related concepts in the project, and finally representing the TIM in a formal way. In Section 4.2, it is explained how to investigate existing models in a project to identify available traceability-related information which could explicitly or implicitly support traceability. Section 4.3 presents a systematic approach to locate and extract traceability-related information from other models in the project and generate a project-wide traceability model. Section 4.4 discusses how the traceability model is maintained when the relevant models change and evolve over the time. Finally, in Section 4.5, the Traceability Analysis Language (TAL) is presented and it is discussed how it facilitates traceability analyses.

Chapter 5 presents a prototype of the tooling required to support a traceability solution developed based on the approach presented in this thesis. In Section 5.1 an overview of the infrastructure used in the implementation is provided. Section 5.1 explains the prototype implementation in detail and highlights its main parts and features.

Chapter 6 applies the proposed approach in a large-scale safety-critical project, the EUR RVSM Programme. The case study explores the extent to which the proposed approach is beneficial, particularly in terms of multi-domain traceability scenarios.

Chapter 7 evaluates the approach presented and explained in this thesis to explore whether the proposed tasks and structures are effective in devel-

oping a multi-domain traceability solution. In Section 7.1, the approach is basically assessed against the identified requirements. Section 7.2 outlines the finding of the EUR RVSM case study. In Section 7.3, the papers representing the outcomes of this research are introduced. The limitations and shortcomings of the approach are identified and discussed in Section 7.4. Finally, Section 7.5 presents a discussion of using the proposed approach in practice.

Chapter 8 summarises the main contributions of the research, discusses them in the context of the research hypothesis, and finally suggests areas of future work.

2

Background and Literature Review

This chapter provides a review of the research relevant to the work presented in this thesis. The review consists of two main sections. The first section discusses software traceability and outlines its principles and benefits. It describes the main traceability activities, namely planning and managing, trace creation, trace maintenance, and using traces, as well as the related literature. Additionally, it presents briefly some of the most widely used traceability tools and it identifies some of the limitations and challenges which have hindered the adoption of traceability approaches in practice.

The second section describes Model Driven Engineering (MDE), whose principles are used extensively in this work. This section starts with a brief introduction to basic concepts in modelling including models and modelling languages. Then, model management operations, especially those that are explicitly used in this work, are explained. Finally, the concept of *modelling in the large* is introduced and discussed in addition to a brief literature review.

2.1 Traceability

Traceability is a topic of great interest in the software engineering domain in general, and in Requirements Engineering (RE) and MDE in particular. The role of traceability was recognised in the NATO working conference held in 1968 to discuss the problems of software engineering [Gotel et al., 2012b]. Afterwards, it has been considered as a quality attribute for software development because of the benefits provided by supporting traceability, such as ensuring other qualities of the software (e.g. understandability), change impact analysis, and coverage analysis [Gotel and Finkelstein, 1994; Lindvall and Sandahl, 1996; Ramesh and Jarke, 2001].

2.1.1 Terminology and Concepts

Historically, traceability has started as an area of requirements engineering, as noted by a study of the existing fundamental literature of traceability. However, nowadays, traceability is considered as a method to manage traces of any types of artefact (other than requirements) [Aizenbud-Reshef et al., 2006; Winkler and Pilgrim, 2010] and as an instrument to integrate such links into development process methods [Lago et al., 2009]. Therefore, it overlaps with different domains of interest, including RE, MDE, knowledge engineering, software project and process management. This makes it difficult to clearly locate the traceability in the software engineering research landscape. Accordingly, there is no single general definition for traceability, and different definitions and terminology are provided regarding the context or domain in which traceability is used or considered.

2.1.1.1 Definition

One of the earliest formal definitions of traceability is found in the IEEE's Standard Glossary of Software Engineering Terminology [IEEE, 1990] which defines traceability as

> "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 [...]"

This general definition has been adapted based on context; hence different definitions of traceability have been provided in the literature. Significant traceability research has been conducted by the requirements engineering community, most definitions of traceability explicitly refer to requirements traceability. In this respect, one of the key and early definitions of traceability has been given by [Gotel and Finkelstein, 1994] in the context of RE:

> "Requirements traceability refers to the ability to describe and follow the life of a requirement in both forward and backward direction, i.e from its origins through its development and specification, to its subsequent deployment and use, and through period of ongoing refinement and iteration in any of these phases."

This definition focuses on requirements and the relationships between them and other artefacts of software development process. Additionally, it focuses on relations between a requirement and the associated artefacts which are subsequently created. Therefore, it does not cover relations between artefacts that are not in a predecessor-successor relation or the relations between artefacts which are created simultaneously. However, these

types of relations exist in software development projects and are useful to be captured and recorded.

Another definition of traceability is offered by [Pinheiro, 2003] which is also produced in the RE domain and focuses on requirements and their relations with other development artefacts. However, it covers more generic relations rather than just predecessor-successor relations between artefacts. According to their definition, requirement traceability

> "refers to the ability to define, capture and follow the traces left by requirements on other elements of the software development environment and the traces left by those elements on requirements."

As mentioned earlier, over the past years, traceability has gained in importance, and traceability topics have become subject to research in many other areas of software development. For example, [Object Management Group, 2003] provides a very specific definition of traceability in the context of MDE and model transformations. Accordingly,

> "a trace records a link between a group of objects from the input models and a group of models from the output models. This link is associated with an element from the model transformation specification that relates the groups concerned."

Although this definition is provided in the MDE domain, it is not general enough to cover different types of relations and traces existed in the context of MDE. This is because it particularly focuses on model-to-model transformations and low-level traces –between objects of models. [Paige et al., 2008] define traceability in their work to provide a classification for traceability in MDE and identify traceability as "the ability to chronologically interrelate uniquely identifiable entities in a way that matters". Although they do not limit the level of traces and type entities, they limit the traces to those that exist between traceable entities which can be only chronologically interrelated. So, it is a partial order rather than a total order.

However, there are definitions of traceability which try to provide a more generic definition of traceability, though they are provided in a particular research community. For example, in the MDE domain, [Aizenbud-Reshef et al., 2006] follow the IEEE's definition of traceability and define traceability as "any relationship that exists between artefacts involved in the software engineering life cycle". Although they do not focus on any specific domain and define traceability with respect to products or artefacts of the development process, the terms *product* or *artefact* generally refer to coarse-grained outputs of development activities such as documents or models. So, although this definition is generic to software development, it does not explicitly consider potential traces between small pieces or parts of those large products or artefacts.

[Spanoudakis and Zisman, 2004] provide an extended definition in the context of software traceability, which contains both artefacts produced during the development process, stakeholders, and artefact rationale. However, it does focus on artefacts and not their parts, as the definitions mentioned above. It defines software traceability as

> "the ability to relate artefacts 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 artefacts and the rationale that explains the form of the artefacts."

Finally, in one of the recent studies on traceability terminology and concepts, [Gotel et al., 2012b] define traceability at the most fundamental level as "the potential for *traces* to be established and used" within the context of a broader *traceability strategy*. The strategy comprise those decisions made in order to determine the stakeholders and system requirements for traceability and to design a suitable traceability solution. They define *trace link* and *trace artefacts*, respectively, as a "single association forged between two trace artefacts" and "a traceable unit of data (e.g. a single requirement, a cluster of requirements, a UML class, a UML class operation, a Java class, or even a person)". In this context, traceability is an attribute of an artefact or a collection of artefacts which makes artefacts to be traceable. As mentioned above, they aim to provide a fundamental definition; hence, the definition is generic in comparison to the aforementioned ones and could cover all types of information which would be traced, with various granularity, at different levels, and with different semantics.

All of the above mentioned definitions help to reach to a general understanding of the concept of traceability in software engineering, even though, some of definitions focus on specific context, characteristics, or properties. In the context of this thesis, traceability is about *establishing and using traces in order to support arbitrary stakeholders' needs* which might have been defined in any context, such as requirements management, certification process, and quality assurance. Therefore, the scope of this thesis requires a definition of traceability which could cover all possible scenarios. We found the definition of traceability provided in [Gotel et al., 2012b] as a suitable definition which properly meets the requirements of this research, and hence we use the following as the working definition:

> "**Traceability** is the ability to define, create, maintain, and use traces between traceable units of data in the context of traceability requirements or goals."

Effective traceability is provided in the context of a traceability strategy. The core part of a traceability strategy is a *traceability solution* which is defined, designed, and implemented for a particular project or situation along

with required traceability tooling [Gotel et al., 2012b]. A traceability solution comprises a *traceability information model (TIM)*, a *traceability process*, and *traceability tools*.

A *traceability information model (TIM)*, which is also called a traceability scheme or traceability metamodel in the literature, is an abstract expression of the intended traceability for a project. It is usually represented as a graph defining the detail required to record in order to address the anticipated traceability-related queries and software and system engineering activities and tasks that traceability supports. Usually, a TIM defines the permissible artefact types, the permissible trace link types, and the permissible trace relationships in the project [Ramesh and Jarke, 2001; Gotel et al., 2012b]. It may also capture additional information such as the cardinality of the trace artefacts associated through a trace link, the primary trace link direction, the purpose of the trace link (i.e. the link semantics), and the location of the trace artefacts [Mäder et al., 2009a].

A *traceability process* defines the particular sequence of activities to be employed to establish traceability and render it usable for a particular project, along with a description of the responsibilities and resourcing required to undertake them as well as their inputs and outputs. The activities are referred to as the *traceability activities* in the literature, and include activities for identification, representation, maintenance, and utilisation of traces [Pinheiro, 2003]. The traceability process defines how to undertake a traceability strategy and traceability activities during the life of a project.

2.1.1.2 Basic Types of Traceability

Tracing can be considered for various purposes and so is performed based on different foundations, such as based on logical interrelations among artefacts or based on temporal dependency between artefacts. The most common types of traceability, in the requirements traceability literature, are *forwards* and *backwards* traceability, *horizontal* and *vertical* tracing, and *pre-RS* and *post-RS* traceability, which are illustrated in Figure 2.1. There are also other conceptual classifications including *functional* and *non-functional* traces, *product-related* and *process-related* trace links, and *implicit* and *explicit* traceability. These terms mainly provide a conceptual (high-level) classifications for trace links. Additionally, they are not exclusive or orthogonal, and a trace link can be categorised in more than one group at the same time.

The ANSI/IEEE Standard 830–1984 [IEEE, 1984] has introduced the terms *backward* and *forward* traceability. The forward and backward direction pertain to the logical flow of the software and the systems development process. Backward traceability refers to the ability to follow the traceability links from a specific artefact back to its sources from which it has been derived. Forward traceability refers to following the traceability links to the

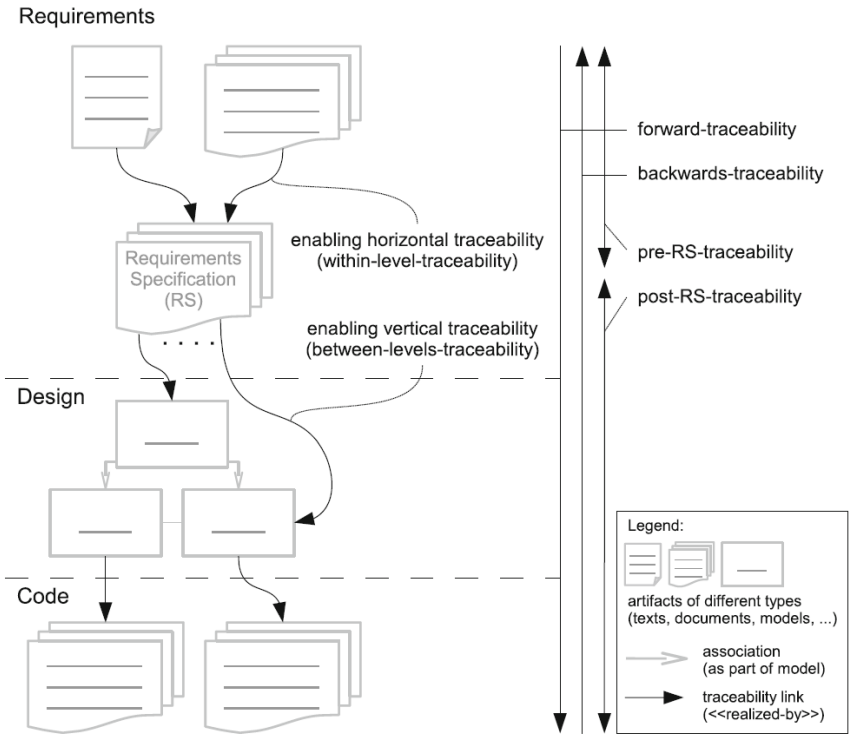


Figure 2.1: Basic types of traceability [Winkler and Pilgrim, 2010]

artefacts that have been derived from the artefact under consideration.

[Ramesh and Edwards, 1993] have introduced the distinction between *horizontal* and *vertical* traceability. These terms differentiate between traceability links between artefacts belonging to the same project phase or level of abstraction, and links between artefacts belonging to different ones. Vertical tracing is used to trace artefacts at different level of abstraction to accommodate lifecycle-wide or end-to-end traceability. Horizontal tracing is about tracing artefacts at the same level of abstraction. These two types can employ both forward and backward tracing. [Antoniol et al., 2001] briefly relates horizontal and vertical traceability to the iteration- and increment-dimensions of the development process, respectively.

[Gotel and Finkelstein, 1994] have introduced and emphasized the classification in *pre-requirements specification (pre-RS)* traceability and *post-requirements specification (post-RS)* traceability. Pre-RS traceability is concerned with traces occurring during elicitation, discussion, and agreement of requirements until they are included in the requirements specification document. Post-RS traceability is concerned with the stepwise implementation of the requirements in the design and coding phases.

[Pinheiro, 2003] has divided traces into two different categories:

1. *Functional traces.* These are created by transforming one artefact into another using a defined rule set.

2. *Non-functional traces.* They refer to traces of informal nature and include reason, context, decision, and technical aspects.

[Ramesh and Jarke, 2001] introduce *product-related* and *process-related* trace links. Product-related links describe properties and relationships of traceable artefacts independent of how they were created. Process-related traces are captured only by looking at the history of actions taken in the process itself and cannot be recovered from the product relationships.

[Mäder et al., 2007] categorises trace links into two groups: *explicit* and *implicit* traces. They describe explicit traceability as any explicitly expressed relationship between entities, while implicit traceability is considered to be any interrelationship between entities which is implied but not explicitly represented. Implicit traces result from an inherent relationship between the traceable items [Paige et al., 2008]. For example, a link between two elements, which belong to different models and have the same name, is considered to be an implicit link. However, current research mostly focuses on and investigates explicit traceability as it requires additional effort to be provided [Mäder et al., 2007].

Finally, [Costa and da Silva, 2007] state that there are two viewpoints in creating traces: the *dependency viewpoint* and *generative viewpoint*. The dependency viewpoint identifies traces describing some implicit or explicit semantic relation or dependency between them. By contrast, the generative viewpoint is interested in the relationships between the generated artefact and the one which was used for the generation.

2.1.1.3 Traceability Classifications

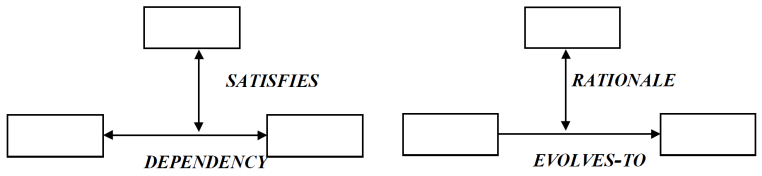
Although many approaches and techniques have been introduced and developed to address different aspects of traceability, majority of the early research concentrates on interpretation and classification of traceability relations. This is because richer semantics in the traceability relationships, rather than abstract relationships, can increase the benefits of using traceability [Dick, 2005].

A traceability classification represents a set of trace link types identified for a specific domain or context. The existing well-known traceability classifications (e.g. [Ramesh and Jarke, 2001; Spanoudakis and Zisman, 2004]) are mainly introduced in the context of requirements traceability, and have been introduced based on the properties and the intention of a specific study. For example, some of the classifications are based on the types of the related artefacts, while others are based on the use of traceability in supporting different requirements management activities [Spanoudakis and Zisman, 2004].

However, there are other traceability classifications for other domains such as [Rummler et al., 2007] for business applications and [Olsen and Oldevik, 2007] for different uses of traceability in model-to-text transformation.

In the following the most well-known traceability classifications and, then, existing approaches to define case-specific classifications are introduced and assessed.

[Ramesh and Jarke, 2001] conducted a thorough study in the domain of requirements engineering. They recognise two types of traceability users: low-end and high-end users. Low-end users consider traceability simply as a mandate from the project sponsors or for compliance with standards. These users describe the various project interdependencies using simple traceability classifications to provide links from initial requirements to actual system components. High-end users consider traceability as a major opportunity for customer satisfaction and knowledge creation throughout the system development process. They use much richer traceability classifications, which enable them to perform richer analysis and reasoning on traces, including requirements management, design allocation, compliance verification, and rationale management. Considering the two types of users, Ramesh and Jarke classify traceability links into four main groups named satisfies, depends on, evolves to, and rationale links, shown in Figure 2.2. They also state that the first two link types are product-related while the other ones are process-related. Then, they provide a considerable number of fine-grained link types for each group regarding the analyses that low and high-end users perform.



(a) Product-related traceability links (b) Process-related traceability links

Figure 2.2: Traceability links classification [Ramesh and Jarke, 2001]

In Figure 2.2a, the high-level object (e.g. a requirement, a standard, a policy, or complex design object) defines some kind of constraint or goal which should be *SATISFIED* by one or more lower-level objects. The shared constraint or goal to be satisfied also implies a *DEPENDENCY* between lower-level objects. Generalization and aggregation abstractions (i.e. configuration of complex objects) are special kinds of dependencies. low-end traceability users tend to be characterized by relying mostly on these two link types.

Figure 2.2b shows the two kinds of process-related links: evolution and rationale links. The important difference is that the evolution link has a temporal direction: the left lower-level design object *EVOLVES-TO* the right one through some kind of action whose *RATIONALE* is captured in the higher-level object. Advanced traceability users typically have a higher share of link types belonging to this category.

Another classification focusing on requirements traceability is proposed by [Pohl, 1996b]. In this work, 18 different trace link types are identified and organised into five different categories. These categories are the following:

- Condition Link Group: this group consists of the relationships between requirements and the various constraints associated with them.
- Documentation Link Group: this group includes relationships between different types of software documentation and requirements.
- Abstraction Link Group: this category includes relationships representing abstraction between requirements, such as generalisation or refinement.
- Evolutionary Link Group: this group consists of replacement relations between requirements.
- Content Link Group: this category includes trace links which denote comparison, conflict and contradiction between requirements.

[Spanoudakis and Zisman, 2004] conducted a thorough survey and organised the available different types of traceability relations into eight main groups. These groups are the following:

- Dependency links describe a relationship in which an element e_1 depends on an element e_2, if its existence relies on the existence of e_2 or if changes in e_2 have to be reflected in e_1.
- Generalisation/Refinement links are used to identify decomposition of entities, composition of entities or refinements.
- Evolution links are used to describe the evolution relation between two artefacts. When artefact e_1 evolves to artefact e_2, then artefact e_1 is replaced by e_2.
- Satisfiability links signify the compliance of an artefact to another one.
- Overlap links describe a relationship in which artefacts e_1 and e_2 refer to common features of a system or of its domain.
- Conflict links describe an incompatibility or an inconsistency relation between two artefacts.

- Rationalisation links are used to represent the rationale behind the creation and evolution of artefacts.

- Contribution links represent the association between artefacts and stakeholders, who have contributed to their creation.

In addition to the aforementioned trace link categories, [Spanoudakis and Zisman, 2004] present a very detailed description (in the form of a matrix) of the various trace link types that have been proposed in the literature and the types of software artefacts that these links interrelate. Moreover, associations between stakeholders and artefacts are presented. Based on their matrix description, they observe that most approaches focus on types of traceability relations that relate requirements specifications, as well as requirements with design. The number of approaches that focus on the links between code and requirements or between code and design artefacts are far fewer. This is attributed to the fact that initially, traceability was a concept very closely related to requirements, as well as to the fact that establishing trace links between code and other development artefacts is difficult. Finally, [Spanoudakis and Zisman, 2004] identify the lack of standard semantics for the various trace link types found in the literature and they highlight the need for richer trace link semantics, since it is a precondition for development.

In contrast to these flat classifications, hierarchical structure of link types is also a common approach in classification. The SysML specification [Object Management Group, 2010a], describes its different link types –DERIVE, VERIFY, COPY, SATISFY, and (implicitly) REFINE– as refined dependencies. The most recent and extensive hierarchical classification has been created by [Dahlstedt and Persson, 2005].

[Dahlstedt and Persson, 2005] propose a classification of interdependency trace links between requirements. This classification is illustrated in Figure 2.3. In the proposed classification, requirements interdependencies are grouped into two main categories: structural and cost/value interdependencies. Structural interdependencies are concerned with the fact that given a specific set of requirements, they can be organised in a structure where relationships are of a hierarchical nature as well as of a cross-structure nature. On the other hand, cost/value interdependencies are concerned with the costs involved in implementing a requirement relative to the value that the fulfilment of that requirement will provide to the perceived customer/user. [Aizenbud-Reshef et al., 2005] has refined this classification to introduce more fine-grained *dependency* relations, such as CLASS-IMPORT-CLASS or METHODINVOCATION-CALLS-METHODDEFINITION.

Another classification of trace link types is proposed by [Von Knethen and Paech, 2002]. This classification investigates the technological aspects of traceability in five different dimensions. One of the proposed dimensions

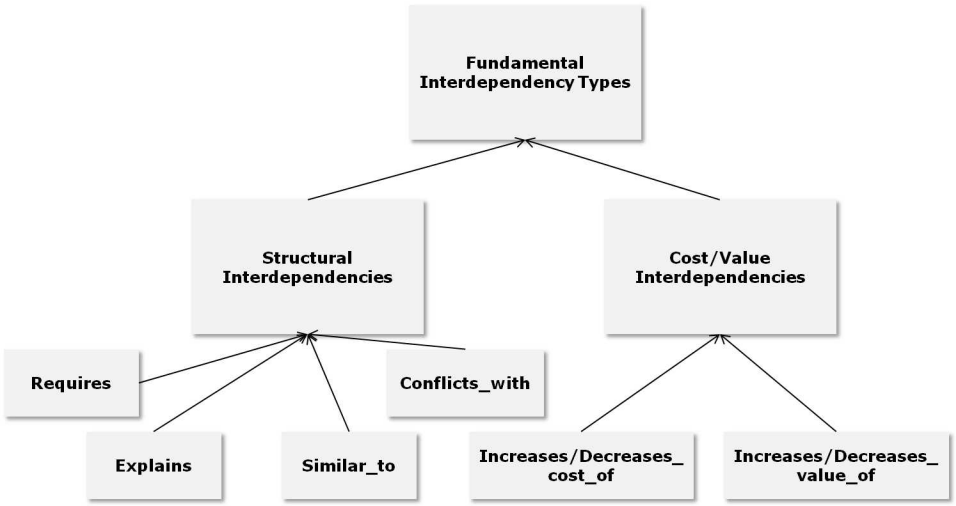


Figure 2.3: Requirements Interdependencies Classification [Dahlstedt and Persson, 2005]

for classifying traceability approaches is the types of relationships they support. That is, what kinds of relationships are described by trace links, what is their direction, what are their attributes, what is the setting of those relationships and finally how are these relationships represented? From the literature the authors identify three general kinds of relationships: (1) relationships between documentation entities at different levels of abstraction, (2) relationships between documentation entities on the same abstraction level, and (3) relationships between documentation entities of different versions of the same software product.

The aforementioned traceability classifications are not precisely defined and they are subject to the view and interpretation of the reader. Additionally, the categories contains unclear and informal definitions which adds more difficulty to use them. For example, it is hard to tell, if a refines link (as in Requirement-refines-Requirement) belongs to one of the classes of evolutionary, containment, (within-level) refinement, or dependency links mentioned by [Von Knethen and Paech, 2002]. Furthermore, most of the link types described in those classifications are binary, something which contrasts with the fact that trace links can very often be n-ary [Munson and Nguyen, 2005]. [Dick, 2005] advocates the need for even more complex trace link structures such as alternatives and conjunctions.

The above mentioned classifications advocate that different stakeholders in different domains have different traceability needs. Hence, different traceability classifications are needed, though they all aim to identify a set of predefined trace link types for a given domain or case. In contrast to this

usual approach, recent researchers have recognised the importance of building case-specific traceability solutions [Mäder et al., 2009a; Aizenbud-Reshef et al., 2006], which consequently leads to defining trace link types for a particular situation. However, there are limited number of studies providing a systematic approach to define case-specific traceability classifications. This is because most of the research in this context simply indicate that users can define their own trace link types.

[Paige et al., 2008] introduces a simple process for building and maintaining case-specific traceability classifications. The process is called the Traceability Elicitation and Analysis Process (TEAP). It is derived from a process developed in [Chan and Paige, 2005] for elicitation and understanding different forms of model-based contracts. The aim of TEAP is to elicit and analyse traceability relationships in order to determine how they fit into a traceability classification. When applying TEAP, engineers typically bootstrap from a simple traceability classification or metamodel, and iteratively and incrementally refine the classification through a number of TEAP cycles. Each cycle in the TEAP enriches the existing classification in terms of one or more key attributes of interest. The classification developed is a living document which is extended iteratively and incrementally over the course of the project. TEAP basically provides a process to develop a classification, so it should be implemented as a concrete design (usually as a metamodel) in the context of a tracing tool or approach.

2.1.2 Traceability Applications

The value of traceability lies in the software and system engineering activities and tasks that the information provided through such interrelations can enable. For example, it can provide visibility into required aspects of the software and system development process and contribute to a better understanding of the software and system under development.

The benefits of traceability are thoroughly investigated in the literature and several applications of it are identified and introduced in different domains and context ([Gotel and Finkelstein, 1994; Ramesh and Edwards, 1993; Von Knethen and Paech, 2002; Winkler and Pilgrim, 2010; Watkins and Neal, 1994; Wieringa, 1995]). In this section, the importance of traceability in various aspects of software engineering is briefly explained.

Requirements Management

Traceability can be used for requirements prioritization, classification, and planning. Requirements can be traced back to goals and mission statements, and then can be rated and categorized in terms of goals, risks, and priorities [Ramesh and Edwards, 1993; Sommerville, 2007], which in turn help in planning releases, iterations, and development cycles.

Change Management and Impact Analysis

Requirements traceability supports the impact analysis of changes by identifying the work products affected by a change and hence ensuring that the requirements, design, implementation, and related tests are all evaluated for impact [Gotel and Finkelstein, 1994; Ramesh and Edwards, 1993]. Impact analysis, or determining the impact of a change on all of the product artefacts, can be done efficiently when traceability, and automated traceability in particular, has been implemented end-to-end. Moreover, traceability information can help the engineers make decisions about whether such changes should be introduced and with which priority (change management) [Spanoudakis and Zisman, 2004].

Validation and Verification

Trace links can provide the basis for performing validation and verification analysis [Ramesh and Edwards, 1993; Watkins and Neal, 1994]. By validation, then mean the analysis undertaken to ensure that a system fulfils its intended purpose. On the other hand, by verification it is meant the analysis undertaken to ensure that a system is built according to its specification.

Testing

Traceability relationships can be used to check the existence of appropriate test cases for testing different requirements and to guarantee that all requirements have been covered by all tests [Gotel and Finkelstein, 1994; Watkins and Neal, 1994]. The results of such analysis usually provide input to software inspection and auditing activities [Von Knethen and Paech, 2002]. Furthermore, traceability can be used to relate possible solutions for failed tests to the actual problems [Arkley and Riddle, 2005].

Reuse

Traceability can also be used to identify reusable artefacts. During the development of new system, requirements already have been implemented can be identified and reused together with their related design and implementation [Ramamoorthy et al., 1988].

System Understanding

One of the main uses of traceability is for understanding various artefacts in reference to the context in which they were created or in reference to other

artefacts related to them [Spanoudakis and Zisman, 2004]. There are several approaches to use traces in this context. Traces can help to understand systems from different points of view [Sabetzadeh and Easterbrook, 2005], to pull together fragmented information [Gotel and Finkelstein, 1994], or to identify crosscutting concerns [van den Berg et al., 2006]. They are also useful to follow design and implementation decisions [Ramesh and Edwards, 1993].

Audit and Certification

Traceability is an explicit requirement of many quality standards, especifically for certification purposes. The Capability Maturity Model Integration (CMMI) [Chrissis et al., 2006] requires the use of requirements traceability for CMMI Maturity Level 2 and has adopted the Institute of Electrical and Electronics Engineers (IEEE) definition for traceability [IEEE, 1990]. Requirements traceability is an explicit requirement of the TL 9000 telecommunications requirements handbook [QuEST, 1998]. The aeronautics industry software development standards DO-178B [RTCA and EUROCAE, 1992] and DO-254 [RTCA, 2005] also require the use of traceability.

Software Project Management

Another common use of traceability is for software project management purposes. Trace links can be used for assessing the development process, because they comprise a log of events occurred during the development process [Gotel and Finkelstein, 1994]. Moreover, they can also be used to monitor the status of the requirements during development as they are implemented and tested [Lago et al., 2009]. Finally, [Jarke and Pohl, 1992] proposes the use of traceability to identify and reuse best practices during software development.

2.1.3 Traceability Activities

As mentioned before (Section 2.1.1), in order to provide traceability in a project, a series of activities are performed in the context of a broader traceability strategy. Although these activities vary in the literature and are named differently (i.e. [Von Knethen and Paech, 2002; Pinheiro, 2003; Spanoudakis and Zisman, 2004; Gotel et al., 2012b]), they cover a common life-cycle for traces in any traceability solution including

- Defining the entity types to be traced and the relationships types between them, representation of the captured traceability information in the form of data structures as well as their visualisation and storage
- Discovering trace links and recording them in the given format

– Using the trace links according to the purposes for accumulating traceability data

– Maintaining the integrity of the trace links while the entities continue to change and evolve

Traceability activities are supportive activities performed as part of all steps in the software development process [Palmer, 1999] and there is no simple relation between activities of a software development process and these activities. Further, traceability activities are not dependent on any particular software process model. By and large, traceability activities are usually interleaved with one another as well as with the other activities of software development in a way which depends on the tracing needs of a project. However, at present, there is no standard nor a set of guidelines to practitioners on how to use the various traceability approaches within a standardised software development process.

[Gotel et al., 2012b] propose a generic traceability process model which comprises four distinctive activities: planning and managing traceability, creating traceability, maintaining traceability, and using traceability. The process model is illustrated in Figure 2.4. One of the main properties of this model is that it highlights the role of a traceability strategy in providing effective traceability, which is not considered explicitly in other studies. In this context, it defines *Planning and Managing Traceability* through which high-level and fundamental decisions are made such as: determining needs, definition of a TIM, and planning other traceability activities. While, other studies have considered the definition of a TIM and issues with representation, visualisation, and storing information, in the context of an activity usually called representation [Pinheiro, 2003]. The other three activities – creating, using, and maintaining traceability– are almost defined similarly in the literature (possibly with different names) for the same intention.

In the following sections, the identified activities, based on the generic traceability process model, are discussed in more detail.

2.1.3.1 Planning and Managing Traceability

Planning and managing traceability is an umbrella activity (in the software engineering terminology), in comparison to other traceability activities, which is carried out during the life of a project in order to keep the traceability solution relevant and effective over the time. The activity consists of a number of preliminary tasks to define, build, and maintain a traceability solution including determining needs and resource constraints, defining the TIM, planning other traceability activities, providing the required tooling, and assessing the solution [Gotel et al., 2012b]. In the context of MDE, this activity can be seen as part of a general setup of all required tools and artefacts, which is called the infrastructure by [Bézivin et al., 2006], in order to

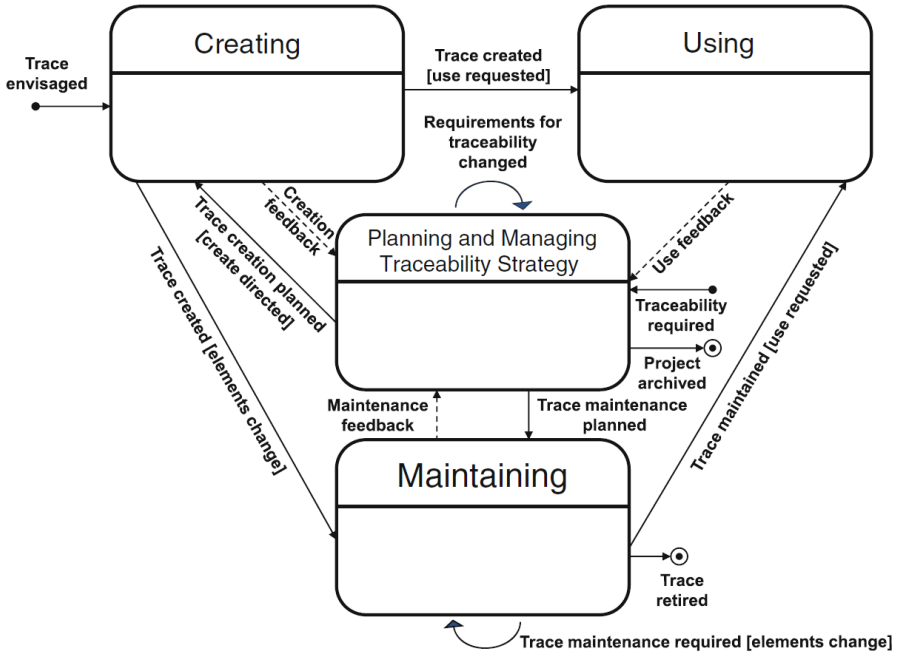


Figure 2.4: A generic traceability process model [Gotel et al., 2012b]

perform automated transformation. [Walderhaug et al., 2006] also describes this step as defining a trace metamodel.

Determining Needs and Resourcing. Building a traceability solution to service all needs is unlikely to be cost-effective, as resources are generally limited [Mäder et al., 2009a]. There are wide variations in the format and content of traceability information across different system development efforts. For example, the types of links of interest to a system tester may be different from those of interest to a system designer. A system tester may be interested in finding out what are the system components that are affected by a requirement while a system designer may be interested in finding out how the components of the system are affected by requirements.

Accordingly, determining whose needs to satisfy, and so which traceability-enabled activities and tasks to facilitate, is a precursor to any discussion about types of trace artefacts, trace links, and tracing mechanism; traceability is requirements-driven. [Gotel et al., 2012a], in "The Grand Challenge of Traceability (v1.0)", which presents a vision of traceability in 25 years, highlight and discuss the role of these decisions, and required strategies and practical guidance to make them, in order to provide a 'fit-to-purpose' traceability solution.

[Heindl and Biffl, 2005] suggest value-based requirements traceability (VBRT) to perform tracing based on prioritised requirements and hence identify which traces are more important and valuable than others. [Lago et al., 2009] present an approach to customize traceability to the situation at hand. Instead of automating tracing, or representing all possible traces, they scope the traces to be maintained to the activities stakeholders must carry out. They define core traceability paths, consisting of essential traceability links required to support the activities.

The importance of definition and specification of the needs for traceability is also acknowledged in the studies which focus on the additional cost of traceability and suggest techniques to control it (e.g. [Ingram and Riddle, 2012; Mäder et al., 2009a; Lago et al., 2009; Egyed et al., 2007; Aizenbud-Reshef et al., 2005]). A common approach, among these studies, is to define the entity types to be traced and the relationship types between them regarding a specific usage scenario of traceability. Accordingly, they indicate that it is essential to determine traceability requirements to establish a cost-effective traceability solution.

However, to the extent of our knowledge, there are few studies which introduce practical guidance on how to determine these needs. [Mäder and Cleland-Huang, 2010] utilises a goal-oriented approach to identify long-term strategic traceability needs and required trace queries to satisfy them. The techniques used to determine traceability goals and construct the TIM has been established based on the systematic Goal-Question-Metric (GQM) approach proposed by [Basili and Caldiera, 1994]. The approach demonstrates how systematically high-level goals are refined (translated) into low-level queries and then relate to project's information.

Defining the TIM. One of the main tasks carried out in the context of planning and managing traceability is defining the TIM, the core component of any traceability solution. A TIM defines the information that should be captured and recorded: the detailed specification of the elements to be captured, granularity, trace link types, and meta-data. Traceability information models are widely investigated in the traceability research area from different perspectives. Accordingly, there are several studies which discuss the importance of such models, define traceability information models for a given domain or context, or propose different approaches to define a TIM (e.g. [Ramesh and Jarke, 2001; Spanoudakis and Zisman, 2004; Mäder et al., 2009a]). Because of the importance of the TIM and the available literature, these studies are introduced separately in the Section 2.1.3.2.

Planning Activities and Tooling. In this sub-activity, a traceability process is defined. Also, it is specified when and how other traceability activities are performed. Additionally, the required tools to support these activities

are defined and provided to users. This sub-activity, as a pre-requisite for traceability, is rarely considered in existing traceability approaches. Traceability activities and tools are implicitly defined and provided in the context of a particular approach. Whilst most of the available approaches do not acknowledge that a practical approach needs to either define a complete process or specify how it is integrated with other approaches [Matragkas, 2011], which will be discussed more later in Section 3.2.

Nevertheless, [Espinoza and Garbajosa, 2008a] explicitly consider the definition of a traceability process as they believe that traceability practices are implemented by a traceability methodology. Following the terminology of software development methodologies, they introduce the Traceability metaModel for methodology definition (TmM), at the metamodel level in software methodology terms [ISO/IEC, 2007]. TmM provides the baseline for the systematic and formal definition of a traceability methodology. It includes three aspects: the process to follow, the intermediate and final products, and the resources, e.g. roles involved in the traceability process. This way, users define a customised traceability methodology (solution) by instantiating the introduced metamodel.

Assessing the Solution. Another purpose of planning and managing traceability is to ensure that the traceability is established as planned, and can adapt to remain effective as needs evolve and as a project's artefacts change. This is because the cost-benefits of traceability are not reached if data does not achieved an 'acceptable quality', which varies between different project [Ingram and Riddle, 2012]. Accordingly, assessing the quality and the execution of the traceability solution is a critical part of the traceability strategy for a project [Gotel et al., 2012b]. Additionally, the traceability provided has to fit the end users' contexts and needs. A feedback loop is essential to improve and adapt the solution based on the result of assessing the quality of the traces with respect to the task or activity for which traceability is required –when traces are used. In this context, [Gotel et al., 2012a] recommend a feedback-driven learning system, which will adapt the traceability that is established to fully address its end users' evolving task contexts and needs, as a component of their traceability process to support a fit-to-purpose traceability solution.

Although, in theory, assessing the solution is introduced as an essential task to keep the solution effective, it has not been fully considered in practice, including in existing traceability approaches. There are few studies which have introduced some quality attributes for a traceability solution (as a whole or for a specific part) and, in some extent, talked about how they are defined and assessed (e.g. [Ingram and Riddle, 2012; Egyed et al., 2007; Espinoza et al., 2006]). [Espinoza et al., 2006] introduce a set of non-functional requirements for a traceability solution, including usability,

feasibility, reliability, functionality, and efficiency, and state that a traceability schema defines a set of metrics to verify the quality requirements of a traceability implementation. However, they suggest to apply research approaches in requirements engineering metrics in this context to verify the quality of the traceability solution (mainly for the collected data). Nevertheless researchers largely agree that the traceability information model (the core element of a solution) needs to specify validation requirements (constraints) for the captured traceability information which should be retained all the time [Gotel et al., 2012a; Mäder et al., 2009a; Ramesh and Jarke, 2001]. Although these constraints usually focus on the data and not the whole solution, they are useful to ensure about the quality of the solution partially.

2.1.3.2 Defining and Representing the TIM

Traceability information models (TIMs) are considered in several studies and are called largely by various names including metamodels, schemes, or reference models, depending on the field in which they are used. These studies proposed different approaches, reference models, frameworks, and classifications of different types of traceability regarding their interest.

Such models provide guidance as to those elements to collect and those relations to establish to support traceability needs [Gotel et al., 2012b]. An important point with these models is that they specify the semantics of traceability through defining trace link types. A trace link could be viewed differently based on the stakeholders' perspective. This is because the meaning (semantics) of a trace link is determined based on how and for what intention the link is used by the user [Ramesh and Jarke, 2001]. For example, in the case of a requirement-to-component trace link, the system designer might consider the requirement as a constraint on the system design, while a system tester might be interested in using the traces to perform coverage analysis in order to find out if every requirement is implemented by at least one component.

As mentioned in Section 2.1.1.3, early research has been mainly concerned with the study and definition of different types of traceability relations. Recent studies, such as [Aizenbud-Reshef et al., 2005; Mäder et al., 2009a; Lago et al., 2009; Matragkas, 2011], acknowledge that effective traceability requires that the types of trace artefacts and trace links have to be defined based on a particular project situation. Distinguishing between different trace link types facilitates a consistent and ready-to-analyse set of traceability relations for a project and, consequently, richer traceability analysis [Mäder and Cleland-Huang, 2010]. Accordingly, the traceability information would support the context and constraints of the project with less effort which increases the benefits of traceability and compensates the additional cost of establishing and maintaining traceability [Bayer and Widen,

2002; Mäder et al., 2009a].

These studies propose a case-specific approach to define a TIM and, accordingly, define the fundamental elements of a TIM rather than a set of predefined link types. The main issue with this approach is with the mechanisms of how to define and use of the required traceability information model for each particular situation. On the other hand, [Limon and Garbajosa, 2005] recommends to have a standard way to specify a TIM in order to support generality and exchangeability in TIM implementation in tools.

In the following, existing specifications of traceability metamodels, in both the RE and MDE domain, are introduced.

[Ramesh and Jarke, 2001] observed traceability in several industrial projects focusing on the information needs of different stakeholders to ensure that traceability is maintained through all phases of systems development activities. Accordingly, they propose a very simple reference model for traceability (Figure 2.5) which introduces the basic terms for categorising traceability-related concepts and their relations. *Stakeholders* represent the people who are involved in the development process and who create, maintain and use the various development objects and the traceability links across them. *Objects* stand for the traceable entities, while *Sources* correspond to sources of information or knowledge such as documents, standards, meeting minutes, policies, etc. The three entities of the metamodel were derived based on previous studies on different aspects of requirements traceability. For example, version and configuration management systems focus on *Source* aspect, while studies at the management level usually focus on the *Stakeholder* aspect [Ramesh and Jarke, 2001].

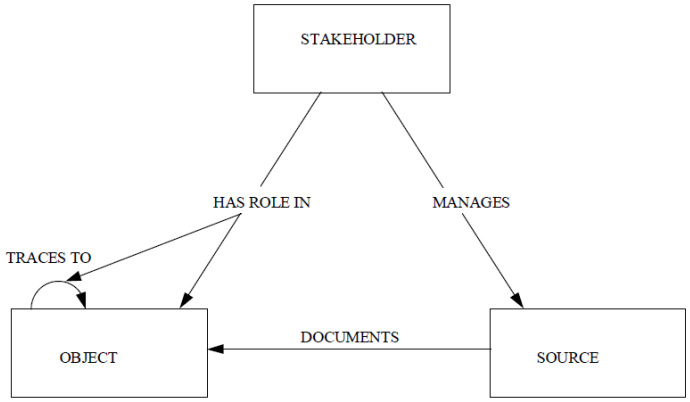


Figure 2.5: The traceability reference model of [Ramesh and Jarke, 2001]

[Ramesh and Jarke, 2001] state that the reference model has to be spe-

cialised and instantiated to create project-specific traceability models to capture the following dimensions of traceability information:

1. **What** information is recorded in the artefact? Is it a requirement, an assumption, an environmental constraint, etc. and on which other information is it based?

2. **Who** has created or updated the artefact and documented the information? Which other stakeholders have been involved in that process and who belongs to the group of potential users of the information?

3. **Where** has the information come from? What is the source of the information?

4. **How** is traceability information represented? How does this information relate to other traceability components?

5. **Why** has the artefact been created, modified, or evolved?

6. **When** has the artefact been created, modified, or evolved?

This conceptual model has been the source of many later studies in the field of requirements engineering to explore different aspects of traceability, such as stakeholder identification and relation [Alexander, 2005; Ozkaya, 2006] or rationale modelling [Dutoit et al., 2006; Potts and Bruns, 1988]. Large amount of studies have focused on TRACES-TO relationship or traceability links, which are explained in Section 2.1.1.3. However, the model does not provide a formal specification for traceability metamodels and there is not any guidance on how to implement and customise a traceability metamodel in tools for a given project using their reference model.

[Limon and Garbajosa, 2005] and [Espinoza et al., 2006], in a similar work, analysed the existing traceability metamodels and recognised that there is a lack of a standard way of specifying traceability between items and traceability relation types (traceability classification). Then, they proposed a Traceability Schema Specification (TS) as an approach to systematically specify the characteristics and needed information to provide traceability. This scheme facilitates traceability specification for a given project, to improve traceability management, and help to automate some part of traceability activities. According to [Limon and Garbajosa, 2005], a traceability scheme has to formally define the following items:

– **Traceability Link Dataset** that provides a wide basis to define all kind possible traceability links regardless the process or the objective of the link

– **Traceability Link Type Set (TYS)** which defines the information each traceability link contains for a given project

- **Minimal Set of Traceability Links (MINS)** which defines the set of links that should be created once a system baseline is closed (for each step).
- **Metrics set** to verify quality requirements of a traceability scheme such as correctness and level of accomplishment of the traceability strategy deployment.

In comparison to the above specification, which includes a dataset of all possible trace links, [Espinoza et al., 2006] proposes a TS which is completely designed to define traceability for a specific project. Accordingly, a traceability scheme includes the following items for a specific project:

- **Traceability Type Set** Types of links and their meaning
- **Traceability Role Set** Stakeholders and their permission to access traces
- **Minimal Links Set** Links have to exist for correctness and completeness
- **Metrics Set** Quality measures to verify quality requirements of a traceability scheme.

[Espinoza et al., 2006] also presents an example of how to implement the proposed TS in a document oriented CASE tool. The authors use the *caseml* language, a XML-based language defined to represent software engineering documents and schemas. All the software project information, including the traceability information, is kept in *caseml* documents in a repository.

[Mäder et al., 2009a] state that a traceability information model consists of two types of entity, traceable artefacts and traceability relations between these artefacts. Additionally, a TIM defines which types of artefacts have to be traced to which related artefacts by what type of relations. In order to be able to define and customise a TIM regarding a specific project, [Mäder et al., 2009a] suggests to use UML metamodels to represent the TIM as it is the standard and most practitioners are quite familiar with it. In the UML model, the traceable artefacts are defined as Class and traceability link types are represented as named associations between related Classes. The OCL is used to define additional constraints which can not be described within the model. They also provide a tool prototype, called *traceMaintainer*, which reads a TIM from the XMI format of the UML model, to provide integration between tools or prevent being dependent on specific UML tool.

There are other earlier UML-based approaches to define traceability. For example [Spence and Probasco, 2000] presents several traceability strategies as metamodels. However, the authors use only one type of trace and do not provide any mechanism to implement the metamodels. [Letelier, 2002]

proposes to use UML profile to define traceable entities and traceability relations, through extending the UML metamodel. Although this approach has several benefits, such as defining project-specific trace link types, it has some shortcomings which are inherently derived from the UML metamodel. For example, the UML trace relation does not support tracing all of types of UML entities (e.g. attribute and relations), while real relations can exist between all supported UML entity types.

The MDE research community considers traceability information models in a more structured manner. In MDE, the type and semantics of the traceability information is expressed in well-defined *traceability metamodels*. These metamodels are usually related to models specifying model transformation, though there are a few number of general traceability metamodels in the literature.

In the context of model transformations, [Jouault, 2005] notes that traceability information can be used in different scenarios, and that each requires a different format or complexity level of information. Accordingly, the author proposes a simple traceability metamodel. The metamodel, as shown in Figure 2.6, consists of two entities, the TraceLink entity and the AnyModelElement entity. The TraceLink entity has a ruleName attribute, which stores the transformation rule name and pointing to AnyModelElement via two multivalued references: sourceElements and targetElements.

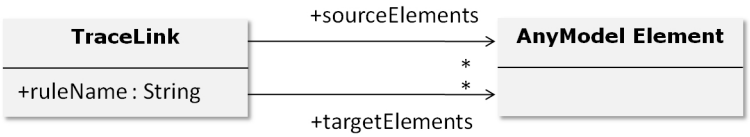


Figure 2.6: Traceability metamodel proposed by [Jouault, 2005]

The OMG, in a 'proposal for an MDA foundation model' [Object Management Group, 2005], introduces a model of a transformation record which is a set of traces produced by a model transformation. The metamodel of a transformation record is depicted in Figure 2.7. The core part of the metamodel –a *TransformationRecord* contains a number of *Traces* which are linked to the model elements (*Objects*)– can be found in several other researches with minor variations.

Similarly, [Amar et al., 2008] introduces a traceability metamodel to structure traces which are generated by the model transformation engine. The proposed metamodel is an extension of the one proposed in [Falleri et al., 2006]. The authors discuss that this metamodel is useful, for imperative as well as declarative transformations, to have a multiscaled trace. The fact that an operation transformation can call another one (or that the rules can trigger other rules) creates levels of nesting that it is useful to be able

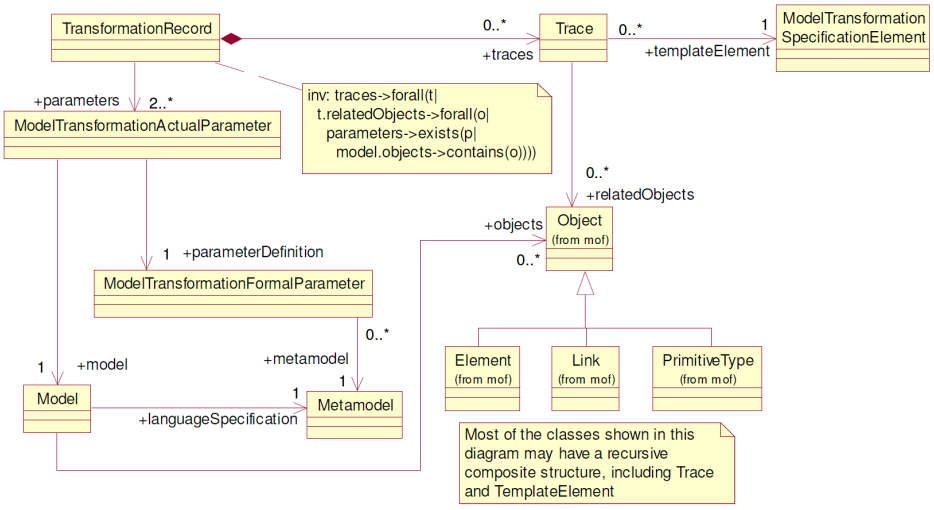


Figure 2.7: The metamodel of a transformation record introduced in [Object Management Group, 2005]

to represent. In doing so, they use the composite pattern on links, which allows them to separate low-level operation from high-level operation. The metamodel is illustrated in Figure 2.8.

[Walderhaug et al., 2006] proposes a generic solution for traceability in model-driven software development which introduces a metamodel and set of guidelines covering both the specification and usage of traceability. The proposed generic metamodel would be used by a traceability designer to build a model of which artefacts and relations to trace, and which information to include in the traces. So, instantiations of this metamodel can be used to model different traceability scenarios. This metamodel consists of four entities. The TraceModel entity is used to represent the container for the various trace links as well as for the various traceable artefacts. The TraceableArtefactType entity defines the mapping of a specific model artefact type to a corresponding traceable artefact. The ArtefactTraceType defines a specific trace type for a TraceableArtefactType. Finally, the RelationTraceType defines a specific trace type for a certain relation between a source and a target artefact type.

Another approach to define a general traceability metamodel is introduced in [Drivalos et al., 2009], which introduces a Traceability Metamodelling Language (TML) to define traceability metamodels regarding a specific traceability scenario. The TML identifies the core traceability concepts and patterns into four metaclass (Figure 2.9): *Trace* which acts as the root of a TML model, *TraceLink* which represents a traceability link between a number of elements, *TraceLinkEnd* that Represents an end of a trace-

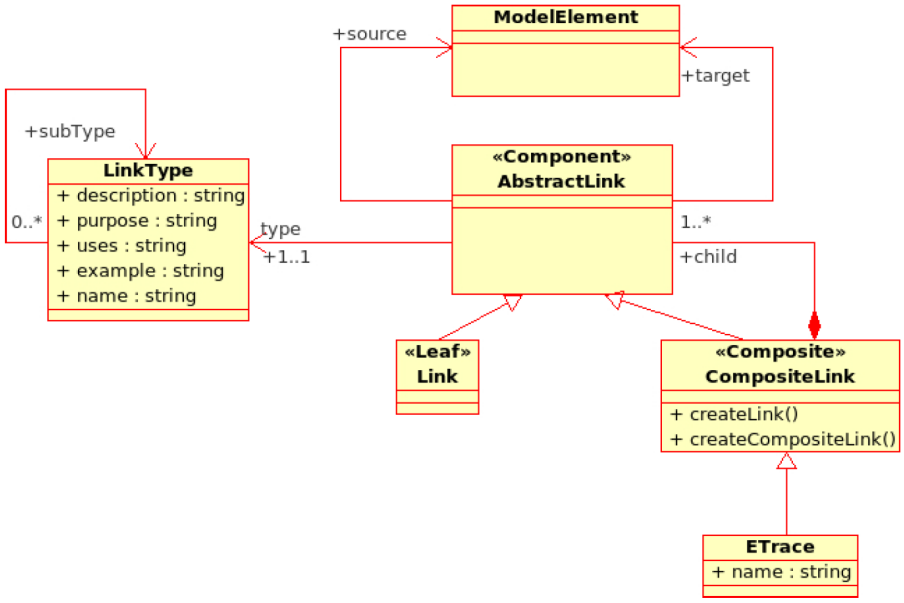


Figure 2.8: Traceability Metamodel proposed by [Amar et al., 2008]

ability link, and *Context* which allows traceability metamodel designers to attach custom information to traceability links. A TML model expresses the traceability metamodel and it is eventually transformed into a case-specific traceability metamodel. Additionally, the TML well supports the common features of the existing specifications of traceability metamodels in the MDE literature, identified by [Winkler and Pilgrim, 2010] and represented in a feature diagram (Figure 2.10).

Several other researchers from both the RE and MDE community have established their own traceability metamodels. These metamodels usually extend a basic traceability metamodel for a specific context of study. Some of them focus on specific context, such as traceability of NFRs ([Kassab et al., 2009]), traceability in software product lines ([Anquetil et al., 2010]), use of traceability specifically in model-to-text transformations ([Olsen and Oldevik, 2007]), and traceability in context of mega models ([Barbero et al., 2007; Seibel et al., 2010]). Some of them still consider general traceability metamodels (e.g. [Grammel and Vigot, 2009; Vanhooff et al., 2007]).

2.1.3.3 Traceability Creation

Traceability creation is the activity of acquiring traces, representing and storing them in some way. The TIM, tools, and the data structures prepared earlier drive the traceability creation sub-activities. In the literature,

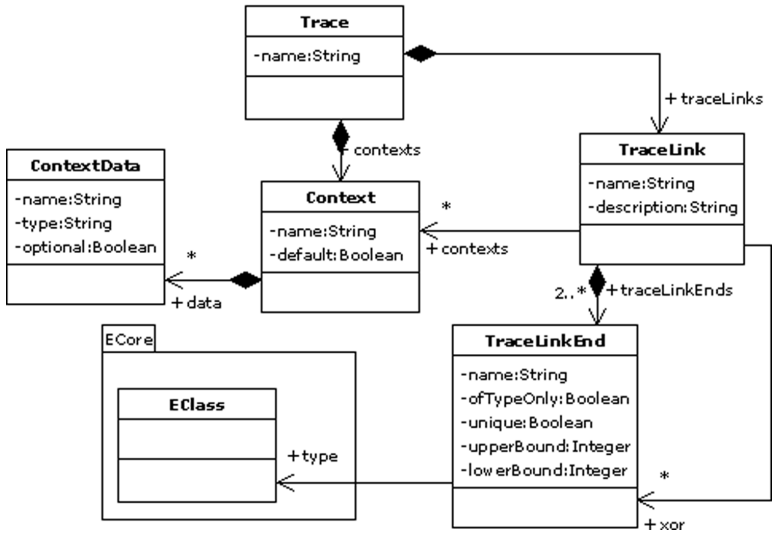


Figure 2.9: The Traceability Metamodelling Language proposed by [Drivalos et al., 2009]

various terms are used to refer to this activity including *trace recording* [Winkler and Pilgrim, 2010], *trace generation* [Spanoudakis and Zisman, 2004], *trace creation* [Aizenbud-Reshef et al., 2006], and *trace production* [Pinheiro, 2003], although they might consider creating traces from different perspectives and consequently focus on different aspects or tasks.

[Gotel et al., 2012b] identifies two approaches in trace creation, *trace capture* and *trace recovery*. In the first approach, the traces are created concurrently with the forward engineering process, while, through trace recovery, traces are discovered at some point later. [Winkler and Pilgrim, 2010] also categorises trace creation techniques in a similar way and indicates that trace recording can be performed either *on-line* or *off-line* which are almost same as *trace capture* and *trace recovery* respectively. In the former one, traces are created automatically as a by-product of the development process. In the latter case, traces are created after the actual activity has been finished. Similarly, [Asuncion et al., 2010] state that traceability creation can be done *prospectively* or *retrospectively*.

Manual trace creation is an effort-intensive, time consuming, and error-prone task [Spanoudakis and Zisman, 2004]. Without automated support, creating traces is typically infeasible in practice, especially for large projects [Asuncion and Taylor, 2012]. Therefore, numerous researchers have been working to automate the process of traceability creation. However, there are yet some cases in which traces have to be created manually, for example due to a regulatory reason [Spanoudakis and Zisman, 2004] or, in the MDE domain, to explicitly record traces because of the tasks carried

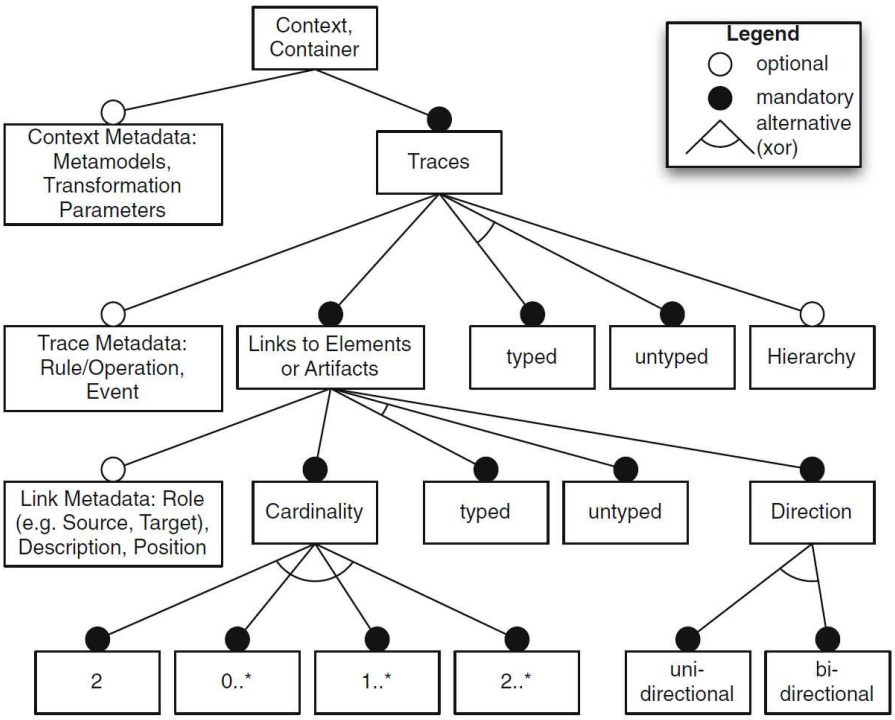


Figure 2.10: Feature diagram of traceability models extracted by [Winkler and Pilgrim, 2010]

out beyond the scope of modelling [Winkler and Pilgrim, 2010].

Trace recovery has been mostly investigated in the context of requirements traceability, in which traces are identified from existing artefacts. In this context, automated trace recovery techniques identify candidate traceability links retrospectively from existing artefacts by using information retrieval or machine learning techniques to analyse text-based artefacts (i.e. [Lucia et al., 2012; Grechanik et al., 2007; Kagdi et al., 2007; Marcus and Maletic, 2003]), or by applying rule-based approaches to recover traces from both models and text-based artefacts (i.e. [Spanoudakis et al., 2004; Mäder et al., 2008b]).

In contrast, trace capture techniques have been introduced in the MDE domain, in which traces can be produced on-line as a by-product of transformation activities. This is specified in the OMG MOF QVT standard for models [Object Management Group, 2011c] and demonstrated in model-to-model and model-to-text transformations (i.e. [Jouault, 2005; Falleri et al., 2006; Gorp and Janssens, 2005; Oldevik and Neple, 2006; Olsen and Oldevik, 2007]). However, there are some studies proposing automated techniques for capturing trace links in software traceability (i.e. [Anderson et al., 2000;

Asuncion and Taylor, 2012]). In the following, a brief discussion of the main approaches to trace recovery and trace capture is provided.

Using Information Retrieval

Information Retrieval (IR)-based methods recover traceability links on the basis of the similarity between the text contained in the software artefacts. The underlying assumption, in these methods, is that if the textual content of two artefacts refers to similar concepts, then the two artefacts are conceptually related and a traceability link between them could be established. The higher the textual similarity between two artefacts, the higher the likelihood that a link exists between them. A distinct advantage of using IR techniques is that they do not rely on a predefined vocabulary or grammar. This allows the method to be applied without large amounts of preprocessing or manipulation of the input, which drastically reduces the costs of link recovery.

An IR-based trace links recovery process has three key steps depicted in Figure 2.11, which is organized in a pipeline architecture.

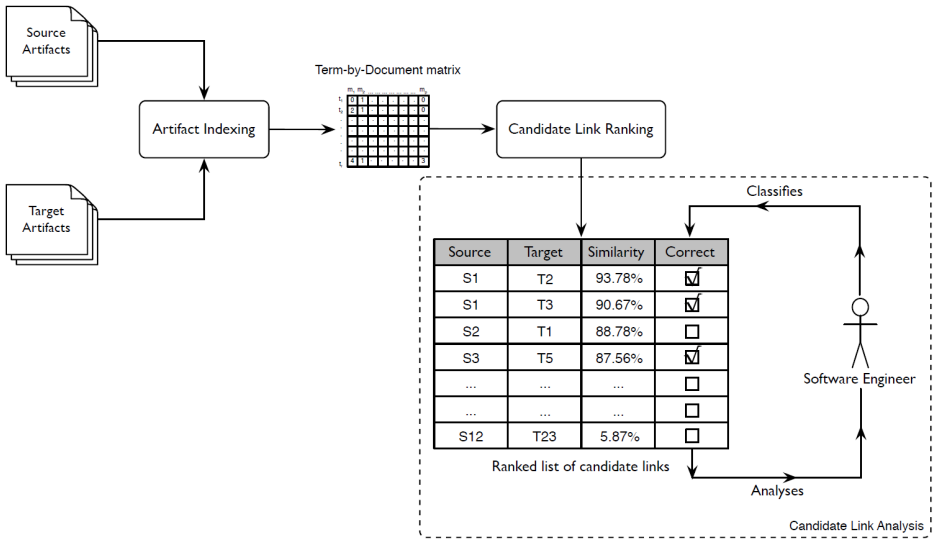


Figure 2.11: IR-based traceability recovery process [Diaz et al., 2013]

1. Extraction, pre-processing, and indexing corpus
   The process starts by extracting information from textual software artefacts, using given granularities. For example, the majority of techniques applied on source code parse and represent the artefacts at a class level granularity (e.g. [Antoniol et al., 2002; Lucia et al., 2007]).

Then, the extracted information is pre-processed and represented as a set of document in the resulting corpus. Finally, an IR method is used to index the corpus and represent them in a homogeneous document space by extracting information about the occurrence of terms within documents. Trace recovery techniques apply different pre-processing strategies on text. However, *text normalisation*, *identifier splitting*, and *stop word removal* are the frequently used steps in pre-processing [Lucia et al., 2012]. Morphological analysis, such as stemming which is the process of reducing inflected words to their stem, base, or root form [Porter, 1980], could also be used. An alternative approach is searching for n-grams rather than stems (e.g. [Zou et al., 2010] uses 2-grams to compare the contents of software artefacts).

The terms extracted from the documents are represented by a $m \times n$ matrix, called *term-by-document* matrix [Baeza-Yates and Ribeiro-Neto, 1999], where m is the number of all terms that occur within the artefacts, and n is the number of artefacts in the repository. A generic entry $w_{i,j}$ of this matrix denotes a measure of the weight (i.e., relevance) of the i^{th} term in the j^{th} document. In the field of IR, various methods have been developed for weighting terms. However, they all basically use these three factors in their formulation, *term frequency (tf)*, *document frequency (df)*, and *decument length (dl)*. For example, [Baeza-Yates and Ribeiro-Neto, 1999] introduces the *tf-idf* indexing mechanism, in which *idf* is the inverse document frequency. This mechanism gives more importance to words having a high frequency in a document (high *tf*) and appearing in a small number of documents, thus having a high discriminant power (high *idf*). The *term-by-document* matrix is the output of the indexing process.

2. Generating ranked list
   In this step, the IR-based recovery method compares a set of source artefacts (represented as documents) against another set of target artefacts and uses the defined similarity measure to rank all the possible pairs by similarities (candidate traceability links). Probabilistic models [Antoniol et al., 2002], Vector Space Model (VSM) [Baeza-Yates and Ribeiro-Neto, 1999], and Latent Semantic Indexing [Deerwester et al., 1990] are the most frequently used in IR-based traceability recovery methods [Lucia et al., 2012]. In the probabilistic model, a source document is ranked based on the probability of being relevant to a particular document. In VSM and LSI, artefacts are represented as vectors of terms. Thus, the source artefacts are ranked against target artefacts by computing the distance between the correspondence vectors.

3. Analysis of candidate links
   Once the list of candidate links has been generated, it is provided to the software engineer for examination. The software engineer reviews the candidate links, determines those that are correct links (*actual links*), and discards the *false positives*. The evaluation process used in this step is based on human intervention and therefore has all the advantages and disadvantages associated with such activities.

   There are two approaches in analysing the ranked list. The first approach is based on the analysis of the full ranked list (i.e. [Hayes et al., 2003; Cleland-Huang et al., 2005b; Lucia et al., 2006b]). The other approach is to cut the ranked list (based on a threshold) (i.e. [Antoniol et al., 2002; Marcus and Maletic, 2003; Cleland-Huang et al., 2007; Zou et al., 2010]). This approach is based on the fact that the density of the correct links are higher at the top of the ranked list [Lucia et al., 2009]. These methods are classified into *cut-point* and *threshold* based strategies. In the former one, the list is cut based on the number of the recovered links; the top μ links are selected and presented to the user. In the former one, the links with the similarity value greater than or equal to the given threshold ε are retrieved.

The retrieval accuracy of an IR-based trace recovery method is generally evaluated by calculating the ratio of a set of retrieved links over a set of relevant links. The set of relevant links is generally provided by developers at the end of the process. IR methods inherently fail to retrieve all of the relevant links and thus IR-based trace recovery methods require interaction between developers and the recovery tool [Lucia et al., 2012].

The retrieval performance of IR methods is measured using two metrics, namely *recall* and *precision* [Baeza-Yates and Ribeiro-Neto, 1999]. Recall is the ratio of the number of correct retrieved links over the number of relevant links. Evaluating recall alone is not enough, as it is easily possible to increase recall by retrieving more trace links (even all links), which results in more irrelevant links. Accordingly, it is required to control and decrease the number of non-relevant links. Precision is the fraction of the links retrieved that are relevant.

$$Recall = \frac{|\{RelevantLinks\} \cap \{RetrievedLinks\}|}{|\{RelevantLinks\}|}$$

$$Precision = \frac{|\{RelevantLinks\} \cap \{RetrievedLinks\}|}{|\{RetrievedLinks\}|}$$

IR-based trace recovery methods differ according to the IR strategy which they use and also the parameters which affect their performance [Lucia et al., 2012].

[Antoniol et al., 2002] proposes a method based on IR to recover traceability links between source code and free text documents. In their approach,

the identifiers are extracted from a source code component used as a query to retrieve the documents relevant to the component. The authors applied both a probabilistic and a vector space IR model for ranking the documents against a query. The result of their experiment show that both models achieve similar results in terms of accuracy and performance.

In the same context, [Marcus and Maletic, 2003] uses LSI to recover traceability links between source code and documentation illustrated in several experiments. They compared the result of their experiments with similar works (e.g. [Antoniol et al., 2002]) using different IR methods (PM and VSM) and discuss that LSI performs at least as well as PM and VSM IR methods combined with full parsing of the source code and morphological analysis of the documentation.

[Diaz et al., 2013] proposes a traceability recovery method, named TYRION (Tractability link Recovery using Information retrieval and code OwNership), for recovering traces between requirements and code. They conjecture that the ownership of the artefacts provides information about the division of requirements (or use cases) to be implemented. They discuss that TYRION provides more accurate list of candidate links than a standard IR-based traceability recovery technique and, thus, code ownership information represents a useful source of information which can be used to complement textual information.

[Hayes et al., 2003] proposes an IR-based approach for improving requirements tracing, focusing on candidate link generation. They have applied different VSMs and evaluate each method with similar approaches and tools. First, they discovered that a classical vector space model does not outperform analysts or existing tools in terms of recall or precision. Then, they developed two extensions to this algorithm. The first uses a simple key-phrase list, one that can be easily pulled from the definitions or acronym section of a requirement document. The retrieval with key-phrases algorithm resulted in improved recall but with decreased precision. Next, they added a simple thesaurus and achieved better recall (85%) and precision (40%). [Hayes et al., 2004] extended their previous study and consider overall requirements tracing. Accordingly, they built a tool, RETRO (REquirements TRacing On-target), which also uses the LSI method and implements analyst feedback into the tracing process [Hayes et al., 2004]. Finally, in [Hayes et al., 2006], they examined the effectiveness of IR methods in automating the tracing of textual requirements through introducing a set of goals for an effective tracing tool. Particularly, they found that analyst feedback improves the final trace results using objective measures.

[Lucia et al., 2004] present a traceability recovery method and tool based on LSI in the context of an artefact management system. The tool provides the software engineer with the set of links not traced by the software engineer and retrieved by the tool and the set of links traced by the software engineer and not retrieved by the tool. [Lucia et al., 2006a] suggest an incremental

traceability recovery approach to gradually identify *optimal* threshold to achieves an acceptable balance between traced links and false positives. Additionally, [Lucia et al., 2006b, 2008] show that using feedbacks within an incremental traceability recovery process improve the retrieval performances of these techniques.

[Cleland-Huang et al., 2005b] addresses the recall and precision problems in dynamic requirements traceability and introduces three strategies for incorporating supporting information into a probabilistic retrieval algorithm in order to improve the performance. The strategies include hierarchical modelling, logical clustering of artefacts, and semi-automated pruning of the probabilistic network. Similarly, [Zou et al., 2006] describes a method for improving the precision of trace results through incorporating the use of phrases detected and constructed from requirements using a part-of-speech tagger. They also suggest to use a project glossary to find additional phrases and weight the contributions of key phrases and terms. They indicate that phrasing primarily impacted the precision of top links in contrast to other precision enhancement techniques such as the use of hierarchical information and pruning methods (e.g. [Cleland-Huang et al., 2005b]) which had a more general impact on overall precision. The outcome of this research is implemented in a tool called Poirot [Lin et al., 2006]. Poirot uses a probabilistic network model to generate trace links dynamically at runtime form variety of requirement management tools.

[Abadi et al., 2008] compares the effectiveness of several different IR techniques to discover traceability links from code to documentation. They compare dimensionality reduction methods (e.g., LSI), probabilistic and information theoretic approaches (e.g. JS), and the standard VSM. The results show that the techniques that provide the best results are VSM and JS. Similarly, the empirical study in [Oliveto et al., 2010] observes that IR methods are almost equivalent. [Gethers et al., 2011] proposes a combination of different techniques and demonstrates that it outperforms stand-alone IR methods.

A common challenge in IR-based techniques is filtering out noise from the list of candidate links, in order to improve the recovery accuracy. Several researches has been carried out in enhancements to improve the accuracy of the IR-based trace recovery methods and make the process less time consuming and error prone. Some improvements focus on the terms that are extracted from the artefacts and pre-processing techniques. [Capobianco et al., 2013] has suggested that domain-specific terms (e.g., jargon) best describe the concepts in the code. They propose to use only the nouns from the software artefacts. Other work has also adapted the weights of the artefacts terms depending on the length of the artefacts [Settimi et al., 2004], a project glossary [Zou et al., 2006], external dictionaries [Hayes et al., 2003; Zou et al., 2010]. Additionally, some approaches focus on transformations at the vocabulary level (e.g., [Zou et al., 2010]), while others leverage rela-

tionships between source code-based artefacts, based on different types of information (e.g., structural dependencies between source code [McMillan et al., 2009] or code ownership [Diaz et al., 2013]). Recently, smoothing filters have been shown to improve the precision of IR-based traceability [Lucia et al., 2011].

Moreover, IR-based approaches are primarily suitable for linking requirements and artefacts that exhibit a high lexical correlation, which in some cases this type of high lexical correlation between the requirement and its implemented artefacts often does not exist [Cleland-Huang et al., 2003]. Finally, the links derived by IR methods are fundamentally generic and have no type. This is because these methods have been founded based on the similarity between texts. Therefore, a link only shows that there is a relationship between two texts without any interpretation of the semantics behind the it. Accordingly, IR-based trace recovery methods can not be easily applied to identify semantically rich traces.

Rule-Based Techniques

The main motivation for rule-based traceability approaches is to support automatic traceability creation in various types of documents generated during different phases of the software development life cycle. In general, rules assist and automate decision making, allow for standard ways of representing knowledge that can be used to infer data, facilitate the construction of traceability creators for large data sets, and support representation of dependencies between elements in the documents. In addition, the use of rules allows for the creation of new relationships based on the existence of other relationships, supports the heterogeneity of artefacts being compared, and supports data inference in similar applications.

[Spanoudakis et al., 2004] propose a rule-based approach to generate traceability between requirements statement and used cases, and analysis object models represented in UML. The system incorporates a traceability rule engine which interprets rules and generates trace relations. The artefacts and the traceability rules are represented in the XML. Two different types of traceability rules have been identified and implemented in this approach: *requirement-to-object-model* rules are used to trace the requirements and use cases to an analysis object model, and *inter-requirements traceability* rules are used to trace requirements and use cases to each other. The approach has been evaluated in several experiments which suggest acceptable results. The author indicates that their approach can be customised by adding new traceability relations and new rules. However, experiments also show that the main shortcoming of this approach is the creation of new traceability rules and, particularly, the identification of syntactic patterns with appropriate level of granularity. Addressing this shortcoming, [Spanoudakis et al.,

2003] developed a machine learning algorithm that produces new traceability rules. In particular, new rules capture those relations which are failed to be recovered by existing rules. New rules are created based on a generalisation of existing traceability rules.

The above approach has been customised and used later in other contexts including product-line [Jirapanthong and Zisman, 2005; Jirapanthong, 2007] and multi-agent systems [Cysneiros and do Prado Leite, 2004; Cysneiros, 2007; de Pádua Albuquerque Oliveira et al., 2007]. Accordingly, [Zisman, 2012] demonstrates the rule-based traceability creation framework which has been used in all of these studies in general. They have presented a traceability information model for the documents of their concerned with different types of trace relationships. The framework assumes documents represented in XML format and uses traceability rules specified in XQuery with some extended functions that they have created. Traceability rules are created based on different aspects, including the semantic of artefacts being traced, the types of required trace relationships, the grammatical roles of words in textual artefacts, and synonyms and other associations of the words in the textual artefacts. The framework has been evaluated in terms of recall and precision for different case studies. The results of the evaluation are comparable to other approaches to support automatic creation of trace relationships.

[Cleland-Huang et al., 2002b] and [Cleland-Huang et al., 2003] use user input to recover trace links between requirements and performance models and between non-functional requirements and design code artefacts. Fine-grained trace links are dynamically generated during system maintenance and refinement based on user-defined links, which are specified during the inception, the elaboration and the construction of the system. This approach supports trace recovery based on invariant rules of design patterns which are used to identify critical components of classes.

There are researches who apply pattern matching concepts in the context of trace recovery. [Pinheiro and Goguen, 1996] suggests the use of a regular-expression language to define patterns to be matched by objects which are related in a certain way. This approach has been provided in the context of a tool called TOOR (Traceability of Object Oriented Requirements) which support traceability between requirements, design documents, specifications, code and other artefacts through user-defined relations. [Ying et al., 2004; Zimmermann et al., 2005] have developed an approach that applies data mining techniques to determine change patterns among sets of files that were changed together frequently in the past. The underlying assumption of their approaches is that the change patterns can be used to recommend potentially relevant source code to a developer performing a modification task. However, no guidance is provided for the type of the relationship of two related entities.

Additionally, [Grechanik et al., 2007] simulated the manual human-driven

procedure of searching for common patterns and similarities between the names and values of program entities and the names of elements of use case diagrams (UCDs), in identifying trace links between use-cases and Java source code. They suggest using machine learning (ML) techniques that classify program entities as belonging to elements of UCDs based on the names of program entities, their runtime values, and the names of elements of UCDs. ML techniques can support partial matches between names and values, for example, when patterns in these names and values are not defined precisely. Their solution is called LEarning and ANAlyzing Requirements Traceability (LeanArt).

Finally, [Asuncion and Taylor, 2012] present a set of automated techniques to capture custom trace links across heterogeneous artefacts. The techniques are presented in the context of their Architecture-Centric Traceability for Stakeholders (ACTS) framework [Asuncion et al., 2010], which is implemented on top of ArchStudio [Dashofy et al., 2007]. They use open hypermedia concepts [Anderson et al., 2000] to capture trace links across tool boundaries and rendering artefacts at different levels of granularity. Third-party tools are connected to ACTS by building open hypermedia adapters for the tools, which are called viewers in hypermedia systems [Anderson et al., 2000]. Customisable rules are used to enable users to define the artefacts to trace and to specify traceability relationships. Rules, represented as XSL Transformation (XSLT) [World Wide Web Consortium (W3C), 1999], are used to analyse the recorded events to create traceability links. In this approach, all the control flow goes from ACTS to adapters while users continue to work with their usual development tools and follow their development process. However, the development of adapters and rule are limited to tools and heuristic that users choose to employ.

This work is similar to the pre-requirements traceability approach presented in [Pohl, 1996a] in which traceability relations are created as a result of creating, deleting or manipulating requirements. The approach is implemented in a tool called PRO-ART. PRO-ART provides (a) a three-dimensional framework for requirements engineering which defines the kind of information to be recorded, (b) a trace-repository for structuring the trace information and enabling selective trace retrieval, and (c) a novel tool interoperability approach which enables (almost) automated trace capture.

Similar to the IR-based approaches, most of the rule-based trace recovery perform properly in situations where terminology is used consistently in the different traceable artefacts, which in some cases are not possible. Additionally, they require setup overhead to specify a set of trace recovering rules.

Transformation and Translation Techniques

Trace information can be automatically created as by-products of transformation activities, that is, activities which automatically transform one artefact to another. This can be done either implicitly or explicitly [Olsen and Oldevik, 2007]. In the former one, a transformation engine generates the trace links automatically when a transformation is executed. In the later one, additional code must be inserted into the transformation code in order to generate a trace model, which can be done by writing the trace code each time or running a higher order transformation (HOT) [Tisi et al., 2009] on the transformation model. Accordingly, model transformation tools support the automated generation of trace links in different ways.

The ATLAS Transformation Language (ATL) [Jouault, 2005], uses HOT to extend a transformation program to support automatic traceability generation. [Falleri et al., 2006] propose a traceability framework in the Kermeta language which requires to add trace generation code in the transformation code. Generating trace links during model transformation is also demonstrated for FUJABA in [Gorp and Janssens, 2005] and in a general notation in [Vanhooff et al., 2007].

In the OMG MOF Query/View/Transformation (QVT) specification [Object Management Group, 2011c], traces between model elements involved in a transformation are created implicitly. The specification describes three model transformation languages that can be used: Relations, Core, and Operational Mappings. In the Relations and Operational Mappings languages, trace-links are created automatically without user intervention. In the Core language, a trace class must be specified explicitly for each transformation mapping. The implementation does not store trace models as external files which could be a challenge for interchangeability between tools [Kurtev et al., 2007].

Similar approaches are applied in the context of model-to-text transformation. In the MOF Models-to-Text transformation language [Object Management Group, 2008], traceability is defined to be explicitly created by the use of a trace block inserted into the code. This approach provides user-defined blocks that represent a trace to the code generated by the block; this is specifically useful for adding traces to parts of the code that are not easily automated. A drawback of the approach is a cluttering of the transformation code. A complementary approach, as taken in MOFScript, is to automate the generation of traces based on model element references [Oldevik et al., 2005; Oldevik and Neple, 2006]. This approach is also taken in the Epsilon Generation Language [Rose et al., 2008].

Obeo Traceability [Information Technology ISO/IEC, 2007] is a traceability tool developed by Obeo that handles traceability links between model elements and code and vice versa. This tool enables round trip support; updates in the model or the code are reflected in the connected artefacts. Analyses are also available using the traces as input, but since this is a commercial tool, restricted information describing the solution is available. It

seems to be based on similar ideas that are used in MOFScript where model elements are traced to exact positions in files.

The Epsilon framework [Kolovos et al., 2008] provides implicit traceability support through an external trace model that can be accessed in Epsilon's workflow mechanism (which is based on ANT). An Epsilon program, such as a transformation or a model merging operation, can expose trace information (in the form of a container of trace links) and this information can be accessed by other model management tasks (such as validations) or even non-MDE tasks, such as visualisations generated with GraphViz [Ellson et al., 2003].

In the context of requirements traceability, [Anderson et al., 2002] addresses the challenge of managing the relationships that exist between software artefacts. They consider this problem from information integration perspective as establishing a relationship between documents typically implies that an engineer must integrate information from each of the documents to perform a development task. Doing so, heterogeneous artefacts are translated into a homogeneous format and then relationships are discovered automatically between text-based software artefacts through the use of keywords. Similar to IR-based and rule-based trace recovery approaches, this technique is also limited to text-based artefacts.

However, the area of trace creation is still challenging. Using IR-based and rule-based are limited to text-based artefacts with high lexical correlation or use consistent terminology. Trace creation as a by-product of transformation also struggles to find a best (an effective) way to generate traces. In implicit trace recovery, trace links do not have any case-specific semantics. They are generic and usually of the form <source element, transformation rule, target element>, while richer traceability links are required in a usable solution (discussed before in Section 2.1.1.3). On the other hand, although in the case of explicit trace recovery, trace links with rich-semantics can be generated, the traceability engineer has to insert traceability code into the transformation specification. Consequently, it is tightly integrated and dependent on the existing environment and technologies.

Miscellaneous/Hybrid Techniques

[Egyed and Grünbacher, 2005] use program execution traces to recover relations between source code, requirements and test-cases. In this approach, test-cases are manually related to requirements. By using these manually defined links together with the dynamic program behaviour logs from the execution of the test-cases, their tool is able to recover trace links between requirements and code. These trace links are derived using transitive reasoning and shared use of common ground. An example of transitive reasoning is when A depends on B and B depends on C then A

depends on C. On the other hand, shared use of common ground is the use of the criterion that if A and B depend on subsets of a common ground (code in this case) and these subsets overlap, then A depends on B.

[Zhang et al., 2006] present an ontology-based approach for traceability recovery, to support reverse engineering. They suggest creating formal ontological representations for both the documentation and source code artefacts. These representations are then aligned to establish traceability links at the given semantic level. In this approach, traceability links are recovered by utilizing the structural and semantic information in various software artefacts and the linked ontologies are also supported by ontology reasoners to infer implicit relations among these software artefacts.

Analysing existing trace relationships to obtain implied relationships may also provide a source of automatically recovered trace links. TraceM is a framework for automating the management of traceability relationships which uses this technique. A key contribution of TraceM is its ability to transform implicit relationships into explicit relationships by processing chains of traceability relationships [Sherba et al., 2003]. This work builds on techniques from open hypermedia [Osterbye and Wiil, 1996]) and information integration [Anderson et al., 2000]. TraceM allows stakeholders to view the explicit relationships in a system and to chain these relationships together to make previously implicit relationships explicit. Moreover, since these implicit relationships are being made explicit within the context of a comprehensive requirements traceability framework, they can be explicitly managed, tracked, and analysed as the software project evolves.

There are also other methods of automatically creating traces in situ, as artefacts are being created and modified during the development process, for example by directly analysing the actions. [Wenzel et al., 2007] use a version history of a model to trace single model elements or groups of elements. Given a history of successive model revisions, tracing takes place by locating that element in another model. [Mäder et al., 2008a] presents an approach in the context of UML-based software development. The approach recognises the development activities applied to models and capture trace links while they are performing. Although, it is initially introduced for traceability maintenance, it is used to create trace links as well. A prototype tool called traceMaintainer has been implemented which supports this approach. [Asuncion and Taylor, 2012] capture traceability links in situ, while artefacts are generated or modified. A main shortcoming of these approaches is that they are restricted to capture traceability links from behavioural information only.

[Costa and da Silva, 2007] proposes a reactive approach to traceability using MDE concepts. The authors identify two viewpoints in creating traces, *dependency viewpoint* and *generative viewpoint*, and discuss that a combination of both approach should be used to identify and generate trace links. The dependency viewpoint identifies traces describing some implicit or ex-

plicit semantic relation or dependency between them. While, generative viewpoint is interested in the relationships between the generated artefact and the one which was used for the generation. In this case, the traces are created as a by-product of the the transformation and in most cases can be done automatically. Figure 2.12 shows a conceptual overview of their approach.

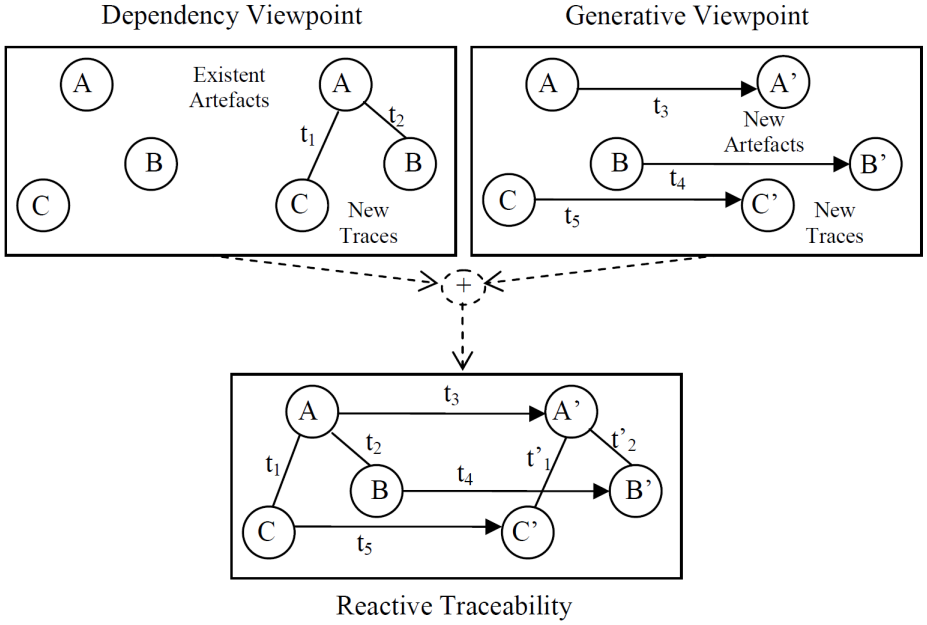


Figure 2.12: Reactive Traceability [Costa and da Silva, 2007]

2.1.3.4 Traceability Maintenance

Traceability maintenance refers to those activities associated with updating pre-existing traces as changes are made to the traced artefacts and the traceability evolves, and creating new traces where needed to keep the traceability relevant and up to date [Gotel et al., 2012b].

Traceability information is subject to gradual degradation as the related artefacts are modified. As a result, the recorded relationships may end up being incorrect or inaccurate and as a result cannot support traceability-enabled activities. One of the most challenging aspects of traceability is how to maintain the integrity of the relationships, i.e. trace links, while the referenced entities continue to change and evolve. Traceability maintenance can also be required following changes to the requirements and constraints that drive the overarching traceability strategies. Additionally, it can be seen as an *enhancement* activity, proposed in [Egyed et al., 2007], to identify missing traces and resolve errors following trace creation process.

[Gotel et al., 2012b] identify two main approaches for traceability maintenance: *continuous* and *on-demand*. In the former approach, the changes are constantly monitored and the update of impacted trace links immediately follow changes to traced artefacts. In the latter approach, a dedicated and overall update of the trace set (in whole or in part), is done generally in response to some explicit trigger or preparation for an upcoming traceability use. In a similar way, [Seibel et al., 2012] categorise traceability maintenance strategies into *reactive* and *proactive* strategies. In the context of MDE, [Drivalos-Matragkas et al., 2010] define *event-driven* and *state-based* traceability maintenance, which can be considered as two techniques, respectively, for *continuous* and *on-demand* maintenance approaches (mentioned above). In the case of *event-driven*, change events trigger the maintenance process, while in *state-based* case, the detection of model changes takes place by comparing different versions of the models, and potential links are found based on the identified changes. In the following, the main contributions in traceability maintenance are presented.

Reactive Traceability Maintenance

In this method, the detection of model changes takes place by comparing an instance of the model under consideration in time t_1 with an instance of the model in time t_0, where t_0 is the time when the model was checked for the last time. This comparison can take many different forms. Usually, all the artefacts under consideration are expressed in a common representation, such as XMI, and then their *diff* is calculated. In some cases, it is required for example when no information about changes is available.

[Maletic et al., 2005] describe an XML-based approach to support the evolution of trace links between models expressed in the XML. The authors also describe a traceability graph and its representation in the XML, independent of specific models or tools. They propose to evolve traceability along with the models by detecting syntactic changes at the same level and type as the trace links (e.g., textual links require textual change detection).

An example of a state-based approach is the one proposed by [Sharif and Maletic, 2007]. In this approach a difference tool such as EMFCompare is used to identify syntactic differences between different versions of a model. Based on these differences and user input the links are evolved.

[Drivalos-Matragkas et al., 2010] propose a state-based traceability maintenance in the context of the TML [Drivalos et al., 2009]. A traceability maintenance script, associated with reconciliation expression, is used to detect and reconcile dangling links. The script traverses all the links in the TML model and when a discrepancy is detected the link is flagged as dangling. Then the maintenance script attempts to reconcile it. If this is not

possible, then the user is notified of the suspect link.

There are also many researchers especially in the context of consistency management between models after their evolution. However, the necessity to maintain traceability, along with changing a related model, has been little emphasised. [Engels et al., 2002] present a classification of UML model refinements to preserve consistency during the evolution of UML-RT models (a UML enhancement for real-time systems), and identify three kinds of atomic modification: creation, deletion and update. [Mens et al., 2005] describe an extension to the UML meta-model to support the versioning and evolution of UML models. They classify possible inconsistencies of UML design models and provide rules, expressed in the Object Constraint Language (OCL), to detect and resolve them.

Proactive Traceability Maintenance

[Cleland-Huang et al., 2002a, 2003] propose event-based traceability (EBT) as a traceability environment that is more supportive of change. They have identified a set of standard change events and a method for monitoring user's actions within a requirements management environment. EBT uses a publisher-subscriber paradigm to observe changes in entities (mainly requirements) and react accordingly. The relationships between requirements and dependent entities are re-established through an event service. For example, a scenario that is dependent upon a requirement subscribes to that requirement in order to receive change notifications. When a requirement changes, an event message is published to the event server and, subsequently all dependent entities are notified. Since the focus of the approach is on recognising types of requirements changes, it does not deal with the more complex task of recognising multi-step change activities to models comprising different element types. However, it is stated that it is possible to identify and define such changes as standard event messages [Cleland-Huang et al., 2003].

A similar approach is used by [Mäder et al., 2008b,a] in the context of UML-based software development. They focus on structural UML models and maintaining post-requirements traceability. Accordingly, they identify three types of change: add, delete, modify. Rules are used to recognise change events and specify the actions which should be done consequently. They provide a tool, traceMaintainer, which is an extension to UML tools and support their approach.

[Murta et al., 2006] describe an approach called ArchTrace to keep the architecture and implementation synchronized as they evolve separately. ArchTrace is a policy-based approach in which policies are either rules, deciding upon actions to take, or constraints, limiting the kinds of actions that can be taken. They provide an initial set of eight policies based on informally

observing developers in action for example when committing a new version of an artefact. In their implementation, xADL 2.0 [Dashofy et al., 2001] is used to describe software architectures and Subversion [Pilato, 2004] to store source code.

[Seibel et al., 2012] present a retrospective traceability maintenance approach in the MDE domain, which follows their previous work [Seibel et al., 2010]. The traceability maintenance approach relies on formal rules to automatically maintain traceability links, which represent dependencies primarily between formal software artefacts. The formal rules (Story Diagrams) specify a precise semantic for specific types of traceability links and classified into *creation* and *deletion* rules. This approach has two traceability maintenance strategies: *batch* and *incremental*. The batch strategy is a reactive traceability maintenance strategy and triggered by a developer on demand based on the current state of the artefacts. The incremental strategy is triggered automatically when changes occur. Currently, the approach is limited to the Eclipse workspace and changes coming from EMF or GMF editors.

2.1.3.5 Traceability Use

Traceability use comprises those activities related to exploiting traces in order to support traceability-enabled activities and tasks. In order to use a trace, it needs to be retrieved and rendered visible in some specific way. Additionally, [Gotel et al., 2012b] indicate that an important part of the use process is assessing the quality of the retrieved trace links in terms of the fitness for purpose with respect to the given task or activity. Accordingly, they define four main components in a use process: retrieving traces, rendering traces, assessing traces, and recording trace use. The last two component provide a feedback loop to improve the overarching traceability strategy. As discussed in Section 2.1.3.1, there are limited studies in regarding assessing the quality of traceability and, even though, they are not explicitly focused on this aspect.

Nevertheless, the challenges of searching and browsing traces in an efficient and user-friendly way has not been fully investigated [Winkler and Pilgrim, 2010], though they are essential for usability of a traceability solution [Winkler, 2008]. One reason for this is that most researchers often consider using traces as trivial which is not [Marcus et al., 2005]. In the following, the existing research and work for retrieving and rendering traces are introduced.

Visualising Traces

According to [Li and Maalej, 2012], there are fundamentally four visualisation techniques: matrices, graphs, lists, and hyperlinks, which are commonly used for visualisation, excluding lists. Lists are often used in

trace recovery process and represent each traceability link (along with the information of source, target artefacts, and other attributes) in one entry. According to [Wieringa, 1995], hyperlinks are cross-references.

– **Matrix** or a *requirements traceability matrix* (RTM) is a two dimensional representation of the relationships between traceable artefacts (usually between requirements and other artefacts). A matrix element $a_{i,j}$ being marked (e.g. checked) means that the artefact of column j and the artefact of row i are linked. Example implementations include DocTrace [Robinsons, 2014] and VisMatrix [Duan and Cleland-Huang, 2006]. DocTrace automatically creates RTMs, which show the traceability and coverage of requirements throughout the set of documents. VisMatrix creates a graphical representation of RTMs showing not only where candidate links exist, but also the strength of those links. Figure 2.13 illustrates a sample RTM in VisMatrix. The labels shown on both axes represent the names of requirements.

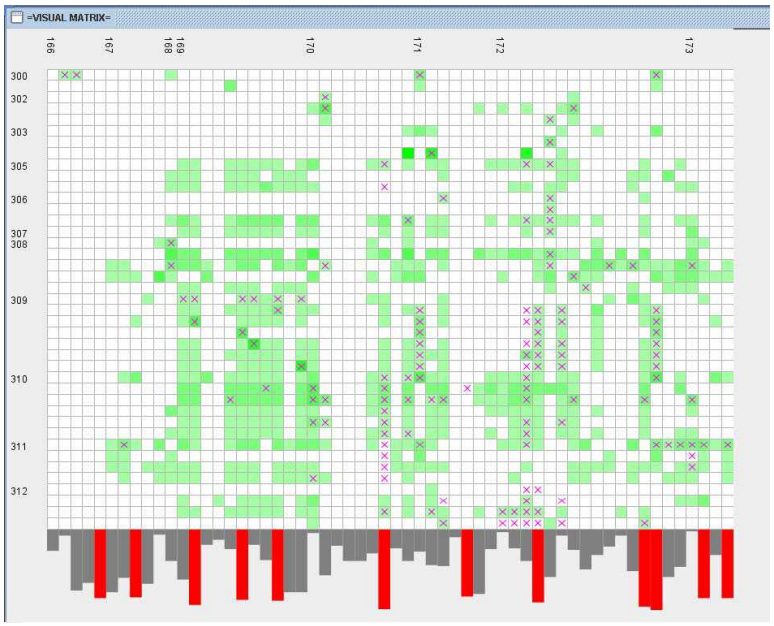


Figure 2.13: A sample traceability matrix [Duan and Cleland-Huang, 2006]

Traceability matrices can be easily generated for simple (small) traceability links and understood by stakeholders with different backgrounds. Additionally, matrices give structured overviews and are appropriate for management tasks [Li and Maalej, 2012]. However, they are not efficient for systems with complex and large number of traceability links. An important issue with matrices is that only binary links between items can be represented. Also, the semantics of

links can not be represented easily in traceability matrices.

– **Graphs** provide enriched visualization of trace links. Mainly, they allow multi-dimensional relationships between artefacts by representing artefacts as nodes and relationships between them as edges. Two nodes are connected if a traceability link exists between the corresponding artefacts.

  In context of requirements traceability, graph-based visualisation is used by different tools. However, as there is no common standard on the notation and requirements for such techniques, requirements traceability graphs are usually just plain graphical diagrams with boxes and lines. PRO-ART [Pohl, 1996a] provides a graphical *Star View Dependency Browser* which shows an artefact surrounded by its directly related artefacts, with navigation feature. TOOR [Pinheiro and Goguen, 1996] also simply uses labels and arrows to represent artefacts and links respectively. Traceline [Integrate, 2014] is a DOORS extension, which provides graph-based visualizations for requirements traceability. ChainGraph [Heim et al., 2008] visualizes shared metadata between requirements in a graph.

  On the other hand, graph-based visualisation is a common form of representing traceability links in the domain of MDE. This is because diagrams are the trivial form of visualisation of models [Winkler and Pilgrim, 2010]. Accordingly, existing notations and representation techniques are used to visualise traceability information.

  However, an important challenge in graph-based visualizations is that they often do not scale well to large data sets because their presentation result in a complex structure that is difficult to manage or understand [Heim et al., 2008].

– **Hyperlinks** connect related concepts, keywords, or phrases in a natural way. The relationships are considered as embedded pointers in an artefact (usually in textual format). Though, links between diagrams can be viewed as cross references (hyperlinks) [Wieringa, 1995]. [Maletic et al., 2003] propose a hypertext model which supports complex linking structures and versioning of individual links for link recovery. In DOORS, out- and incoming links of an artefact are visualized as bidirectional hyperlinks.

  [Li and Maalej, 2012] found out that hyperlinks demonstrate fine-grained relationships and do not provide a coherent representation of trace links. Accordingly, they are more preferred for implementation and testing tasks. Moreover, hyperlinks are always binary links while n-ary relations are required in real traceability scenarios [Sherba

et al., 2003]. Also, they do not provide any information of the type of the link or the rationale of its existence.

– **Miscellaneous** There are other techniques for visualising traces. [Marcus et al., 2005], in TraceViz for tracing between source code and external documentation, propose a map of coloured and labelled boxes to show links from and to a single artefact which is similar to the hyperlink representation. [Merten et al., 2011] suggest using Sunburst and Netmap visualizations for large amounts of traceability links. [Ratanotayanon et al., 2009] developed a prototype for associating arbitrary lines in text-based files with a feature map, called Zelda. Finally, [Pilgrim et al., 2008] propose a 3D enhancement to trace visualisation in the context of model transformation chains. Using a 3D diagram editor, all the diagrams along with their traceability information are visualised in a single view. They use layered planes to visualise traceability between different levels of abstraction.

Researchers agree that there is no single perfect way to retrieve and visualise traces and finding a suitable visualisation techniques strongly relates to the usage context [Marcus et al., 2005; Winkler, 2008; Li and Maalej, 2012]. [Winkler, 2008] indicate that it is essential to determine what the user wants to achieve when he uses a particular visualization. Accordingly, they identify three influential factors in visualisation: tasks, users, and modes of accessing information (report, search, and browse), and suggest to a catalogue of requirements induced by the different user roles and tasks should be created. Similarly, [Marcus et al., 2005] formulated a set of high level requirements for visualizing traceability links.

Retrieving Traces

In the context of requirements traceability, basic and common traceability tasks, such as coverage analysis and impact analysis, are usually provided as a feature in the tool used for traceability. For example, [IBM, 2014a] provides a feature that visualise chains of links across multiple types of artefact. However, these functionalities are primary ones and do not support non-trivial traceability scenarios. Usually, explaining complex traceability queries are provided by tool-specific APIs or by direct access to the underlying data structures. For example, Sparx Enterprise Architect^{TM} [Sparx Systems Pty Ltd., 2014] allows user-defined queries to be modelled as SQL statements on the underlying database. But, these techniques require substantial knowledge of the tool's internal data structure or of its APIs, which is not compatible with preferences of traceability users. Generally, users prefer to specify queries at the abstraction level of traceability [Mäder and Cleland-Huang, 2010].

To address this problem, several researchers have developed languages and notations to specify trace queries at the right level of abstraction. However, introduced approaches are relying on and affected by the underlying data structures and visualisation strategy used in the tool, and they need to deal with diversity of representation formats.

[Sherba et al., 2003], in TraceM, provide a query service to filter relationships to create different views of information based on stakeholders needs. [Zhang et al., 2006], in their ontology-based approach for trace recovery, use Racer query language (nRQL) [Haarslev et al., 2004] to retrieve traces. nRQL is a language to retrieve instances of concepts and roles in the ontology. However, the use of nRQL is restricted to users with a high mathematical and logical background. [Schwarz et al., 2008] define a metamodel to store traceability information, named Requirements Reference Model (RRM), then use the RRM as TGraph schema (a very general kind of graphs), and represent instances of RMM as TGraphs. Graphs constitute an abstract representation of artefacts and their traceability relationships. Then, a graph-based querying approach is used to extract traces. They use Graph Repository Query Language (GReQL) [Ebert et al., 2002] in which queries are explained in a SQL-like syntax. However, they do not provide information how their approach is implemented.

[Maletic and Collard, 2009] introduce the Traceability Query Language (TQL), a XML-based language to model trace queries for artefacts represented in XMl. TQL is build on top of XML addressing language (XPath) and TQL queries are transformed into XQuery. Extensive knowledge of XML and XPath is a prequisite of using this approach in addition to the fact that the whole approach is restricted to XML documents. Finally, [Mäder and Cleland-Huang, 2010] propose a Visual Trace Modelling Language (VTML) which allows users to model queries visually within in the context of their approach to goal-oriented traceability. VTML queries are modelled over the TIM and reuses the information previously specified in the TIM to describe a query. They show that VTML allows users to read and construct queries more easily.

In the MDE domain, to the extent of our knowledge there is no specific research on how to retrieve traces. This might be due to the fact that traceability information are represented as a general model and so general-purpose model management languages (e.g. EOL [Kolovos et al., 2006b] and OCL [Object Management Group, 2012]) can be used to retrieve model elements from trace models to support different user-defined traceability scenarios. A very simple example is to use model-to-text transformation languages (e.g. EGL [Rose et al., 2008]) to generate reports.

However, existing languages require knowledge of the structure of the trace models and also they are not at the level of abstraction of traceability users. Such users usually want to retrieve or find (traceable) elements regarding existing trace links (i.e. elements which are related to element X with a trace

link type R). They may also want to check a constraint on the elements and trace links (i.e. each element of type Y have to be traced to an element of type Z). So, they are just interested in traceable elements and trace links between them, which have been defined by a TIM. Explaining and executing such queries with general-purpose languages, particularly imperative ones, require users to explicitly specify how to apply a query on a trace model and generate an arbitrary output. Traceability users prefer to describe trace queries and have results represented in a desired format (e.g. list or a graph), regardless of how queries are implemented and how final representations are generated.

2.1.4 Traceability Tools

Requirements traceability in practice is usually supported in requirements management tools, such as DOORS [IBM, 2014a] and Rational RequisitePro [IBM, 2014b]. However, there are traceability-dedicated tools (or prototypes) which have been mostly developed in the context of a specific research or study either as a standalone traceability tool or as an extension to existing tools or framework.

[INCOSE, 2010] provides a list of the available requirements management tools which support traceability. DOORS is the most widely used tool in industry to manage requirements and record traces [Alexander et al., 2005]. Generally, these tools enable users to organize requirements and other artefacts. Artefacts can be linked to each other and to external files and thereby make them traceable, for example using hypermedia technologies. Traceability links are then visualized in a traceability matrix, as cross-references in a table-view or in a model- or graph-based diagram. However, most of the tools do not support customizable set of link types according different usage scenarios of traceability.

Regarding MDE, traceability tools are mostly available as prototypes which have been developed as a native tool or by extending the original tools [Galvao and Goknil, 2007]. The only industrial application of traceability is the bi-directional synchronization of models and code in roundtrip-engineering UML tools [Winkler and Pilgrim, 2010].

There are several research prototypes developed based on the approach presented in the related research. Table 2.1 provides an overview of these tools, which have been introduced in Section 2.1.3 in the context of each study. The table specifies the related study, the main purpose of the tool (scope), and highlights main features of the approach or tool.

On the other hand, there are studies arguing that current industrial approaches do not typically address end-to-end traceability and, accordingly, introduce custom end-to-end traceability tool. [Asuncion et al., 2007] present a software traceability tool which is used to store and manage traces throughout the entire life of a development project, from the requirements

phase to the test phase. A main contribution of their approach is to combine end-to-end requirements traceability and process traceability. Similarly, [Kirova et al., 2008] developed an integrated traceability environment, called TraceabilityWeb, based on their experience of evaluating different traceability methodologies and tools. They have mainly focused on integration with other tools in their organization, and using this tool to support their processes.

Finally, there are also some tools which provide traceability regarding requirements of specific contexts or domains. For example, [Lee et al., 2003] present ECHO, a tool-based approach to requirements engineering and traceability in agile projects. Echo creates the traceability web between customer needs and specified solutions by providing a means to capture conversations with customers, structuring them into requirements artefacts, and then including them in the information model (as traceable items). In SPL engineering, pure::variants [pure-systems GmbH, 2014] and GEARS [BigLever Software Inc, 2012] are the two leading tools which provide extensions to allow integration with other commercial traceability tools. They allows developers to integrate the functionalities provided by requirements and traceability management tools with the variant management capabilities specifically provided in these tools for SPL engineering.

Choosing the right tool support for traceability is not an easy and straightforward decision. This is because available tools are different in several ways and even focus on different aspects of traceability, such as the aspects of development process that they cover, the traceability approach, and type of artefacts. Therefore, a simple comparison between tools' features would not come to a tool by itself. The result of comparison should be examined against the capability and services required within a specific context or project. For example, the decision depends on the requirements management system already used in the organisation or project. The most appropriate tool is the one that meets stakeholders' needs, yields the benefits that are anticipated at an acceptable cost. In this respect, [Gotel and Mäder, 2012] indicate that acquiring tool support for traceability requires a systematic enquiry and accordingly present a seven-step guide for practitioners to conduct such enquiry.

Table 2.1: Research-based traceability tools

Tool	**Scope**	**Features**
TOOR [Pinheiro and Goguen, 1996]	Trace Recovery	– Pattern matching – Uses regular expressions to define patterns to be matched

Tool	Scope	Features
PRO-ART [Pohl, 1996a]	Trace Capture	– Rule-based – Traceability relations are created as a result of creating, deleting or manipulating requirements
ACTS [Asuncion and Taylor, 2012]	Trace Capture	– Rule-based – Traceability relations are created as users carry out development activities
TraceM [Sherba et al., 2003]	Trace Recovery	– Transforms implicit relationships into explicit relationships
traceMaintainer [Mäder et al., 2008a]	Trace Capture	– UML-based software development – Traceability links are captured while development activities are applied
ArchTrace [Murta et al., 2006]	Trace Maintenance	– Policy-based approach – Keeps the architecture and implementation synchronized
DocTrace [Robinsons, 2014]	Trace Visualisation	– Automatically creates requirements traceability matrix (RTM) – Coverage of requirements in the implementation

Tool	Scope	Features
VisMatrix [Duan and Cleland-Huang, 2006]	Using Traces (Visualisation & Retrieval)	– A graphical representation of RTMs – Shows candidate links and the strength of them
ChainGraph [Heim et al., 2008]	Trace Visualisation	– Graph-based – Visualizes shared metadata between requirements in a graph
Traceline [Integrate, 2014]	Trace Visualisation	– Graph-based – A DOORS extension to provide visualization for requirements traceability
TraceViz [Marcus et al., 2005]	Trace Visualisation	– A map of coloured and labelled boxes to show links from and to a single artefact – Tracing between source code and external documentation
Zelda [Ratanotayanon et al., 2009]	Trace Visualisation	– Associating arbitrary lines in text-based files with a feature map
GEF3D [Ratanotayanon et al., 2009]	Trace Visualisation	– 3D editor – Visualises model transformation chains

Tool	Scope	Features
Poirot [Lin et al., 2006]	Trace Recovery	– IR-based – Uses a probabilistic network model to generate trace links from requirement management tools
RETRO [Hayes et al., 2004]	Trace Recovery	– IR-based – Uses the LSI method and implements analyst feedback into the tracing process
LeanArt [Grechanik et al., 2007]	Trace Recovery	– Machine learning (ML)-based – Identifies trace links between use-cases and Java source code
ECHO [Lee et al., 2003]	Trace Recovery	– Requirements engineering and traceability in agile projects – Captures conversations with customers
TraceLab [Center of Excellence for Software Traceability (CoEST), 2014]	General	– A workbench for designing, constructing, and executing traceability experiments – Highly customized to support rigorous Software Engineering experiments

Tool	Scope	Features
REAMP [Schmid et al., 2006]	Trace Capture	– Software Product Line (SPL) development – An extension to DOORS for requirements modelling in SPL engineering

2.1.5 Traceability Challenges and Limitations

The benefits of an appropriate traceability approach are widely accepted [Gotel et al., 2012c]. However, empirical studies show that implementing a successful and cost-effective traceability solution is still a challenge and is usually failed in many contexts [Egyed et al., 2007]. Many researchers have investigated the difficulties or problems which hinder the adoption of traceability in practice, considering this issue from different points of view. Some of them address difficulties associated with traceability activities, while others are interested in the shortcomings of supporting technologies. There are also studies considering the cost-benefit and return on investment (ROI) of a traceability solution in general.

Additionally, [Winkler and Pilgrim, 2010], in a comprehensive survey of traceability, classify the factors which hinder the adoption of traceability in practice into four groups.

- Natural factors; attributed to the imprecise and incomplete nature of traceability (e.g. no common understanding of a complete and well-defined traceability metamodel)
- Technical factors; related to enabling technologies for a traceability solution (e.g. lack of integration with other development activities/tools and automated trace recording)
- Economical factors; concerning the Return on Investment (ROI) of traceability
- Social factors; attributed to the inevitable role of human in traceability activities (e.g. in trace recording)

Considering this thorough survey on the use of traceability in practice and also other researches focusing on specific aspect of traceability, we investigate current difficulties and challenges from four viewpoints: definition, fundamental characteristics, enabling techniques (traceability activities), and practicality or applicability (implementing a solution).

2.1.5.1 Definition

This viewpoint covers those difficulties which happen because of ambiguous and imprecise definitions in the traceability domain regardless of how traceability is implemented in a project. These difficulties are similar to the natural factors defined by [Winkler and Pilgrim, 2010]. One of the factors affecting the adoption of traceability in practice is the lack of a commonly accepted definition of traceability and inconsistency in the use of traceability terminology and concepts [Gotel et al., 2012b]. Current standards provide little guidance and the models and mechanisms vary to a large degree and are often poorly understood, so traceability in many organizations is haphazard [Ramesh and Jarke, 2001]. Addressing this issue, [Gotel et al., 2012b] provide a resource for traceability fundamentals along with a glossary of traceability terms and concepts. Such work will provide a common view of traceability and help the research community and industry to structure and understand problems better.

Additionally, researchers highlight the need for a standard way of specifying a traceability metamodel, mainly to support interoperability between traceability solutions, and hence, propose reference models or formal specification of traceability metamodels (at the metametamodel level), such as [Ramesh and Jarke, 2001; Limon and Garbajosa, 2005; Espinoza et al., 2006] (introduced in Section 2.1.3.2).

2.1.5.2 Fundamental Characteristics

We argue that there are some challenges which are inherent to traceability and which cannot be eliminated completely. To address these challenges, a traceability approach has to convince users of the benefits of having traceability and control or dominate the negative effects of them in some way.

Fundamentally, supporting traceability requires substantial effort to create and maintain traceability relations. Therefore, traceability is always a costly activity and it is not easy to measure the return on investment (ROI) of traceability [Palmer, 1999]. A practical solution needs to provide a less effort-intensive approach for creating and updating trace relations or increase the benefits of traceability to compensate its additional costs. In this context, a part of current research focuses on techniques to create, maintain, and use traces, and try to improve them. Such studies are discussed in Section 2.1.5.3.

On the other hand, some researchers address the additional cost of traceability at the higher level, focusing on the overarching strategy which drives traceability activities. They provide guidelines and standard templates supporting the whole planning activity for traceability. The primary idea behind these approaches is to tailor and customise a traceability solution (and consequently traceability activities) to the needs of a particular project. This

approach has been acknowledged in various contexts.

[Dömges and Pohl, 1998; Pinheiro, 2003] note that traceability methods and tools have to be easily customised in order to lower traceability costs. [Mäder et al., 2009a] advocate defining traceability metamodels specifically for a project at hand and propose practical guidelines accordingly. [Aizenbud-Reshef et al., 2005] state that an optimal traceability metamodel is the one that is customisable and extensible by the user. [Heindl and Biffl, 2005] suggest value-based requirements traceability (VBRT) to identify which traces are more important and valuable than others. Similarly, [Egyed et al., 2005] propose a value-based approach to assist engineers into deciding which traces are needed, when they are needed and at what level of precision, completeness and correctness. They show that the approach reduces the cost of traceability by avoiding unnecessary trace recovery and maintenance. Cost of traceability can also be reduced by tailoring the precision, completeness and correctness of trace links depending on their intended usage. In a related work, [Egyed et al., 2007] conducted three case studies to examine the trade-off between these three attributes and traceability cost. The results of these case studies show that cost and effort of traceability can be reduced by reducing the granularity of the traceable artefacts.

[Lago et al., 2009] present a scoped approach to traceability instead of automating tracing, or representing all possible traces. They scope the traces to be maintained to the activities stakeholders must carry out. They define core traceability paths, consisting of essential traceability links required to support the activities. They illustrate the approach through two examples: product derivation in software product lines, and release planning in software process management.

[Ingram and Riddle, 2012] focus on cost as the key reason to neglect or abandon traceability efforts, and identify key issues to maximise the cost-benefit of a traceability solution (optimal scenario). They introduce three strategies to minimise the cost of gathering trace data: establishing traceability goals, trace creation and evolution, and using automated tools. Moreover, they state that traceability data should be at an "acceptable quality" in order to reach the cost-benefit of traceability. The *quality* of data is defined as a function of the granularity, recall and precision, and level of coverage of trace links, which are specified regarding the given project.

On the other hand, humans inherently play an important role in capturing and maintaining traceability as traceability practices can never be fully automated [Egyed and Grünbacher, 2005]. Even in the relatively formal context of model driven software development, fully automated establishing and maintaining traceability links is still an issue [Winkler and Pilgrim, 2010]. In this context, it is important to motivate users and convince them of the benefits of having traceability. Empirical studies show that direct, immediate, and short-term benefit motivate users to carry out traceability activities [Alexander, 2002; Ebner and Kaindl, 2002; Arkley and Riddle,

2005].

Additionally, [Hoffmann et al., 2004] argue that traceability acceptance will increase if supporting tools are made more usable. [Arkley and Riddle, 2005] also indicate that traceability activities should be designed in a way that can be integrated with the way users work. Otherwise, they would be imposed on them and consequently would be discarded. Moreover, [Hayes and Dekhtyar, 2005] argue that users do not trust the traces produced by automated traceability methods and many times they make the results of such methods worst, which can happen because of missing motivation or lack of understanding of traceability.

2.1.5.3 Enabling Techniques

These challenges are related to techniques and methods used to create, maintain, and use traces in a project. The ultimate goal of the related research is to improve the efficiency, usability, or productivity of traceability activities in various ways. Some of them try to minimise manual activities (human interaction) and so to automate traceability activities as much as possible, for example through using IR- and rule-based methods for trace recovery. Other studies are interested in enhancing the recall and precision of traces. Finally, several efforts have been spent on improving tool support for traceability. The current and ongoing research in this context were introduced, and their strengths and weaknesses were discussed in Section 2.1.3.

2.1.5.4 Practicality and Applicability

This viewpoint focuses on problems which are related to the practicality or applicability of a traceability approach; when a traceability solution is implemented and used.

Experimental studies of existing tracing approaches *in practice*, observe that usually the traceability metamodel is undetermined [Von Knethen and Paech, 2002]. It has not been precisely defined what artefacts are traced and what trace links are captured. Also, selecting the right level of granularity for traceability is a challenge [Kirova et al., 2008]. It is difficult to determine what level of granularity is appropriate and useful for a given project. The granularity varies depending on the usage scenario. For example if there is a lot of in-team knowledge and experience in architecting or designing specific products within the domains, there can be less emphasis on requirements to design traceability, hence a higher level of granularity will be acceptable. So, support for selecting granularity, quality, and completeness of the traceability metamodel is essential. [Mäder et al., 2009a] also acknowledge the lack of *practical* guidance on how to design, implement, and use project-specific traceability metamodels as one the main reasons why such metamodels are not used in practice, though they are considered and discussed as the core

element of any traceability solution.

Additionally, tracing approaches do not provide sufficient and clear process support [Von Knethen and Paech, 2002; Winkler and Pilgrim, 2010]. Generally, traceability is not directly supported by software development process [Ramsin and Paige, 2008]. Therefore, tracing approaches have to clearly specify when and how traceability activities are actually carried out during software development.

Insufficient tool support has been identified as a factor which hinders the adoption of traceability in practice [Von Knethen and Paech, 2002; Kirova et al., 2008; Winkler and Pilgrim, 2010]. Studies show that automated trace creation and maintenance are essential in adopting a traceability solution, as recording and using traces manually are time-consuming and error-prone tasks. The studies also highlight the lack of interoperability between tools. In industry, there is a need for a comprehensive approach to traceability which considers interoperability between requirements management tools, modelling tools, integrated development environments (IDEs), and communication tools. The tooling support could implement standardized interfaces or would support a common trace data format.

2.1.6 Traceability in Specific Domains

Previous sections introduce and discuss traceability generally in software and systems development. Nevertheless, there are several researches focusing on traceability in specific domains. They advocate that an effective traceability solution has to be designed and implemented according to a particular project's need. These studies investigate the domain or context, locate the role of traceability in that domain, identify the specific needs, and accordingly propose a complete traceability approach or an element of a solution.

Traceability has been widely investigated in those areas in which traceability is considered as a mandatory or essential requirement, such as NFRs, safety-critical systems, and software product lines. This section gives a brief introduction of existing research in these domains. Particularly, due to the context of the running example (Section 1.1.1) and the case study (chapter 6), we spend more on the traceability in the context of safety-critical systems.

2.1.6.1 Tracing Non-functional Requirements

NFRs traceability has been considered in different phases of software development process, for different purposes. In early phases of software development, tracing NFRs provides support for software project planning and control [Ramesh and Edwards, 1993] and can help architects determine whether all NFRs have been fully covered in the proposed design, and con-

versely identify unresolved concerns [Tang et al., 2010]. The ability to connect each design and implementation element back to design decisions and its related NFRs can also provide capabilities for verifying and evaluating the completeness of the design [Tekinerdogan et al., 2007b]. Additionally, existing traces from NFRs to the design provide support for change impact analysis [Mirakhorli and Cleland-Huang, 2011]. NFR traceability is also an integral part of documenting the architectural decisions and their relationship to quality goals and NFRs [Tang et al., 2007].

Nevertheless, NFRs are more difficult to trace than functional requirements as they often exhibit cross-cutting and wide impacts across the system and general traceability approaches do not address challenges of tracing NFRs directly or explicitly. [Mirakhorli and Cleland-Huang, 2012] have analysed and explored existing techniques for tracing NFRs, and then identified the fundamental issues related to tracing NFRs. They indicate that NFRs traceability is multi-level, multi-path, heterogeneous, multi-granularity, tacit traces, trade-offs, semantically typed, strategic, minimalistic.

Fundamentally, NFRs play a strategic role in driving the architectural design of a software intensive system. In this regard, a common approach to tracing NFRs is to explicitly specify relationships between NFRs and downstream work products through existing architectural analysis and management processes. Four representative software engineering activities that incorporate the creation and utilization of NFR traceability links are the Architectural Tradeoff Analysis Method (ATAM) [Kazman et al., 2000], Architectural Documentation (e.g. Views-and-Beyond [Bass et al., 1998]), Enterprise Architectural Frameworks (e.g. C4ISR developed by the U.S. Department of Defense (DoD) [U.S. Department of Defense (DoD), 1997]), and management of architectural knowledge [Vliet, 2008].

This way, traceability information is embedded into the architectural documents and therefore it is difficult to use trace links to support other activities. However, building traceability techniques on top of such methods demonstrates the benefits of traceability efforts quickly. On the other hand, architectural management tools provide rich environments for tracing between designs decisions, rationales, and other supporting information, but have not yet been integrated with architectural modelling tools [Tang et al., 2010]. They also support only relatively high level trace links; no finer grained traceability between NFRs and specific design or code elements.

There are several other tracing approaches which are specifically designed for creating and maintaining NFR traces. Another approach is Goal-Centric Traceability (GCT) [Cleland-Huang et al., 2005a; Cleland-Huang, 2005] which provides traceability support for managing and maintaining NFRs and their related quality concerns over the long-term life of a software intensive system. GCT is a goal-oriented approach which assumes that quality concerns are modelled in an arbitrary goal hierarchy. [Cleland-Huang and Schmelzer, 2003; Song et al., 2011] suggest recording traces from NFRs to

software designs simultaneously while applying existing design and architectural patterns. They assume that, in most cases, non-functional requirements provide solutions in pattern. The concepts of Aspect Oriented Requirements Engineering (AORE) provide an promising framework for tracing NFRs. Therefore, several researchers have explored ideas of using early aspects to trace NFRs including [Tekinerdogan et al., 2007a; Kassab and Ormandjieva, 2006]. [Kassab et al., 2009] propose a metamodel which explicitly captures the concepts of NFRs, FRs, and their relationships, and it is independent of any programming paradigm. It can be customised and instantiated with respect to the required programming type. The metamodel is ultimately transformed into a rational model and Datalog is used to implement queries to represent traceability information.

A main advantage of these techniques is that they provide wider usage of traceability links in comparison to the architecture-centric approach. However, they require specific modelling environments or development practices and suffer from scalability issues. A number of approaches, such as [Salazar-Zárate et al., 2003; Krishna and Gregoriades, 2011], develop UML profiles and extend UML models to integrate NFRs into functional behavioural models. Considering advantages and disadvantages of the above mentioned approaches, [Mirakhorli and Cleland-Huang, 2012] introduce a hybrid approach, called Architectural Centric Traceability (ACT), which is designed specifically for tracing NFRs across the software development life cycle, but also closely integrated into common architectural assessment and analysis techniques [Burge and Brown, 2008]. ACT provides traceability support for preventing the typical architectural erosion and quality degradation that occurs during long-term system maintenance and evolution of a software system.

2.1.6.2 Traceability in Safety-Critical Systems

Traceability is especially essential for critical systems which must satisfy a range of functional and non-functional requirements, including safety, reliability and availability [Mason, 2005].

Additionally, traceability is an important component in all the standards available for different industries, including DO-178B [RTCA and EUROCAE, 1992] for the Aerospace industry, Automotive SPICE [IEC, 2012] and ISO 26262 [ISO, 2011] for Automotive industry, and IEC 60880 [IEC, 2008] and IEC/TR 61508 [IEC, 2005] in nuclear power generating software. They routinely require full lifecycle traceability to assist in evaluating such systems. Traceability from hazards to derived safety requirements and to implemented and verified design solutions provides essential evidence to argue that a system is operationally safe [Lutz, 2000]. However, in practice traceability links provided by software developers are usually incomplete, inaccurate, ineffective for demonstrating software safety.

[Katta and Stlhane, 2012] address the need for traceability methods tailored to safety systems and propose a conceptual model of traceability for safety systems covering both the development process and safety analysis processes. The conceptual model presents the data generated during development and the safety analysis phases, and the relations between them. The model is the core part of a traceability management approach in safety systems, which ultimately generate safety cases. The model can also be used as a base line for communication between developers and safety analysts.

Additionally, [Cleland-Huang et al., 2012] proposes a family of reusable traceability queries as a blueprint for traceability in safety-critical systems. The queries that consider formal artefacts, designed to help demonstrate that: 1) identified hazards are addressed in the safety-related requirements, and 2) the safety-related requirements are realized in the implemented system. Practitioners building safety critical systems can use these trace queries to make their traceability efforts more complete, accurate and effective.

In the context of the MeMVaTEx project, [Albinet, 2008; Peraldi-Frati and Albinet, 2010] introduce a model-based methodology for requirements expression, traceability, and verification in real-time systems development. Their methodology relies on the EAST-ADL2 framework [ATESST2 Consortium, 2010] and two of the UML2 profiles: MARTE [Object Management Group, 2011a] for real-time embedded systems and SysML [Object Management Group, 2010a] for system requirements modelling. The methodology defines different models used at each abstraction level of the process. The results are a requirement model and a solution model which is related to the requirements. Verification and validation models and techniques are connected to these models. The key feature of their methodology is developing a dedicated UML profile as a DSL which imports a subset of needed stereotypes from mentioned profiles.

On the other hand, safety standards mandate generation of product-specific evidence demonstrating the satisfaction of safety requirements and that the system is acceptably safe to operate [Habli and Kelly, 2006], which requires the creation of a safety case for the system [Despotou and Kelly, 2008]. A safety case is a document which consists a set of various deliverables, including diagrams, tables, and texts, to show that a system is acceptably safe in the operation context [Kelly, 1998]. Therefore, gathering the evidence for the safety case is very critical. A safety case needs to be linked properly to other related concepts in safety engineering process, such as safety models, safety analysis, and verification and validation. Although the relationships are specified through the safety case notations, they are not well connected and they are defined in a totally separate context [Despotou and Kelly, 2008].

[Mason et al., 2003; Mason, 2005] address the problems of establishing and maintaining traceability and consistency across disjoint safety engineering tools to provide valid evidence for safety case. They introduce MATra, the

Meta-modelling Approach to Traceability, for managing the engineering of aerospace systems. The framework is realised by exporting information from the internal models of tools to an integrated environment consisting of 1) a set of *metamodels* expressing the application domain, 2) *well-fromedness* constraints over the metamodels, and 3) *associations* between the metamodels. However, the framework is limited to the safety domain and intends to overcome the poor integration between safety tools and the consequent inconsistencies.

[Panesar-Walawege et al., 2010] provide a conceptual model that characterizes the evidence necessary for arguing about software safety. Their model captures both the information requirements for demonstrating compliance with IEC 61508, and the traceability links necessary to create a seamless continuum of evidence information, called the chain of evidence.

2.1.6.3 Traceability in Software Product Lines

Software Product Line (SPL) Engineering has to explicitly deal with interrelated, complex models such as feature and architecture models, hence traceability is fundamental to keep them consistent. It is required both between different software models (e.g., feature and structural models), and between different development phases (e.g., from domain engineering to application engineering).

[Ajila and Kaba, 2004] present traceability mechanisms to support software product line evolution. The product line approach to software development requires designers to consider requirements for a family of products and the relationships between these requirements. They examine three kinds of evolution processes -architecture, product line, and product to define change management mechanisms. These mechanisms share four strategies change identification, change impact, change propagation, and change validation. Then, an evolution model based on dependency relationships structure of the various product line artefacts is developed.

[Jirapanthong and Zisman, 2005] advocate the use of traceability relations to support SPL engineering. Relations between product members and the product line architecture, and among the product members themselves must be automated. For this purpose, a rule-based approach for generating traceability relations among different types of artefacts is presented. Accordingly, ten different types of traceability relations (between the various documents) are identified among feature, sub-system, process, module, use case, class diagram, statechart diagram and sequence diagram models. An extended version of XQuery is used to represent the rules and to support extra functions to cover some of the traceability relations being proposed.

[Anquetil et al., 2010] present a model-driven traceability framework to software product line development. Model-driven techniques are adopted with the aim to support the flexible specification of trace links between dif-

ferent kinds of SPL artefacts. A traceability metamodel is defined to support the flexible creation and storage of the SPL trace links. The framework is organized as a set of extensible plug-ins that can be instantiated to create customized trace queries and views.

Finally, a survey on existing traceability tools was conducted in the context of the AMPLE project [AMPLE, 2007]. The objectives of this survey were to investigate the current features provided by existing tools in order to assess their strengths and weaknesses and their suitability to address SPL development. The tools were evaluated in terms of the following criteria: (i) management of traceability links; (ii) traceability queries; (iii) traceability views; (iv) extensibility; and (v) support for SPLs and MDE. The conclusions that were drawn from their survey were that none of the investigated tools had built-in support for SPL development, and a vast majority of them are closed, so they cannot be adapted to deal with the issues raised by SPL.

There is some recent progress in providing traceability support for product lines. Two of the leading tools in SPL development, pure::variants [pure-systems GmbH, 2014] and GEARS [BigLever Software Inc, 2012] have defined some extensions to allow integration with other commercial traceability tools. Pure::variants includes a synchronizer for CaliberRM and Telelogic DOORS that allows developers to integrate the functionalities provided by these requirements and traceability management tools with the variant management capabilities of pure::variants. Similarly, GEARS allows importing requirements from DOORS, UGS TeamCenter, and IBM/Rational RequisitePro. [Schmid et al., 2006] focus on tool support for SPL engineering and discuss how a seamless integration of product line concepts in existing tools is possible and desirable. When analysing the requirements a tool extension for product lines must support, traceability support is considered as mandatory. Thus, the authors introduce in REMAP (an extension to DOORS for requirements modelling in SPL engineering) an explicit and automatically generated trace between a requirement in the product line and a requirement in the product: if a requirement is added by instantiation to a product, REMAP creates a link from the requirement in the product to the respective requirement of the product line infrastructure.

However, these external tools (e.g. DOORS, RequisitePro) handle traceability for traditional systems. Apart from their individual weaknesses (see Table 1), they all lack the ability to deal explicitly with specificities of SPL development, for example, dealing with variability. They do not provide advanced and specific support to deal with change impact analysis or feature covering in the context of SPL development.

2.1.7 Traceability in Different Engineering Approaches

Traceability has been mandated by military regulations and thus, it has been part of waterfall-drived development processes. Today, however, traceability

has gained in significance in any development approach. In this section, we briefly discuss the role of traceability and how it is considered in different engineering approaches including V-model (an extension to waterfall model), agile software development, and enterprise architecture frameworks.

2.1.7.1 V-Model

The V-Model is an extension of the waterfall model of software development, which is the traditional development process model. In the V-model, development activities are basically performed sequentially, but the process steps are bent upwards after the coding phase, to form the typical V shape (illustrated in Figure 2.14) [Sommerville, 2007]. The model specifies two groups of activities: the development process (left side) and testing activities (right side). For each stage in the development process, there is a related testing activity.

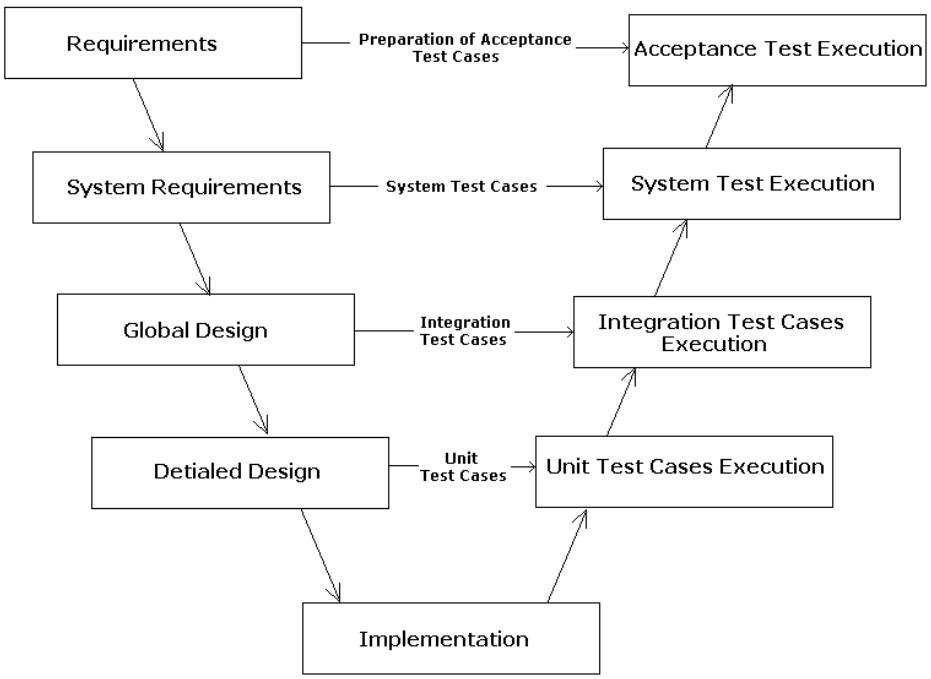


Figure 2.14: The V-model of software system development

As mentioned before, waterfall-based process models are usually used in developing critical (high-integrity) systems. For example critical software systems processes usually follow the V-model of software development [Ge et al., 2010]. This is due to the special requirements of such project, such as

each phase and its documents needs to be approved so that next phase can start. Accordingly, traceability is a mandatory attributes of such development processes because of the regulations and standards in these context.

Additionally, the model requires traceability in a slightly different way from the normal traceability mandated in general (between requirements, design, implementation, and test units). Throughout the system development, users need to explicitly specify that each development activity has a corresponding testing activity and vise versa, which in other words, is to provide traceability at the process level, between activities. However, to our knowledge, current research has not explicitly considered traceability in such projects. This might be because of the fact that, in such projects, traceability is a mandatory requirement regardless of how it is provided and how much it costs; there is not any specific constraints on traceability practices (for example in comparison to agile projects).

2.1.7.2 Agile Software Development

Agile software development is a style of software development in which both the requirements and the delivered solution evolve incrementally through a series of iterations. Such methodologies are characterized by an emphasis on human interactions and collaborations, lightweight development processes, frequent deliverables, and minimal documentation [Ambler and Jeffries, 2002; Schwaber and Beedle, 2001; Beck and Andres, 2004]. XP [Beck and Andres, 2004] and Scrum[Schwaber and Beedle, 2001] are two well-known and commonly used agile methodologies in industry.

Traceability is generally perceived by agile developers as an additional, heavy-weight activity which returns little value to the project [Cleland-Huang, 2006]. However, traceability fundamentally could benefit such projects in many ways. [Appleton, 2005] identified several reasons for tracing in an agile project including change impact analysis, product conformance, process compliance, project accountability, baseline reproducibility, and organizational learning. On the other hand, agile practices are increasingly adopted in larger, distributed, and safety-critical projects [Paige et al., 2011], and in these environments the benefits of traceability outweigh its costs.

Accordingly, traceability has been considered in the context of agile development, also under the term *agile traceability*. As discussed in [Cleland-Huang, 2012], what differentiates agile traceability from traceability in non-agile projects is the way in which traceability practices are provided (implemented and used). For example, [Lee et al., 2003] discuss that successful traceability, in agile projects, has to start as early as possible and the method for creating and maintaining traceability must be non-intrusive to the development team. They present a tool-based approach which deliver transparent model in which traceability is implicitly maintained while the content is created. [Appleton et al., 2007] describes several techniques for

light-weight tracing such as utilizes existing configuration management tools in order to capture traceability links, as configuration management is a natural part of an agile project. Another approach is to generate traces as a by-product of the normal agile process at low cost. Just-in-time Traceability (JITT) follows this approach and uses information retrieval techniques to automatically create candidate traceability links. In this way, no permanent traces are stored or maintained, and the primary cost and effort of the trace is incurred at the time of need (when traces are retrieved). Finally, [Cleland-Huang, 2012] suggests a very basic traceability information model which only includes the most essential concepts in agile development, such as requirements, test case, and code. Though, it is acknowledged that this model is not scalable and according to the size and longevity of the project, different TIMs are required which is highlighted in [Espinoza and Garbajosa, 2011].

2.1.7.3 Enterprise Architecture Framework

Enterprise Architecture (EA) is a well-defined practice for conducting enterprise analysis, design, planning, and implementation, using a holistic approach at all times, for the successful development and execution of a strategy [Federation of EA Professional Organizations, 2013]. An architecture framework is a set of structures for developing a wide range of different architectures in order to design a target state of an enterprise in terms of a set of building blocks (represented by architectural views). It specifies a list of recommended standards and compliant products that can be used to implement the building blocks.

MODAF and TOGAF are two well-known frameworks for developing an EA. The Ministry of Defence Architecture Framework (MODAF) [The UK Ministry of Defence, 2012] is the recognised enterprise architecture framework developed by the UK Ministry of Defence (MoD) based on the US Department of Defence Architecture Framework (DODAF). The Open Group Architecture Framework (TOGAF) [The Open Group, 2011] is developed and maintained by members of The Open Group [1], based on the Technical Architecture Framework for Information Management (TAFIM), developed by the US Department of Defense (DoD). Also, the Reference Model of Open Distributed Processing (RM-ODP) [ISO/IEC and ITU-T, 1998] which is a reference model to describe an open distributed processing (ODP) system, introduces an enterprise architecture framework for the specification of such systems. RM-ODP is a joint effort by ISO/IEC and ITU-T.

One of the responsibilities of an EA is to provide complete traceability from requirements analysis and design artefacts, through to implementation and deployment. This is because the architecture needs to be understood by all participants and not just by technical people.

[1] www.opengroup.org

However, considering TOGAF 9.1 specification, it is observed that traceability is rarely referred to and the only sections where it is used is in the various architecture domains where a requirements traceability report or traceability from application to business function to data entity has to be created. TOGAF specification does not provide specific techniques to provide traceability. Its core metamodel also provides limited number of architectural content to support traceability across artefacts. However, as discussed in [Thorn, 2013], in order to support traceability, it is possible to use an enterprise architecture tool with TOGAF artefacts, and thereafter, use matrices and diagrams to build the required traceability.

In MODAF, traceability is considered in the context of mappings used to relate various concepts in different view points together. The MODAF is organised into seven viewpoints, each of which contains several architecture views. It mostly uses traceability matrices to specify the relationships between concepts. For example, in the 'System Viewpoint', the Function to Operational Activity/System Function Traceability Matrix is a specification of the relationships between the set of operational activities/system functions applicable to an architecture and the set of functions applicable to that architecture. Additionally, the 'All View Viewpoint' demonstrates an overarching description of the architecture and allows to search and query architectural models. Thus, this viewpoint implicitly provides traceability between viewpoints. Nevertheless, the MODAF tools are model-driven and the MODAF Meta Model (M3) is the reference model that underpins MODAF which is implemented as a profile of UML 2.1; architectural views are represented as UML models and therefore traceability practices introduced in MDE domain can be applied if required.

Similarly to MODAF, RM-ODP provides an EA framework based on viewpoints; it describes a distributed application according to five viewpoints. Each viewpoint is associated with a viewpoint language that specifies the vocabulary and presentation of that viewpoint. Although viewpoints are separately specified, they are not completely independent and key concepts in each are identified as related to concepts in the other viewpoints. In this context, RM-ODP defines *correspondences* between viewpoints which support mutual consistency between them, in addition to specifying the relationship between viewpoints. Correspondences are specified with the Enterprise Language which are statements that relate the various different viewpoint specifications, but do not form part of any one of them. UML4ODP [ISO/IEC and ITU-T, 2009] is a set of UML profiles which allows ODP modellers to use the UML notation for expressing their ODP specifications based on RM-ODP.

2.2 Model-Driven Engineering

The proposed approach to traceability in this thesis, which is explained in Chapter 4, is based on the principles of MDE. In this section, we briefly introduce MDE and its principles relevant to and used in the proposed approach.

MDE is an approach to address the inability of third-generation languages to alleviate platform complexity and express domain concepts effectively [Schmidt, 2006]. In MDE, models are the primary artefacts which are constructed and manipulated throughout the engineering process. Accordingly, software engineers using MDE particularly work with specific types of artefact, such as models, metamodels and model transformations. Furthermore, MDE involves new development activities usually referred as model management operations. In the following, the artefacts and activities involved in MDE are briefly explained. Finally, regarding the proposed approach in this thesis, modelling in the large is discussed shortly.

2.2.1 Models

Models are the primary concept in the MDE. Different definitions of the term model are provided in the domain of computer science (e.g. [Starfield et al., 1990; Ludewig, 2003; Henderson, 2003]) and in the domain of MDE, in particular (e.g. [Bézivin and Gerbe, 2001; Seidewitz, 2003]). [Kurtev, 2004] reviewed and identified commonalities and variations between these definitions.

Considering the fundamental purpose of MDE, the particular strength of models is based on the idea of abstraction, as recognised in [Ludewig, 2003]. Abstraction is a purposeful simplification of the reality; representing related details and discarding irrelevant details from what is being modelled. Accordingly, a model is *an abstraction of something that exists in reality* [Kleppe et al., 2003]. The reality is referred to as *the original* in [Ludewig, 2003] and it could be an object, phenomenon, a system, a system of systems, or a domain.

[Kurtev, 2004] also acknowledges that the relation between a model and its original is highlighted in many of the definitions. [Ludewig, 2003] considers this property under the *mapping* criterion and [Seidewitz, 2003] refers to it as the interpretation of a model. By interpreting a model, model elements are mapped to the elements of the *original* being modelled. So that an engineer can determine the truth of the model and perceive the model meaning relative to the original.

Models can be structured (presented in a well-defined language) or unstructured (an informal drawing). However, in software engineering, and hence, in MDE, models are structured rather than unstructured [Kolovos, 2008] which may have either a textual or graphical representation [Kolovos

et al., 2006b].

2.2.2 Modelling Languages and Metamodels

Models are expressed by *modelling languages* [Seidewitz, 2003]. A modelling language is a set of syntactic and semantic constraints used to define a model. In MDE, a modelling language is often described as a model and, hence the term *metamodel* is used in place of modelling language. However, the two terms are slightly different: a *metamodel* defines the structure of a group of valid models expressed by a certain modelling language [Seidewitz, 2003].

A model *conforms* to a metamodel when the metamodel specifies every concept used in the model definition, and the model uses the metamodel concepts according to the rules specified by the metamodel. Conformance can be described by a set of constraints between models and metamodels [Paige et al., 2007]. When all constraints are satisfied, a model conforms to a metamodel.

In MDE, a modelling language is commonly specified by a concrete syntax, an abstract syntax and semantics [Rose, 2011].

- **The concrete syntax** provides a notation for constructing models that conform to the language. For example, a model may be represented as a collection of boxes connected by lines. A standardised concrete syntax enables communication. Concrete syntax may be optimised for consumption by machines (e.g. XML Metadata Interchange (XMI) [Object Management Group, 2007]) or by humans (e.g. the Unified Modelling Language (UML) [Object Management Group, 2010b]).

- **The abstract syntax** defines the concepts described by the language, such as classes, packages, datatypes. The representation for these concepts is independent of the concrete syntax. For example, the implementation of a compiler might use an abstract syntax tree to encode the abstract syntax of a program (whereas the concrete syntax for the same language may be textual or graphical).

- **The semantics** identifies the meaning of the modelling concepts with respect to the domain. For example, consider the tree construct which is a usual construct in modelling. The semantics of a tree in a language is likely to be different from the semantics of a tree in others. The semantics of a modelling language may be specified rigorously, by defining a reference semantics in a formal language such as Z [Information Technology ISO/IEC, 2002], or in a semi-formal manner by employing natural language.

Metamodels are essential in domain-specific modelling where metamodels define the relationships among concepts in a domain and precisely spec-

ify the key semantics and constraints associated with these domain concepts [Schmidt, 2006].

Metamodels also facilitate model interchange [Gitzel and Korthaus, 2004] and, consequently, interoperability between modelling tools. Metamodels are expressed in some modelling language. In this context, specifying metamodels with a common modelling language ensures consistency in the way in which modelling constructs are specified and supports the construction of interoperable MDE tools. To facilitate interoperability between MDE tools, the OMG has standardised a language for specifying metamodels, the Meta-Object Facility (MOF) [Object Management Group, 2011b]. Metamodels specified in MOF can be interchanged between MDE environments. MOF is sometimes called as a metamodelling language, as it is a modelling language for describing modelling languages.

2.2.3 Model Management Operations

In MDE, models are manipulated throughout the engineering process in order to produce a product. This set of model-related operations is referred to as model management [Bernstein and Melnik, 2007], which covers all kind of manipulations such as model transformations, refactoring, comparisons, merging, and validation. Typical model management operations are described in the following, particularly those are relevant to this thesis, including model transformation, validation, and composition/merging/weaving.

2.2.3.1 Model Transformation

Model transformation is one of the integral parts of MDE [Schmidt, 2006]. A model transformation is an operation which automatically generates a number of output artefacts (targets) from a number of input artefacts (sources), according to a given transformation specification [Kleppe et al., 2003]. Although model transformations have to involve models, either as input or output, in the literature, a broad range of software development artefacts are considered as potential transformation artefacts such as texts, specifications, and program code [Czarnecki and Helsen, 2006].

Figure 2.15 gives an overview of the main concepts involved in a model transformation. The figure shows the simple scenario of a transformation with one input (source) model and one output (target) model. However, in general, the input and output could be a non-model artefact and the transformation may have more than one source and target. The transformation is defined with respect to the metamodels (or the structure of the involved artefacts, in case of non-model artefacts). The definition is executed on concrete models by a transformation engine. Transformations are usually specified as a set of transformation rules. Each rule defines the way in which

a set of elements in the source is transformed to an equivalent set of elements in the target [Kolovos, 2008].

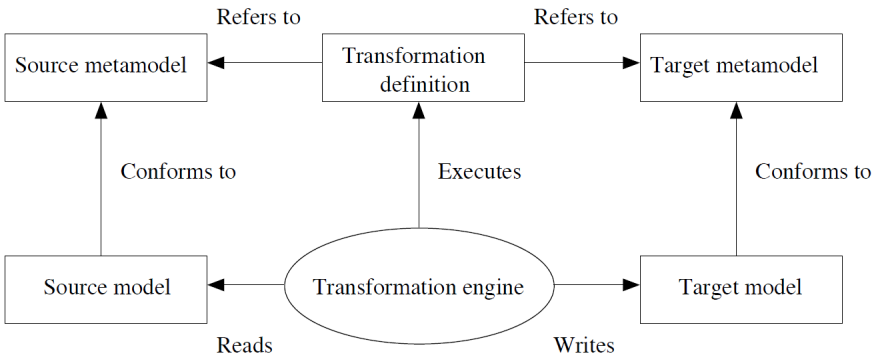


Figure 2.15: Basic concepts of model transformation [Czarnecki and Helsen, 2006]

Model transformations are usually described with transformation languages specially tailored for transforming models [Guerra et al., 2010]. In practice, there are many different types of transformation languages. [Czarnecki and Helsen, 2006] presents a comprehensive, hierarchical feature model to categorise different model transformation approaches. [Clark et al., 2004] identify the important choices that need to be made in designing a transformation language. Some of the features are highlighted over the others and, hence, transformation languages are largely classified according to their paradigm and directionality:

– *declarative* or *imperative*: Declarative transformation languages only provide mechanisms for mapping source to target model elements and the execution of rules are determined by the transformation engine. In contrast, imperative transformation languages operate at a low level of abstraction and users explicitly specify how transformation rules are scheduled, as well. Each of these two approach demonstrates particular advantages and shortcomings. Declarative languages are generally limited to scenarios in which the transformation is a matter of a simple mapping, while imperative transformation languages support a wider range of transformation scenarios. However, such languages require users to manage the execution process manually which can be hard to develop and maintain [Kolovos, 2008].

  Addressing these shortcomings, *hybrid* languages have been introduced which provide both a declarative rule-based execution approach be-

sides imperative features for handling complex transformation scenarios [Kolovos et al., 2008].

– *unidirectional* or *bi-directional*: Unidirectional transformations can be executed in one direction only, in which case a target model is computed (or updated) based on a source model. Bi-directional transformations can be executed in two directions, which is particularly useful in the context of model synchronization. Bi-directional transformations can be achieved using bi-directional rules or by defining several separate complementary unidirectional rules, one for each direction [Czarnecki and Helsen, 2006].

Additionally, model transformations can be categorised into three groups: model-to-model, model-to-text, and text-to-model transformations [Rose, 2011]. The difference between the first two groups is that a model-to-model transformation generates models –an instance of a target metamodel, while the target of a model-to-text transformation is just texts. A text-to-model transformation specifies transformation from textual artefacts to models which usually comprises parsing technologies. Each type of transformation has unique characteristics and tools, though they share common characteristics. Accordingly, different languages have been developed and introduced for each type.

Model-To-Model (M2M) Transformation. M2M transformations are used to generate models from other models and are the most widely used type of transformation. This is because, in a MDE process, high-level models are refined through transformations and, hence, it is an essential need to transform models between different languages and levels of abstraction [Guerra et al., 2010].

Many M2M transformation languages have been proposed, such as the Atlas Transformation Language (ATL) [Jouault et al., 2006], VIsual Automated model TRAnsformations (VIATRA) [Varro and Balogh, 2007], and the Epsilon Transformation Language (ETL) [Kolovos et al., 2008]. Queries/Views/Transformations (QVT) [Object Management Group, 2011c] is a standard for M2M transformation.

Model-To-Text (M2T) Transformation. M2T transformation is used to generate text which could be a piece of code or a document. It is also used for model serialisation (enabling model interchange). There are various M2T languages including MOFScript [Oldevik et al. 2005], the Epsilon Generation Language (EGL) [Rose et al., 2008], which have been developed according to the standard of M2T languages published by Object Management Group.

One of the main requirements of M2T transformation languages is to provide mechanisms for specifying sections of text that will be completed manually and must not be overwritten by the transformation engine.

Template-based M2T transformation is the common approach in existing languages. A *template* usually consists of the target text containing *dynamic* sections which are written in specific language (syntax). These sections describe how the output text should be generated; they access information from the source and generate the output through iterative expansion. The contents of static sections are directly written to the output stream when the transformation is invoked.

Text-To-Model (T2M) Transformation. T2M transformation is most often implemented as a parser that generates a model rather than object code. Parser generators such as ANTLR [Parr, 2007] can be used to produce a structured artefact (such as an abstract syntax tree) from text. T2M tools are built atop parser generators and post-process the structured artefacts to produce a model that can be managed with a particular modelling framework. Xtext [The Eclipse Foundation, 2013b] and EMFtext [Heidenreich et al., 2009] are examples of T2M tools that, given a grammar and a target metamodel, will automatically generate a parser that transforms text to a model.

2.2.3.2 Model Validation

Model validation provides a mechanism to specify semantics and constraints associated with a model in the context of a particular domain. It allows users to perform model checking to detect errors and generate valid models, which is important in MDE to be able to test or check a model regarding particular concerns and questions [Henderson, 2003]. Similarly to model transformation, model validations are usually described with specific languages, developed to describe validation constraints, rather than with general programming languages (such as Java) [Kolovos, 2008].

The Object Constraint Language (OCL) [Object Management Group, 2012] is an OMG standard for defining constraints in modelling languages specified using UML and MOF. OCL provides an expression language with model querying and navigation facilities. The structural constraints are captured in the form of *invariants* attached to MOF meta-classes. [Kolovos et al., 2009] identify the shortcomings of the syntax and semantics of OCL *invariants* for capturing structural constraints and, accordingly, introduce the Epsilon Validation Language (EVL). EVL, in comparison with OCL, supports inter-model constraints, distinguishing between warnings and errors, dependant constraints, and repairing. The xlinkit toolkit [Nentwich et al., 2002] is also a lightweight approach which supports specifying inter-model constraints.

2.2.3.3 Model Compositions/Merging/Weaving

Model composition involves combining different models in a MDE process to produce a unified model [Bézivin et al., 2006]. It is mainly based on related work in aspect-oriented modelling [Cottenier et al., 2006] and database schema integration [Pottinger and Bernstein, 2003]. In comparison to model transformation and validation, model composition is not a well-understood concepts which is called with different terms in available frameworks. It is considered as *weaving* in Atlas Model Weaver (AMW) [Fabro et al., 2005], *merging* in Epsilon Merging Language (EML) [Kolovos et al., 2006c], and *gluing* in Glue Generator Tool (GGT) [Bouzitouna et al., 2005].

[Bézivin et al., 2006] provide a canonical set of definitions for model composition. Accordingly, in a model composition process the links between the models to be composed are captured in a *correspondence model*, an equivalent to the *weaving model* in AMW, *expression composition (EC)* in GGT, and *comparison rules* in EML. The correspondence model is created by a *match operation* which takes a set of models as input and searches for equivalents between their elements. Finally, the new output model is generated by a *compose operation*, which takes two models M_A, M_B and a correspondence model C_{AB} as input and combines their elements. [Bézivin et al., 2006] also distinguish between *merge* and *composition*, though they are used interchangeably. They indicate that merge is a special case of composition and define the *merge operation* which generates an output model including all the information from input models without duplicate information.

2.2.3.4 Other Model Management Operations

In addition to model transformation, validation, and composition, further examples of model management activities include model comparison (e.g. [Kolovos, 2009]), in which a set of matching elements between models is produced, model migration (e.g. [Rose et al., 2009; Gruschko et al., 2007]), which involves updating a model in response to changes to the metamodel, and testing model management operations (e.g. [Garcí-Domíguez et al., 2011]), in which model management operations are tested regarding a combination of input models, task and expected outputs.

Model tracing is also regarded as a model management task but, regarding the context of this thesis, is considered and discussed separately in the context of traceability in general and in the MDE domain in Section 2.1 and 2.2.4.

2.2.4 Modelling in the Large

[Bézivin et al., 2004] explicitly acknowledge the need to distinguish between modelling in the small and modelling in the large, similarly to the differences noticed between programming in the large and in the small [DeRemer and

Kron, 1975]. By modelling in the small, engineers are mostly interested in relationships and interactions between elements of models and metamodels, such as model transformation and model composition. Modelling in the large focuses on establishing and using relationships between models and metamodels (as single entities), ignoring the internal details of them.

[Bézivin et al., 2004] propose *megamodelling*, as the activity of modelling in the large, and the notion of a *MegaModel* to describe a registry of macroscopic entities (models, metamodels, operations, and services). In a later work, [Bézivin et al., 2006] introduce the Atlas Model Management Architecture (AMMA) addressing the fact that, however, it is important to be able to carry out both modelling in the small activities and megamodelling in a well-coordinated way. AMMA defines a lightweight platform providing integration between these complementary aspects. It consists of two main set of tools, one set of tools for modelling in the small (e.g ATL for model transformation and AMW for model weaving) and another set of tool for modelling in the large including Atlas MegaModel Management tool (AM3) particularly. AM3 defines the way the metadata of a given platform, including what is described in a *MegaModel*, is managed in a given scope.

[Salay et al., 2008, 2009] also focus on the relationships between models at a high-level of abstraction and introduce *macromodelling* to manage collections of related models. They propose a framework to formally define the relationship types between model types, such as *submodelOf* and *refinementOf*, within a new type of model, called a *macromodel*. Macromodels can be used to support development, comprehension, consistency management, and evolution of related models. A macromodel is a hierarchical model whose elements are used to denote models and model relationships, and can express integrity constraints on its elements.

Nevertheless, tranditionally, modelling in the large has been considered in the context of *multimodelling*, though it follows slightly different goals. Multimodelling focuses on creation and manipulation of interrelated models or *multimodels*. Existing approaches mainly provide mechanisms to specify how the contents of models are related in order to support tasks such as compositing different views (e.g. [Yie et al., 2009]) and consistency checking and inconsistency management (e.g. [Nuseibeh et al., 1994]). [Salay et al., 2008], in the context of their framework, consider a multimodel as a macromodel.

Traceability, in MDE, is similarly studied in two contexts: traceability in the small (classical traceability) and traceability in the large. Traceability in small mostly focuses on the trace information between model elements, while traceability in large is interested in traceability between models as a whole and handling such information.

[Barbero et al., 2007] presents global model management (modelling in the large) as a generic solution to support traceability in the large by putting traceability-related information into a *megamodel*. They suggest to use weaving models and megamodels as two complementary approaches

to deploy a generic and complete traceability solution. Weaving models are used to support traceability in small and store traceability information between model elements which are loosely coupled with the related models. Then, the megamodel contains relationships between models, for instance transformation models, weaving models, and UML models, and provides a global view of all the models involved in the traceability process.

Macromodelling, introduced in [Salay et al., 2008, 2009], is also used to join classical traceability and traceability in global model management. In their framework, the relationship types (trace link types between models) are defined by a traceability metamodel. The metamodel contains metamodel elements and associations of the metamodels it relates to as well as metamodel morphisms to map related metamodel elements to metamodel elements of the traceability metamodel. In a given development project, an initial macromodel is created expressing the required models and their relationships. The macromodel provides the comprehension of the collection of models in the project.

[Seibel et al., 2010] present a comprehensive traceability approach which combines classical traceability and global model management in form of dynamic hierarchical megamodels. They also highlight the importance of efficiency in maintenance of traceability and so provides efficient maintenance of traceability on top of megamodels. The approach considers both low-level traceability models, which contains trace links between model elements, and high-level traceability models (megamodels) specifying the links between models only. It defines hierarchical dependencies between high-level and low-level modelling artefacts and between traceability links at different levels to glue both traceability models into a combined traceability model.

The above approaches provide traceability in the large in the domain of MDE, with a similar approach to multi-domain traceability presented in this thesis. Although they could not be applied in the contxt of requirements traceability directly, they provide helpful suggestions and ideas to apply modelling in order to provide a multi-domain traceability solution, regarding existing challenges and difficulties.

2.3 Chapter Summary

This chapter presented the basic principles of traceability in software engineering and identified the core traceability activities. For each activity, the state-of-the art approaches were introduced and a discussion on their advantages and shortcomings was provided. Following this and based on available empirical surveys, limitations of current traceability practices were discussed and outlined. At the second part of the chapter, the principles of Model Driven Engineering were briefly introduced, regarding the scope of this research. In the next chapter, based on the conducted literature review, we

will give an overview of current traceability approaches and identify existing challenges related to research presented in this thesis.

3

Analysis of Traceability Approaches

In Chapter 2, a detailed review of traceability in software engineering was conducted. In this review, the different approaches to traceability were presented and open issues were identified. In this chapter, these findings are summarised (Section 3.1) and further analysed regarding the context and scope of this research (Section 3.2). Based on this analysis, Section 3.3 synthesises research challenges for traceability in multi-domain context, highlighting those to which this thesis contributes. Finally, requirements of a multi-domain traceability solution are defined in Section 3.4.

3.1 Overview of the Literature

This section presents an overview of the literature (a categorisation scheme) and provides a foundation for analysing existing traceability approaches regarding the context of this thesis. It helps to determine those approaches directly relevant to this work, in addition to identify research gaps in the domain of traceability.

Generally, the existing literature of traceability covers the following areas in the field of traceability. Each study and research could be located in more than one group depending on the scope and those aspects of traceability which it fully or partially (explicitly or implicitly) covers.

- Traceability fundamentals; defining traceability-related concepts such as terms, principles, and activities (Section 2.1.1)
- Traceability strategy; providing strategies to support traceability. For example, how to develop a complete traceability solution (e.g. [Espinoza and Garbajosa, 2008b]) or provide cost-effective traceability (e.g. [Egyed et al., 2005; Heindl and Biffl, 2005])

- What to Trace; specifying the syntax and semantics of the traceability information referred to as a traceability information model, metamodel, schema, classification, or reference model (Section 2.1.3.2)
- Creating Traces; providing techniques for manual and (semi-)automated trace identification and generation (Section 2.1.3.3)
- Maintaining Traces; introducing mechanisms for maintaining the integrity of the relationships while the entities continue to change and evolve (Section 2.1.3.4)
- Retrieving and Using Traces; discussing how to retrieve and use traces for example by specific query languages (Section 2.1.3.5)
- Visualisation; providing techniques for representation and visualisation of the traceability information (Section 2.1.3.5)
- Tooling; providing tooling support: commercial tools (e.g. DOORS, RTM, TOOR) or research tools (e.g. TraceM, ART, PRO-ART) (Section 2.1.4)
- Applications of traceability; discussing applications and benefits of traceability in different contexts (Section 2.1.2)
- Challenges; discussing existing problems and limitations and providing approaches and suggestions to overcome them (Section 2.1.5)

For the analysis, we focus on traceability in practice and, therefore, on approaches which propose a complete or partial traceability solution; introduce techniques or methods to support traceability activities, or provide tooling support. Therefore, we exclude studies in the first and the last two groups which outline and discuss conceptual features and issues of traceability. In the following, we discuss main parameters distinguishing existing traceability approaches from each other. Then, an overview of the approaches, based on these parameters, is provided and analysed.

Considering the traceability approaches described in Section 2.1, it is observed that available approaches are largely varied regarding the techniques or approach which they introduce to perform a particular traceability activity or task. In this context, available traceability approaches can be compared together in the context of the activity which they cover and basically categorised based on the general techniques that they use, those were introduced in Section 2.1.3.

On the other hand, traceability approaches differ with respect to the level in which they consider traceability. Some of the approaches introduce or develop strategies to design and implement traceability solutions and provide practical guidelines accordingly. These approaches define *abstract* solutions

describing a traceability process model or pattern regardless of the actual techniques or tools used to perform related activities (e.g. [Lago et al., 2009]). Some of the approaches define either a *concrete* traceability solution or introduce particular *low-level* techniques and tools to carry out activities.

A traceability approach could also focus on some aspects and provide supporting techniques while specifying how they are integrated with other techniques or methods in order to introduce a comprehensive solution. Most of the existing tracing approaches focus on one or two aspects of traceability, assuming there is already a traceability solution and their approach supports them. Usually they do not explicitly specify how their approach is integrated with other aspects of traceability in a comprehensive solution [Matragkas, 2011].

Additionally, traceability approaches differ with respect to the context in which they are proposed. This is because the context affects a solution regarding the artefacts which the solution considers and, hence, it can be applied to, the techniques introduced to carry out traceability activities efficiently in that context, and the tooling. Most of the approaches focus on specific context (e.g. software product lines, MDE) or particular scenarios (e.g. traceability between requirements and code, traceability between models), mainly in order to provide efficient and better traceability solutions.

An approach may also work with artefacts with specific formats (e.g. natural language, structured texts, models) or particular types of artefacts (e.g. requirements, class diagram, code). This way, the approach is restricted to the particular situation that it considers. Although an approach may introduce a technique based on artefact-specific characteristics, it could consider different types of artefacts in various formats by providing mechanisms to deal with them in an appropriate way. In this context, the types and formats of artefacts, covered by a traceability approach, are influential in applicability of an approach in practice.

According to the above discussion, we defined a set of parameters to examine each traceability approach. Considering the main elements of any traceability solution (TIM, traceability activities, and tooling), an approach is studied to determine which elements are covered and how (i.e. details of the proposed techniques or methods). Additionally, an approach is evaluated based on the types and formats of *artefacts* supported in the approach. An approach may be generic in terms of type or format.

In order to come up with more detailed and comparable results, we refined the above parameters into lower-level ones. For example, traceability activities include four activities and there are already recognised approaches in the context of each activity. We iteratively performed this refinement until we reach to the desired level of granularity. Though, the refinement could be continued to define more detailed parameters depending on the purpose of an evaluation.

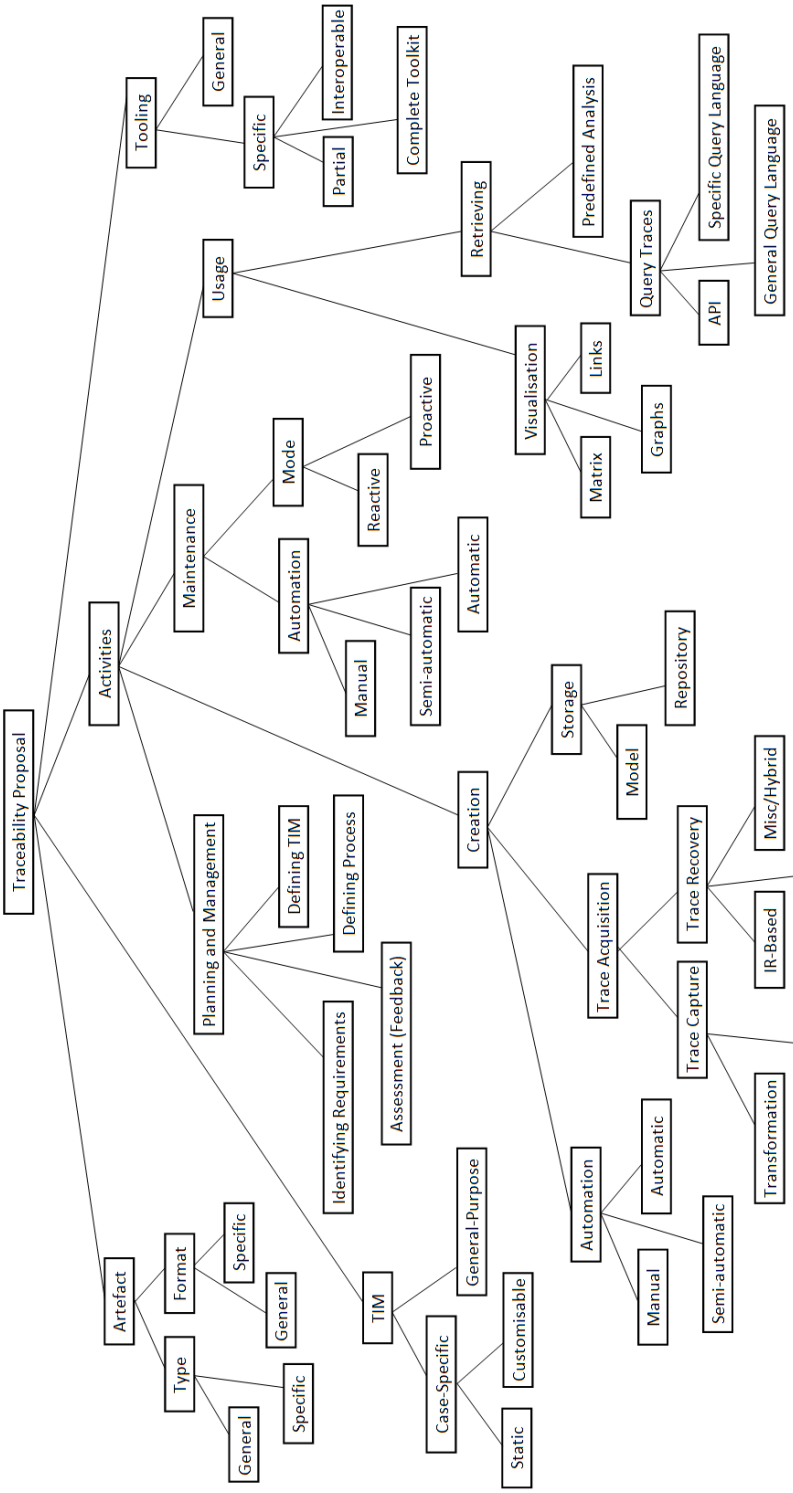


Figure 3.1: Categorisation parameters for traceability approaches

Appendix A presents a brief description of the categorisation parameters and Figure 3.1 shows them in a hierarchical diagram. In the first level, the main perspectives for evaluating an approach, including artefacts, TIM, activities, and tooling, are defined. The lowest-level nodes (leaves) show the categorisation parameters which are used to examine available resources to find out whether an approach covers or supports a parameter or not or it is unspecified. The middle nodes illustrate the hierarchical classification of the parameters.

A summary of the traceability approaches which were introduced and discussed in chapter 2, based on the above mentioned parameters, is provided in Appendix B. The summary provides a concise overview of existing approaches while highlights their main differences. It also supports the discussion at the beginning of this section. In particular, this overview shows that:

1. Most approaches are concerned with one or two specific aspects of traceability and do not consider integration within a comprehensive solution.

2. Most of the approaches are applied to specific types of artefacts or particular formats and do not mention how to deal with other types or formats.

3. A few approaches focus on the semantics of trace links and support case-specific TIMs. However, most of them focus on a specific context and provides a defined TIM for that context.

4. Few approaches cover planning and management activities. However, some of them cover a limited number of related tasks implicitly or partially.

5. Most of the approaches are concerned with trace creation and maintenance and try to improve these activities with more efficient techniques.

6. A few approaches consider retrieving traces specifically for traceability purposes.

There are also few classifications of traceability approaches in available surveys of traceability literature, which are almost similar to the above categorisation and show similar results. Amongst them, [Santiago et al., 2012] and [Matragkas, 2011] are more extensive and detailed in comparison to other ones (i.e. [Galvao and Goknil, 2007]). [Santiago et al., 2012] conduct a systematic literature review of traceability management in the context of MDE and identify the main issues covered by traceability proposals as a hierarchical feature diagram [Czarnecki, 2002]. Features determine the

parameters which distinguish traceability proposals from each other. Figure 3.2 depicts the first two levels of the identified features.

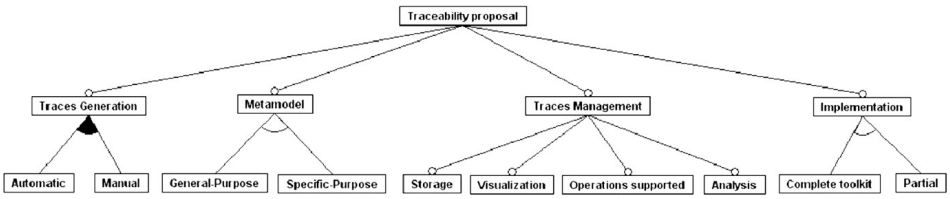


Figure 3.2: Feature diagram representing the main issues covered by traceability proposals proposed in [Santiago et al., 2012]

[Matragkas, 2011] observes parameters affecting the applicability of a traceability approach in different scenarios and highlights the factors that are more critical to a traceability approach including coverage of traceability activities, support for dependency and generative viewpoints, support for link semantics, automation, artefact support, traceability representation, and traceability maintenance. [Matragkas, 2011] also applies the *phenetics* or *numerical taxonomy* classification approach and, based on the similarity between approaches, identifies four main clusters of traceability approaches, named *Generative approaches*, *Representation-intensive approaches*, *Hyperlink-based Approaches*, and *Text-intensive approaches*.

3.2 Discussion

This section discusses the state of current traceability research regarding the context of this thesis, project-specific, multi-domain traceability solutions, and highlights main concerns and issues. Based on the provided overview in the previous section, the discussion focuses on those approaches which are similar to the research presented in this thesis in one of the following ways: providing traceability in the context of a complete solution, supporting project-specific traceability, end-to-end traceability (e.g. broader context, across boundaries), and traceability in MDE (particularly traceability in the large).

Effective traceability happens in the context of a traceability strategy which is implemented as a *traceability solution* for a particular project [Gotel et al., 2012b]. As mentioned in Section 2.1.1.1, a traceability solution comprises a TIM, a traceability process, and supporting tools. Therefore, a practical traceability approach should either cover all aspects of a traceability solution or clearly specify how it can be integrated within a comprehensive traceability solution (for example by providing extension points or integration mechanisms). However, as illustrated in the previous sec-

tion (3.1), most of the tracing approaches are concerned with one or two specific aspects of traceability. They are defined in a rather isolated way and do not acknowledge how their approach would be adapted in a solution, which makes their integration into a traceability solution a challenge and hinders their adoption in practice (also discussed in 2.1.5). Only few traceability approaches provide an almost complete solution: acknowledge the importance of being part of a comprehensive solution or cover four traceability activities (e.g. [Schwarz et al., 2010; Seibel et al., 2010; Matragkas, 2011; Asuncion and Taylor, 2012]).

An end-to-end traceability solution requires considering all types of artefacts in different formats. This is because during any development process, all kinds of artefacts, including natural language descriptions of requirements, spreadsheets, models, or executable source code, are inevitably created and used. Therefore, an end-to-end, effective traceability solution in a project requires to deal with all kinds of artefacts. It may be general and applicable to any artefacts or provide extension mechanisms to deal with all types and formats of artefacts, if it works efficiently with specific artefacts. However, as discussed in the previous section (3.1), most of the tracing approaches are limited to specific types and formats of artefacts and do not mention how they deal with other kinds of artefacts.

As mentioned before (Section 2.1.3), a cost-effective solution is defined and designed for a particular project situation. In such solutions, a TIM is defined based on project-specific traceability goals in the abstraction level of traceability (defines traceable artefacts and trace links between them) and used to collect and record required traceability information. As shown in the previous section (3.1), some of the existing approaches acknowledge this and define case-specific TIMs. However, there is a lack of practical guidance to define and use case-specific TIMs in general [Mäder et al., 2009b].

Additionally, in the existing project-specific traceability approaches, a TIM is often described independently from other models, though traceability models (instances of TIMs) are connected to other models in a project. None of them acknowledge that a project-specific TIM is inherently a redundant model which mostly provides a pervasive and coherent view of the *available* information in other models, regarding traceability perspective. They do not provide any explicit mechanisms to deal with this dependency and consequent redundancy.

On the other hand, as discussed in Section 1.1, traceability is a *multi-domain* concern. Traceability is usually mandated in projects extending across multiple engineering domains (e.g., software, mechanics and safety). Consequently, traceability may be required throughout the project lifecycle and so a practical (an effective) traceability solution needs to operate across the project's different domains; a TIM is a multi-domain construct as it refers to artefacts (documents, models, databases, project activities context) from different domains and may require relationships between multiple domains.

In this context, the above discussion is more challenging. It is essential to specify (explicitly consider) the relationships between a multi-domain TIM and other models, in different engineering domains, in order to be able to effectively deal with the dependency between multi-domain traceability models and other models.

Current research mostly focuses on traceability in a single context or, at most, similar contexts (e.g. requirements specification and coding) in which it is feasible and possible to integrate them together for example by using a particular family of tools or a common artefact. For example, [Asuncion, 2008] address the multi-faceted traceability and introduce an architecture-centric stakeholder-driven approach. The approach proposes to use architecture as the primary artefact to trace all other artefacts and capture the traceability links as a side effect of development tasks. It also allows stakeholders to define and maintain their trace relationships. Although the approach considers traceability as a multi-domain concern to some extent, it is dependent to the architecture model and focus on traces created as a side effect of development tasks. In the context of MDE, [Barbero et al., 2007; Salay et al., 2009; Seibel et al., 2010] consider traceability in the large with a similar perspective to the multi-domain traceability approach presented in this thesis. They focus on managing the relationships between models as whole and so introduce methods for multimodelling. Although their work could not be directly applied in traceability in general as they fundamentally address challenges in a MDE process, they provide helpful suggestions and idea to address existing challenges in multi-domain traceability using model-driven approach.

Existing traceability approaches demonstrate acceptable results and improvements in their target context, however they still fall short to support those tracing scenarios extending across tool boundaries [Asuncion and Taylor, 2012]. None of the existing approaches address situations in which engineering activities cannot be integrated or it is not feasible or cost-effective to do it. For example, safety engineering tasks are largely dependent on specialised tools and techniques which can not be easily integrated with general software development tools.

The multi-domain nature causes additional complexity and difficulties in supporting traceability which are outlined and discussed in the next section (3.3).

3.3 Challenges of Multi-Domain Traceability

The review of the literature and the example (provided in Section 1.1.1) suggest several challenges specific for multi-domain traceability.

As discussed in Section 3.2, traceability in systems engineering is inherently *multi-domain* and is required throughout the project lifecycle. Thus,

any traceability solution needs to operate across the project's different domains. Most of the following challenges originate because of the multi-domain nature of traceability. Depending on project requirements, a TIM possibly needs to support various development activities such as requirements or safety engineering and to capture *inter-domain* traceability information. An important concern in such situations is to keep the relationships among domains explicit while avoiding coupling and interference between domains. In the following, we discuss these challenges in more detail.

3.3.1 Domain-specific Traceability Information

For a systems engineering project, each domain (and hence domain-specific models) contains information which could explicitly or implicitly provide part of the information needed to generate a complete, project-wide traceability model, although it would be specific to a domain and insufficient to achieve traceability goals. In this context, it is recommended to (re-)use this existing information to generate the traceability model to avoid rework in order to control the additional cost of supporting traceability. However, (re-)using existing information results in redundancy and coupling between the traceability model and domain-specific models, which would lead to later inconsistency and synchronisation difficulties. This is because it is essential to keep the traceability model updated and synchronised with the other models [Kannenberg and Saiedian, 2009]; it should be updated whenever any of the models change to reflect the new states. Therefore, we need to use a proper way to build the traceability model which effectively manages this mandatory redundancy and coupling between these models while (re-)using them.

On the other hand, engineers usually prefer to work with familiar tools or techniques; the existing tools, models, and techniques which are specialised for a given domain. So, a traceability approach should allow users to use current tools and techniques while extracting required information from their models, artefacts, or documents.

3.3.2 Inter-domain Traceability Information

As mentioned above, we cannot just rely on available domain-specific information in a project: it mainly contains domain-specific concepts and normally does not cover any inter-domain relationships. The relationships between domains are usually defined informally or incompletely. For example, some times there are common objects in two domains which represent same concepts, though they are recorded separately in each domain. In these cases, relationships between domains are informally inferred from such objects; there is no formal or precise definition of which objects are equivalent or partially related. On the other hand, this way would result in inconsis-

tency between domains as it is not clearly specified which domains have to be updated in case of a change in one of them and how.

However, the relationships between domains are essential to accumulate the traceability information in order to provide a project-wide view of traceability. The engineer needs to have comprehensive knowledge about the domains and the relationships between them. Therefore, inter-domain traceability information should be defined and provided formally and precisely to support traceability needs. In this respect, finding and resolving redundancies and inconsistencies between domains are also of the main concerns in collecting the traceability-related data.

3.3.3 Diverse Information in Heterogeneous Format

Available information (which might be used to generate the project-wide traceability model) is provided in different formats, including documents (plain text or structured languages), models (e.g. UML class diagrams), databases, tools, or XML documents. Using a diverse set of information sources to extract traceability information, which is usually represented in heterogeneous formats, is one of the main problems in working with traceable artefacts and trace links [Mäder and Cleland-Huang, 2010]. To collect traceability-related information from existing information, we need a systematic approach to extract the required information and integrate them as the ultimate traceability information. In this context, integration of information from various domains which are expressed in different and heterogeneous formats is a major concern.

3.3.4 Separation of Concerns (SoC)

Engineers are interested in their own domain and usually prefer to work with familiar tools from a specific domain. Traceability models are created from traceability perspective and provide a different view of the project's information. Therefore, although the engineers might use the traceability information (represented in traceability models), it is not recommended to force all the stakeholders to work with the traceability model directly in addition to their domain models. In this context, a traceability solution should be transparent to the engineers while using domain-specific models.

3.4 Requirements of a Multi-Domain Traceability Solution

As stated in Chapter 1, the aim of this work is to provide a comprehensive multi-domain traceability solution, applying a modelling approach. The discussion which was taken place in Chapter 1 and Section 3.2 establishes a

foundation to identify the requirements of a practical and applicable multi-domain traceability. Although the focus of this work is on particular features of multi-domain solutions, there are a number of requirements which could benefit any traceability approach, though they are not directly related with multi-domain traceability. This section contributes requirements of a multi-domain traceability solution regarding the context of this research.

This thesis addresses the aforementioned challenges of multi-domain traceability in addition to general challenges of any effective traceability solution. On the other hand, as mentioned in Chapter 1, we propose to apply a modelling approach to build a traceability solution and use modelling (MDE) techniques to support the proposed traceability activities. Accordingly, we define our requirements in the context of three high-level requirements (goals): providing a traceability solution (G1), supporting multi-domain traceability (G2), and building a model-based traceability solution (G3). However, in some cases, a requirement could be considered as part of multiple groups.

We use the Goal Structuring Notation (GSN) [Kelly, 1998] to give an overview of the aforementioned requirements. The GSN is a graphical argumentation notation which is mainly used in the context of safety arguments [Kelly and Weaver, 2004]. Figure 3.3 shows the main symbols of the notation. A *goal* is a requirement or a constraint to be met by the system and it might be refined into sub-goals. *Strategies* (or arguments) specify how goals are addressed which are supported by direct reference to evidence, represented as *solutions*. A *context* defines the context in which a goal or a strategy is stated.

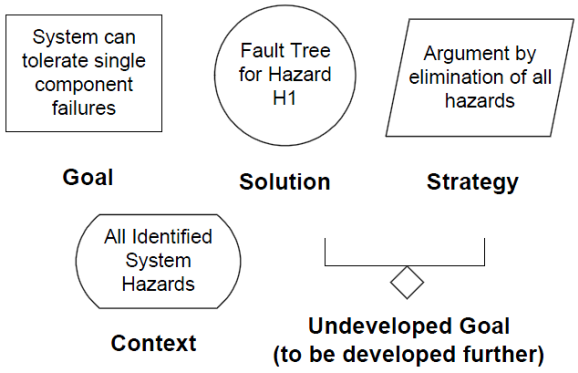


Figure 3.3: Main elements of the GSN [Kelly and Weaver, 2004]

In terms of the GSN, three high-level requirements (G1, G2, and G3) are considered as goals. Lower-level (underlying) requirements are defined as sub-goals (Rx.y or Rx.y.z). Descriptions for each requirement conceptually outline how a requirement would be satisfied and, so, they are represented as

strategies and titled with *St* and the correspondent numbering. The body of knowledge presented in the Chapter 2 is also considered as contexts of goals and strategies depending on the case. Later on (Chapter 7), the GSN diagram is extended with the proposed techniques (as solutions) which is used to generally demonstrate how the requirements and the research objectives are fulfilled. The GSN diagram representing the identified requirements is depicted in Figure 3.4 (a larger view of the diagram is provided in Figure C.1).

The principles of traceability (Section 2.1) and the challenges and difficulties of it in practice (Section 2.1.5) help to identify requirements of a traceability solution in general. Additionally, "The Grand Challenge of Traceability (v1.0)" [Gotel et al., 2012a], which presents a vision of traceability in 25 years, expresses main characteristics of a solution. It also provides a comprehensive reference of how each feature could be provided and the suggested strategies or mechanisms.

We define the requirements of a traceability solution as the followings.

- **R1.1:** Traceability is **requirements-driven** [Gotel et al., 2012a] and a cost-effective traceability solution is built based on stakeholder needs and the context of a specified project situation. Therefore, it is a fit-for-purpose solution.

  Accordingly, a fit-for-purpose traceability solution and, hence, its main elements, including TIM, activities, and tooling, need to be **customised (R1.1.1)** to accommodate stakeholder requirements and contexts for traceability. Doing so, any approach for building a customised solution has to provide

  - practices to determine stakeholder needs and identify *necessary information* which supports them (St1.1.1.1).
  - mechanisms to select and define *the [right] level of granularity* for traceability information to be captured and recorded. The granularity is usually dependent on the context of the project and the usage scenario (St1.1.1.2).
  - guidelines to develop a *customised process* of specified traceability activities specifying how they are integrated with project's activities (St1.1.1.3).
  - required *tools* which are sufficiently *configured* to support stakeholder requirements and their processes (St1.1.1.4).

  Additionally, a fit-for-purpose traceability solution needs to be **flexible (R1.1.2)** in order to keep the solution relevant over the time and address *evolving* stakeholder needs and contexts for traceability. This feature is concerned with maintaining the solution while it is used. Accordingly, It has to provide

  - mechanisms to change and adapt the TIM, the process and the tooling to traceability users' changing task contexts and needs (St1.1.2.1).
  - a *feedback loop* to reflect effectiveness of the solution with respect to the changing needs (St1.1.2.2).

- **R1.2:** A practical traceability approach should be **comprehensive**. Doing so, it should either support *all the basic traceability activities* described in Section 2.1.3 or provide required *extensibility mechanisms* for the activities it does not support (St1.2).

- **R1.3:** Automating traceability activities will reduce the cost and complexity of traceability and, so, increase its applicability [Egyed et al., 2005]. However, traceability activities can never be fully automated [Egyed and Grünbacher, 2005]. Accordingly, a traceability approach should **automate** as much as possible the different traceability activities (St1.3).

- **R1.4: Tooling support** for a traceability solution is essential to maximise the return on investment. As mentioned in Section 2.1.5, insufficient tool support has been identified as a factor which hinders the adoption of traceability solutions in practice [Kirova et al., 2008; Winkler and Pilgrim, 2010]. A practical solution has to provide required tools to support traceability (St1.4).

- **R1.5:** Current studies highlight the lack of interoperability between tools. In industry, there is a need for a comprehensive approach to traceability which considers **interoperability** between different tools such as requirements management tools, modelling tools, and integrated development environments (IDEs). A comprehensive traceability solution needs to address this issue and provide mechanisms to support interoperability (St1.5).

Considering the specific challenges of multi-domain traceability, the requirements for a multi-domain solutions are the followings.

- **R2.1** A multi-domain solution needs to consider existing **domain-specific information** (domain models) and the inherent coupling between these models and traceability models. It has to provide a systematic way to deal with this mandatory redundancy and coupling between these models while (re-)using domain-specific information (St2.1).

- **R2.2** The relationships between domains are essential to provide a project-wide view of traceability information. A multi-domain solution has to provide mechanisms to identify and capture **inter-domain traceability information** (St2.2).

- **R2.3** Available information, used to generate the project-wide traceability model, is provided in **different formats and contexts** (domains). To collect traceability-related information from existing information, a solution needs to provide a systematic approach to deal with this diversity in format and context (St2.3).

- **R2.4** Engineers are interested in their own domain and usually prefer to work with their familiar tools from a specific domain. A multi-domain traceability solution should be transparent to the engineers, as much as possible, and support **Separation of Concerns (SoC)** while (re-)using domain-specific models (St2.4).

Finally, the followings define the requirements for any modelling approach which have to be addressed in a model-based traceability solution, too.

- **R3.1** A model-based traceability solution works with models. So, the proposed approach has to be **applicable to models** and use model management operations to do proposed tasks to build a solution and carry out traceability activities (St3.1).

- **R3.2** The traceability-related information could be represented in different kinds of **non-model artefacts** (e.g. plain text, XML files, source codes). A model-based traceability solution needs to consider these non-model artefacts and specify how it deals with them, for example by providing mechanisms to create a model-based view of them (St3.2).

- **R3.3** Models conform to **different metamodels** and have different semantics. A model-based solution should not be dependent on specific metamodels and it should demonstrate how it works with different metamodels and use them in consistent way (St3.3).

- **R3.4** It is common to use **different notations** to express different concepts in different domain. So, a traceability approach in MDE should not be notation specific and it should be either generic or extensible in order to address the diversity of notations (St3.4).

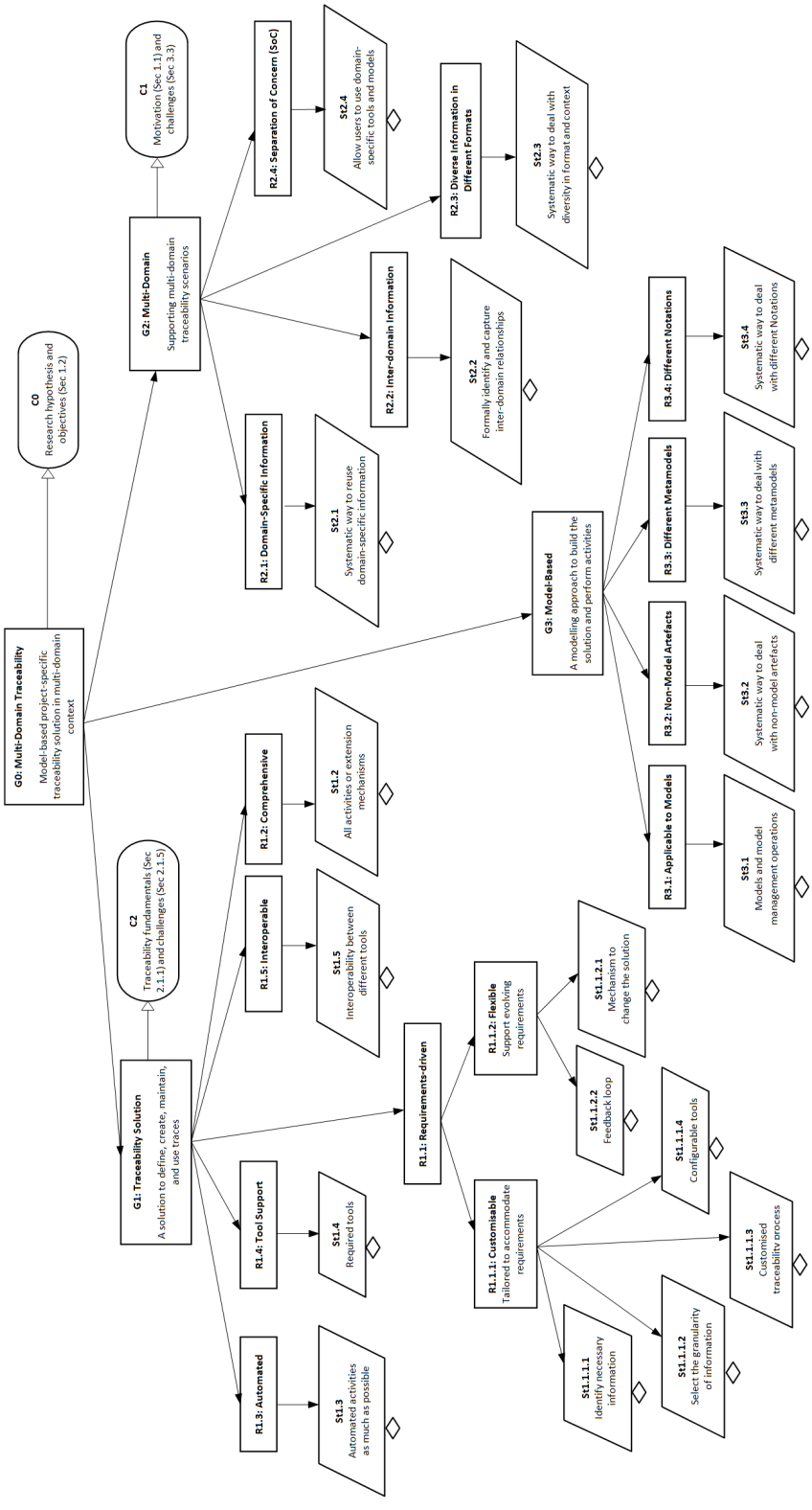


Figure 3.4: GSN diagram representing requirements of a multi-domain traceability solution (overall argument)

3.5 Chapter Summary

In the first part of this chapter, an overview of the traceability literature was provided and analysed regarding the context of this thesis. Based on this analysis the research challenges of this work were identified. Accordingly, in the last part of this chapter, the requirements of a multi-domain traceability solution were defined.

4

A Multi-Domain Traceability Solution

In the previous chapters, traceability approaches found in the literature were presented and analysed in the context of supporting multi-domain traceability. The analysis led to the identification of a set of requirements for a traceability solution in multi-domain projects. In this section the contributions of this thesis are presented. The contributions comprise a novel approach for building a multi-domain traceability solution. The approach is model-driven, and uses MDE techniques and tools to design and build a model-based traceability solution. The proposed approach aims to provide a less effort-intensive strategy for creating and maintaining trace relations, especially between different domains, to outweigh the additional cost of traceability.

Traceability solutions are designed, implemented, and used in the context of an overarching strategy to support traceability which comprises planning and management activities [Gotel et al., 2012b]. In this sense, the approach presented in this thesis demonstrates a strategy to provide traceability in multi-domain projects. It describes how stakeholder and requirements for traceability are determined, a suitable traceability solution is designed and implemented, and the requirement and the solution are kept relevant and effective during the life of a project. The approach also demonstrates how other traceability activities are carried out in the context of the proposed solution, namely how traces are created or captured, maintained, and used depending on traceability usage scenarios.

The approach uses modelling concepts for building and implementing the solution and performing the required and defined tasks. For example, the TIM is defined in the form of a domain-specific modelling language (DSML), all the input or output artefacts are represented as models and MDE principles and techniques are used to derive, extract, and identify traceability

information from different models, record the information in a traceability model (TM), and perform traceability analyses, based on traceability goals. In this context, the solution is implemented and supported within a *model-based* tool which is built atop an existing modelling framework concurrently with the solution and according to it. The tool works with models and allows users to perform traceability activities. We will discuss later, in the following sections, how we deal with non-model artefacts.

The approach covers two aspects of a traceability solution:

1. *Infrastructure*: steps which ultimately prepare or generate the infrastructure required to perform traceability activities, which comprises metamodels, utility models, and an environment to do operations. In fact, these steps support the required tooling to put the solution in operation.

2. *Operational*: tasks which capture traces, maintain them, use them with the use of the provided infrastructure.

A conceptual view of the proposed approach is illustrated in Figure 4.1. Usage scenarios are the input to the approach; they trigger the whole process. Through the next steps, the infrastructure is generated and, thereafter, it is used to carry out traceability activities as prescribed.

Initially, traceability goals are determined based on the usage scenarios: these are essential to identify the required information needed to answer project-specific questions or undertake traceability-enabled activities and tasks. Then, with a goal-oriented approach, the project activities, artefacts, and tools are observed and analysed, and available traceability-related concepts and relationships between them are identified. These concepts and relationships establish the basis for the traceability information model. These two activities are explained in more detail in Section 4.1.1 and Section 4.1.2.

Based on these activities, a TIM is defined which formally specifies the entity types to be recorded and the relationship types between them. The TIM acts as the reference to capture and collect traceability information in the project. From the implementation perspective, the TIM is developed as a domain-specific modelling language (DSML), the metamodel for the TM which represents captured traceability information (Section 4.1.3 and Section 4.1.3.1).

As discussed in Section 3.2, when observing a TIM, most of the traceability information is available in different domains. For example, a TIM could contain (define) *requirement* as a traceable element which can be found in the requirements engineering domain's models. In this context, we need to extract the required information and put it all in a project-wide TM. In this way, a TM is a *view* (similar to 'view' in databases) built on top of other models, which could be generated automatically by a query over these models. Doing so, we identify and specify the relationships between

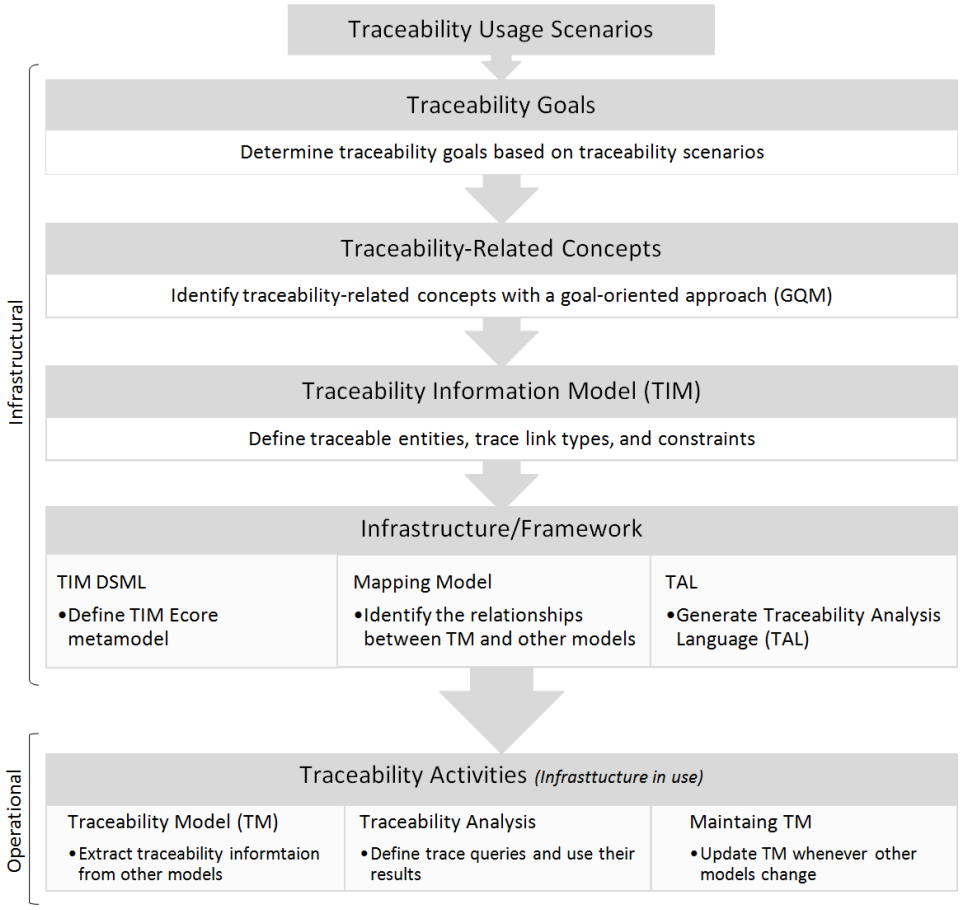


Figure 4.1: Conceptual view of the proposed approach

TM and other models. These relationships –called *mappings*– are described in a model (mapping model) and formally define how each concept in the TM (entities and relationships) is related to a concept in the underlying models. Using the mapping model, the traceability-related information is automatically extracted from existing models and instantiated in a *single* traceability model which conforms to the TIM.

Once the TM is generated, traces are used to support traceability usage scenarios. Traceability analyses are defined based on traceability goals, which are normally expressed in an abstract level, and performed on the TM. The analyses are described by a project-specific Traceability Analysis Language (TAL), which is mostly defined based on the TIM. The TAL lets us to query the TM and check specific constraints on it with respect to the defined traceability goals.

Finally, the approach considers the change and evolution in the traceabil-

ity goals and their effects on the other traceability-related artefacts (models). It provides specific practices to manage such changes, in addition to applying existing practices to maintain the integrity of the traceability model, regarding the changes in other models (general traceability maintenance).

Figure 4.2 introduces the main elements of the proposed traceability solution and how they are related to each other. It shows the location of the traceability solution from traceability perspective and illustrates how modelling principles and techniques are used to provide traceability.

As depicted in Figure 4.2, a solution is designed by engineers who focus on traceability and are only interested in (required) traceable elements and valid trace link types (traceability abstraction level). They determine traceability goals, define the TIM accordingly, and create the mapping model. Then, the modelling techniques and tools are used to generate and prepare the operational environment of the solution (the front-end). They generate the project-specific TAL, create traceability model, and execute traceability analyses. These structures and model operations represent the part of the infrastructure which is common in all solutions, called *Core* and shown in the yellow box. The GQM model, TIM, and mapping model constitute the project-specific part of the infrastructure as they are defined for a given project. Traceability users (end users) create the TM and use the generated TAL to define traceability analyses, based on traceability goals. The TM is used to perform traceability analyses and users use the analyses results for arbitrary purposes.

The following sections explain how elements of the proposed solution are defined and implemented, and required tasks are carried out. In Section 4.1, the steps to define and implement a project-specific TIM are introduced. Section 4.2 explains the prerequisite activities to generate a TM and Section 4.3 provides a systematic approach to capture traceability information in order to generate the project-wide TM. Section 4.4 discusses how the traceability model is maintained when the relevant models change and evolve along the time. In Section 4.5, the Traceability Analysis Language (TAL) is presented and it is demonstrated how it facilitates traceability analyses.

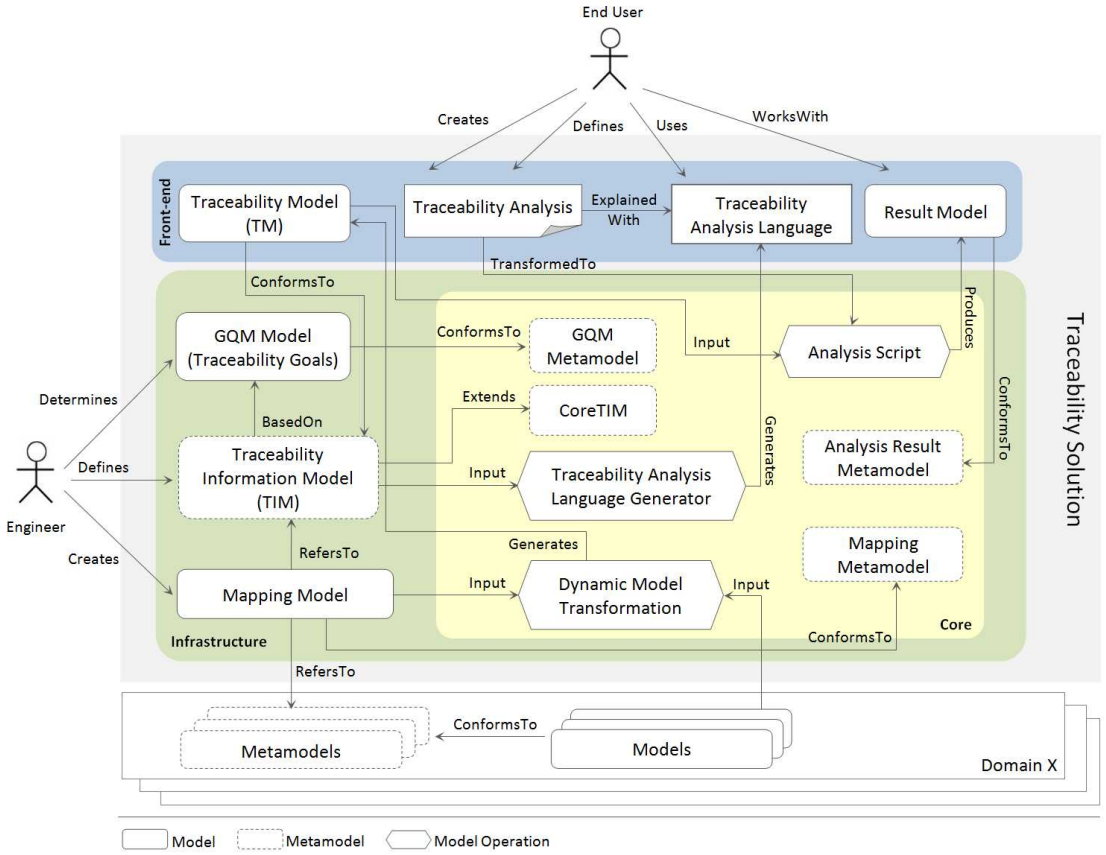


Figure 4.2: Main elements of the proposed traceability solution

4.1 Traceability Information Model

The core element of any traceability solution is a TIM (also called as traceability metamodel). A TIM determines the information required to support traceability goals, including which artefacts should be traced, the level of detail of the traces, and how traceability links should be classified regarding their usage, context or semantics [Ramesh and Jarke, 2001]. The first step to establish a traceability solution is to define a TIM, however traceability metamodels are still rarely defined and used [Mäder et al., 2009b]. Defining a TIM provides an easier and less effort-intensive approach for creating and updating traceability relations. It increases the benefits of traceability to compensate for its additional cost. As is widely accepted, a major drawback of traceability is the high effort associated with creating and updating relations [Mäder et al., 2009a].

However, researchers seem to agree on the value of a project-level definition of a TIM (e.g. [Aizenbud-Reshef et al., 2005; Egyed et al., 2007; Mäder and Cleland-Huang, 2010]). A project-specific TIM drives traceability activities such as the type of captured information (which is defined based on the project) and facilitates a consistent and ready-to-analyse set of traceability relations for a project. It also supports user-defined value considerations, a recommended feature for a traceability solution [Egyed et al., 2007].

The project-specific traceability approach allows engineers to focus on essential project-specific traceability information to find the critical information to support traceability. It also expresses the semantics of the traceability information as it is defined in the context of the project, especially in comparison to general-purpose traceability frameworks and tools [Mäder and Cleland-Huang, 2010]. However, this approach is not widely used in practice as current software development or requirements management tools provide little support to practitioners for building and using a customisable project-specific TIM [Mäder et al., 2009a].

In this thesis, the TIM is defined specifically for a project, based on its characteristics, constraints, and the traceability goals and described and implemented as a DSML which thereafter is used as the reference to collect traceability information. This way, users can use modelling techniques and tools to work with the TIM to generate traceability models and execute traceability analyses. It also helps to provide an integrated model-based tool which supports the proposed approach.

The project-specific TIM is defined in three steps that are performed iteratively until the target TIM has been developed. The steps, which are effectively based on a software engineering lifecycle (identifying requirements, conceptual modelling, and specification modelling), are defined as follows:

1. Determine traceability goals

2. Identify related project concepts

3. Define the TIM

At each iteration, the user considers and focuses on the current traceability requirements of the project. For doing so, the TIM is incrementally completed and extended whenever it is required which results in a lightweight TIM rather than a large TIM with many features which might be used in future. The iterative definition of the TIM also supports flexibility as it enables the user to change the TIM based on new traceability requirements. We now explain each step in more detail.

4.1.1 Step 1: Determine Traceability Goals

In this first step, traceability goals are determined based on the specific usage scenarios for the traceability information. Goals are defined in a lower level than scenarios and each scenario is usually refined into more than one goal. In general, goals represent the way in which engineers would support a scenario. For example, in a safety-critical project, a safety engineer might need to retrieve all requirements that mitigate identified hazards in order to construct a safety case; or, a developer might need to check whether the code they are modifying impacts specific quality constraints captured in the software requirements specification. Supporting each traceability goal needs specific information to be captured and recorded in the project. Traceability goals are defined based on project goals (e.g. satisfying functional requirements) or contextual constraints and requirements (e.g. certified as operationally safe).

For example, one of the main concerns in safety system development is providing valid and sufficient evidence for the argument which shows that a system is acceptably safe to operate in a particular context (e.g., civilian airspace) [Hawkins and Kelly, 2009]. Therefore, a contextual requirement for the example project, introduced in Section 1.1.1, is 'providing evidence for safety arguments'. The safety arguments in this project are: 'implementation is safe' and 'implementation is correct'. Accordingly, the traceability goals for this project are defined as follows; these will drive later steps to establish the traceability solution.

> "Providing evidence to show that the implementation is safe"

and

> "Providing evidence to show that the implementation is correct"

4.1.2 Step 2: Identify Related Project Concepts

Regarding the traceability goals, the project activities, roles and tasks are analysed to identify activities and artefacts related to or involved in supporting traceability goals. These concepts form the backbone of the traceability

information model. In this step, users could also determine if there is any missing concept (activity or artefact) in the project which has to be provided (done or produced) to satisfy traceability requirements.

The technique used to identify related concepts is based on the Goal-Question-Metric (GQM) approach proposed by [Basili and Caldiera, 1994]. GQM defines a systematic way to refine goals into questions and then into metrics to satisfy them. [Cleland-Huang et al., 2012] also suggest to use GQM in order to identify long-term strategic trace queries and the underlying data and traceability links needed to support them.

A GQM model is a hierarchical structure (e.g. Figure 4.3) starting from goals which are refined into several questions. Questions characterise the way in which a specific goal is achieved. Every question has a set of associated data in order to be answered. So, each question is then refined into associated data. The same data can be used to answer different questions.

In the context of traceability, an example GQM model is depicted in Figure 4.3. The model starts with traceability goals, at the top, which are refined into traceability-related questions, representing low-level trace queries and constraints. Each question is then refined into associated project concepts (e.g. artefacts, models, activity). In this way, the lowest level, in the model, cover all the traceability-related concepts in the project to satisfy identified traceability goals.

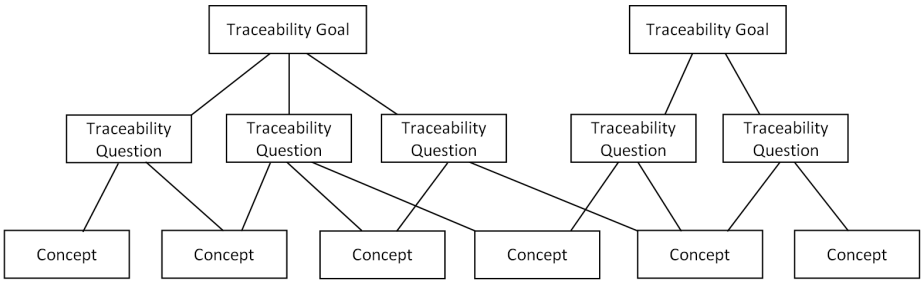


Figure 4.3: GQM model to identify traceability-related concepts

To illustrate the approach used to identify traceability-related concepts in more detail, an example from the IADDS project (see Section 1.1.1) is provided. In this project, to show that the *implementation is safe*, the safety engineers need to provide evidence indicating that safety requirements are satisfied, all potential hazards are identified, and identified hazards are mitigated and omitted. These expressions specify the questions indicating how the traceability goal is satisfied (in a lower level). So, the questions are defined as follows. The associated concepts for each question are specified too.

– Are the safety requirements satisfied by the implementation?

  To be able to answer this question, the following concepts and relationships between them have to be defined and captured in the project: safety story, development models, test cases, and test results.

– Have all the hazards been identified?

  To justify that all hazards have been identified correctly, we need to record and trace the relationships between user stories, hazard assessments, and identified hazards.

– Are the identified hazards omitted?

  This question needs the following concepts and the relationships between them: hazards, (derived) safety story, development model, code (implementation), and safety analysis model which shows that hazards do not happen or the possibility of a hazard happening is acceptable.

Figure 4.4 shows a part of the GQM model to identify traceability-related concepts in the IADDS project for the first traceability goal.

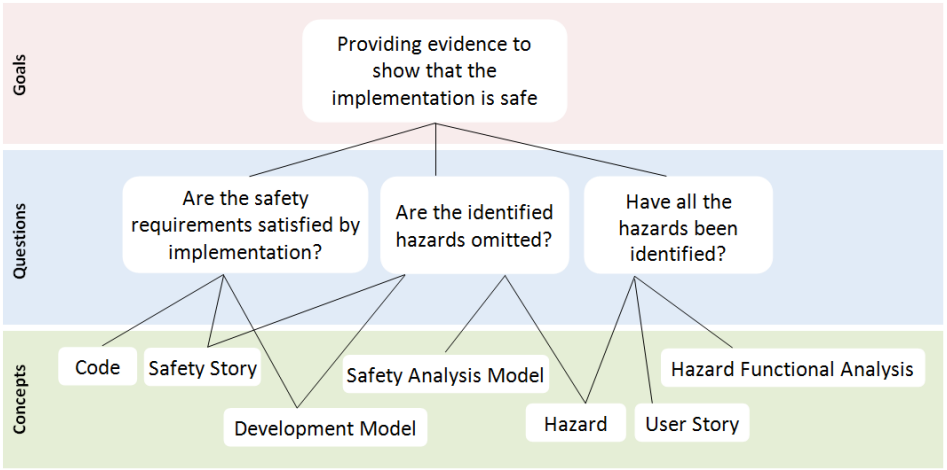


Figure 4.4: Part of the GQM model to identify traceability-related concepts to show 'implementation is safe' in IADDS project

4.1.2.1 GQM Metamodel

In the context of our modelling approach, we define a metamodel for the GQM approach which allows users to create GQM models within the provided tooling and have an integrated view of all the artefacts (models) involved in the solution. Additionally, this way, GQM models can be used in the traceability solution similarly to other models. For example, a GQM

model could provide justification to show why a TIM contains specific elements. So, it can be attached to a TIM.

The abstract syntax of the GQM metamodel has been defined using Ecore and is presented in Figure 4.5. In the following, the concepts of the GQM metamodel are explained.

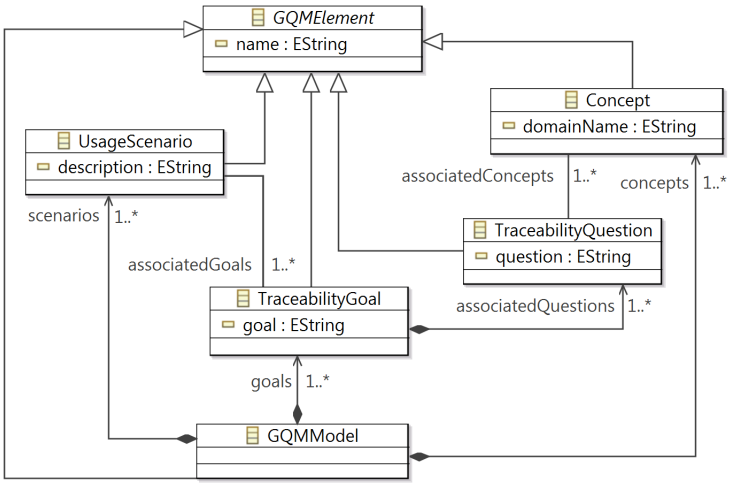


Figure 4.5: GQM Metamodel

GQMModel: Acts as the root of a GQM model. It defines a name and contains a number of UsageScenario, TraceabilityGoal, and Concept.

UsageScenario: Represents a usage scenario for the traceability. It defines a name and a description of the scenario. A scenario is associated with a number of TraceabilityGoals.

TraceabilityGoal: Represents a traceability goal. It defines a name and a goal. Each goal is associated with a number of TraceabilityQuestion into which the goal is refined.

TraceabilityQuestion: Represents a traceability question in the context of its upper-level goal. It defines a name and a question and contains a number of Concepts.

Concept: Represents a project concept required to answer a question. It defines a name and possibly the name of the domain that the concept belongs to.

Figure 4.6 shows the EMF model which represents the GQM model depicted in Figure 4.4.

4.1.3 Step 3: Define the TIM

Once the traceability goals have been determined (step 1) and the related concepts in the project have been identified (step 2), the TIM is defined.

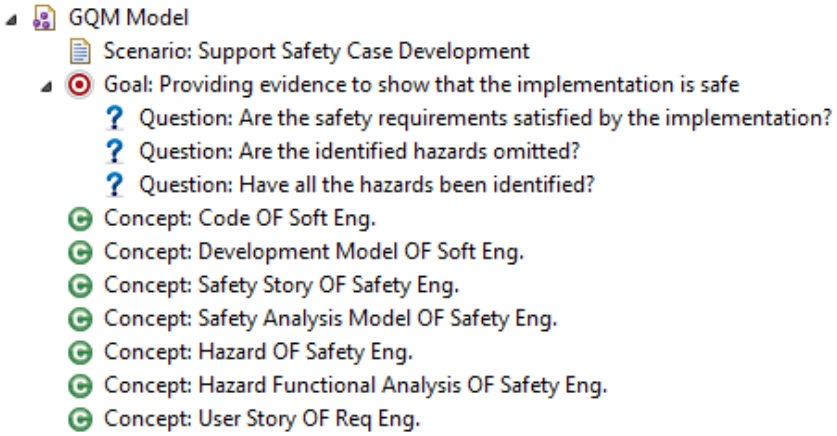


Figure 4.6: Part of the EMF model for the GQM model depicted in Fig. 4.4

According to [Espinoza et al., 2006], a TIM formally describes

- **Traceable Elements:** the objects that should be traced (captured and recorded)
- **Trace Links:** permitted traceability link types
- **Constraints:** validity requirements

A TIM is defined at the abstraction level of traceability: it defines traceable elements and valid trace link types between them, regardless of elements or associations not involved in traceability. Therefore, a TIM is a simple (but possibly a large) metamodel which specifies traceable concepts and relations between them through defining trace link types for the project. It also specifies validity constraints that can not be specified by the metamodel itself. These constraints can be specified using any constraint language compatible with the metamodelling infrastructure. As discussed before, traceability models (instances of TIM) are conceptually related to other models in the project, as they contain concepts in common. However, TIMs are described independently from other models.

Through several experiments in defining or studying project-specific or domain-specific TIMs, we have identified the recurring patterns in the structure and characteristics of traceability metamodels as discussed in [Taromirad et al., 2013]. Fundamentally, a TIM is defined from a traceability perspective and is totally independent from development artefacts. Therefore, it usually includes elements and relations which are already defined and recorded in other models in the project. Additionally, a traceable element defines the minimum information required to support traceability queries

(i.e. name and ID) rather than containing all the properties and information for the associated concept. Therefore, the TIM can be used as the reference to collect required traceability information in the form of a TM which conforms to the TIM.

These recurring patterns led us to develop a core traceability metamodel –called CoreTIM– with the aforementioned characteristics. The CoreTIM defines the basic structure of a TIM and defines the fundamental elements which have to be included in a traceability metamodel. The core metamodel is to be extended and completed by users to include project-specific concepts and semantics. In the following section, we will introduce the CoreTIM and how it is extended and used in a particular project.

4.1.3.1 CoreTIM

The abstract syntax of the CoreTIM has been defined using Ecore and is illustrated in Figure 4.7 (with an example extension). In this section, the concepts of the coreTIM are explained in more detail.

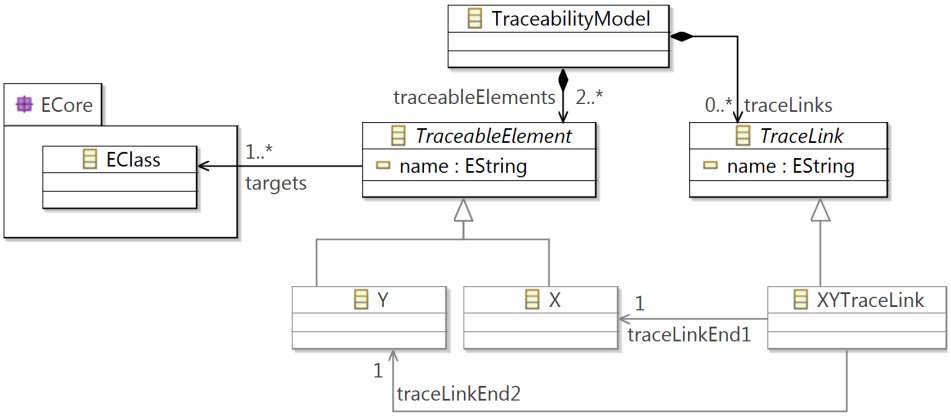


Figure 4.7: Abstract syntax of CoreTIM

TraceabilityModel: Acts as the root of a traceability model. It contains a number of TraceableElements and TraceLinks and could be renamed to the user-defined name in the context of the project.

TraceableElement: Represents a traceable object which may be the end of a traceability link. It defines a name and can be associated to a number of the original traceable concept, which are represented conceptually by this TraceableElement. The class is extended, in the context of the given project, to specify the project-specific traceable concepts (e.g. class X and Y in Fig. 4.7). Note that at this step, the type of the associated target model element is not specified. This information will be identified and defined

later on during identifying the relationships between TM and other models (discussed in Section 4.3).

TraceLink: Represents a traceability link between two elements. It defines name and is extended to define concrete valid trace link types in the project. Each subclass (e.g. class XYTraceLink in Fig. 4.7) has two references to TraceableElements, named *traceLinkEnd1* and *traceLinkEnd2*, to clearly specify the type of the trace link ends. *traceLinkEnd1* and *traceLinkEnd2* represent the source and the target of the trace link respectively.

The TIM is also accompanied by constraints. Structural constraints, such as cardinality of references, can be expressed within the metamodel. For more complicated and non-structural constraints, such as constraints defined in terms of attributes and their values, model validation languages (e.g. OCL [Object Management Group, 2012] and EVL [Kolovos et al., 2009]) can be used. A typical example of complex constraints, in the context of traceability metamodels, is to describe the following condition: Assume that A, B, and C are TraceableElements, R_1 is a TraceLink between A and B, R_2 is a TraceLink between B and C, and R_3 is a TraceLink between A and C. Also, there are a, b and c of type A, B, and C respectively. Then, the condition is that if there is a trace link of type R_1 between a and b, and a link of type of R_2 between b and c, there should be a trace link of type R_3 between a and c.

4.1.3.2 Project-Specific TIM

A project-specific traceability metamodel is defined by extending the CoreTIM. The extension defines the project-specific TraceableElements and TraceLinks and expresses the specific constraints regarding the project in hand. Note that, at this step, the type of the original traceable concepts (*targets* reference in a TraceableElement) is not specified in the metamodel. The type of these concepts in terms of target model elements is defined in later steps in which engineers identify the relationships between the TM and other models in the project. After these steps, the TIM is updated to provide an accurate and semantically rich project-specific traceability metamodel.

A project-specific TIM can also define other concepts in addition to the traceable elements and trace link types. An engineer may need to record when a trace link is created and who creates it, or she might want to attach a note to each trace link to provide arbitrary information about a link. Such information has to be specified in the traceability metamodel which is an extension to the CoreTIM, so called ExtendedCoreTIM. An ExtendedCoreTIM defines additional classes, attributes, and associations with respect to the specified needs and preferences. It is then enriched with project-specific TraceableElements and TraceLinks.

For example, consider the goal-oriented approach used to identify traceability-related concepts and define a TIM accordingly. The GQM model

can be considered as the context or rationale for why a TIM is defined in its current way. So, a project-specific TIM can define an entity called *Context/Rationale* which can be attached to any entities and shows why an entity is defined. Accordingly, a Context/Rationale for the whole TIM (root of the model: TraceabilityModel) or any element of it could have a reference to the GQM model of a project or an element of it. In Section 5.2.2, we illustrate an example of extending CoreTIM through attaching GQM models to a traceability metamodel.

The project-specific TIM for the example project was defined based on traceability goals. As discussed in Section 1.1.1, traceability is a mandatory requirement of safety standards (i.e. DO-178B RTCA and EUROCAE [1992]). In such systems, traceability from hazards to derived safety requirements and to implementation supports the safety argument to show that the system is operationally safe. In this context, we defined our traceability goal as 'providing evidence for the safety argument: the implementation is safe'. Based on the project development activities and artefacts, the project-specific traceability information model was defined accordingly. Fig. 4.8 depicts the TIM for the IADDS project based on the traceability goals.

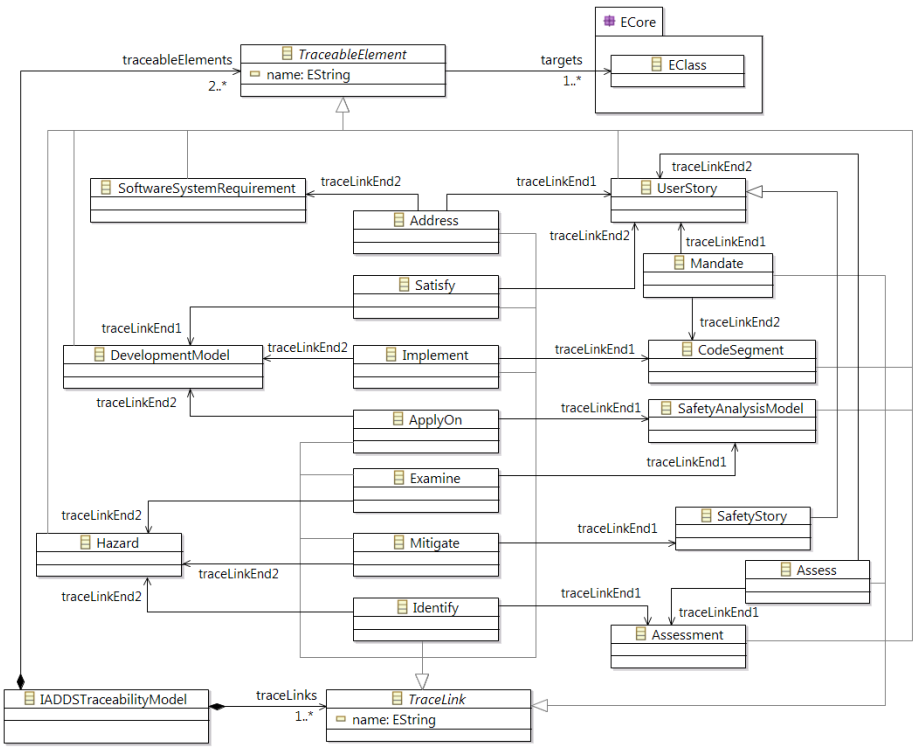


Figure 4.8: Part of the project-specific TIM for IADDS

As illustrated in Figure 4.8, the root of the TIM, 'IADDSTraceabilityModel', consists of concepts to be traced ('TraceableElements'), such as 'UserStory', 'DevelopmentModel', and 'Hazard', and trace link types ('TraceLinks') which are defined between specific traceable elements, such as 'Satisfy' and 'Mitigate'.

In the safety domain, hazards are categorised into two groups based on the result of safety analyses: mitigated and unmitigated. In the case of mitigated hazards, at least one derived safety requirement is defined to prevent the hazard from occuring. In contrast, when a hazard cannot be prevented it is classified as unmitigated, hence there is no derived safety requirement for that hazard. Considering this, in the TIM for IADDS project, the 'Hazard' has an additional attribute called *mitigated*, indicating if the hazard is mitigated or not. Accordingly, if a hazard is classified as mitigated there should be at least one element of type 'SafetyStory' and a link of type 'Mitigate' between the hazard and the safety story. This constraint is an example of complex inter-model constraints, which can be described with model validation languages (EVL in our example project). Listing 4.1 shows the EVL code for this constraint. Other validation languages that allow evaluation of expressions over heterogeneous models (e.g. XLinkit [Nentwich et al., 2002]) could also be used.

```
1  context IADDSTIM!Hazard {
2    guard : self.mitigated
3    constraint LinkedToSafetyStory {
4      check : Mitigate.all().exists(
5          m | m.traceLinkEnd2 = self
6          and
7          m.traceLinkEnd1.isDefined());
8      message : "Hazard `" + self + "' should be linked to at
           least one SafetyStory";
9    }
10 }
```

Listing 4.1: Example EVL constraint in the TIM in IADDS

As mentioned before, if we consider the TIM, it is observed that the traceable elements contain requirements engineering, software development, and safety engineering artefacts, which most of them are recorded and available in the related context (domain). The TIM provides a comprehensive view of all the required traceability information by including them in one integrated metamodel.

Accordingly, we suggest extracting traceability information from available information (other models) and automatically create a traceability model which conforms to the TIM. Before explaining the steps to extract and create the traceability model, the available traceability information in different domains will be studied in more detail; this appears in the next section (4.2). Then, the proposed approach to generate the TM is introduced

in Section 4.3.

4.2 Traceability-Related Information

Once the TIM has been defined, traceability information can be collected and recorded in a traceability model. But, as discussed before, considerable amount of required information to support traceability is provided and available in models in different context or domains. Although these models might be defined for other purposes (not explicitly for traceability), they could provide traceability-related data. This section focuses on such information scattered in different domains, generally called as *traceability-related information*, and discusses how this information can be used to generate a project-wide traceability model. In this context, related traceability information is investigated to prepare the prerequisites to generate and populate the final traceability model.

As mentioned before, the required information, which supports traceability, could be represented in various formats (e.g. plain text, XML files) with different underlying structures and metamodels. To support analyses and an overall MDE approach to developing tools, we need to provide a model-based view of the non-model information (wherever it is possible and reasonable) as a prerequisite of generating a TM; this can be challenging. However, we focus on those models that are available and that information which can be automatically transformed to models: there are several types of model in different domains (e.g., requirements models, safety analysis models) that provide substantial traceability information. Our approach does not restrict the types of model that can be considered in a traceability solution, but in the illustrative example that follows we consider a specific set of models.

4.2.1 Investigate Available Information

Based on the TIM, available information (models) are investigated to find out how much of the required information is provided and available, in which ways, and how it can be used.

In general, there are two types of information:

1. **Domain-specific information:** information which is specific or limited to one domain

2. **Multi-domain information:** information which covers two or more domains

For example, considering the general development process of the example project, the following information was available and provided in the involved engineering domains:

1. Requirements Engineering: Software System Requirements (plain text), Properties and Constraints (plain text), User Stories (tabular format)
2. Safety Engineering: Hazards (tabular format), Derived Safety Requirements (tabular format), Safety Analysis Models, and the relationship between Hazards and Derived Safety Requirements
3. Software Engineering (Development): Architecture Model, Class Diagrams, Test Cases, Code Segments

In addition to the above domain-specific information, the following multi-domain information is defined and provided:

1. Relationship between Hazard and User Story (in tabular format)
2. Relationship between Safety Story and User Story (in tabular format)

As the result of this investigation, the available information including models and trace link types either within or between domains which supports traceability is identified. Additionally, the investigation will identify the missing information which is essential to generate a complete project-wide traceability model such as models or a formal relationship between two concepts. This requires us to provide currently not provided data or concepts. The missing information is divided into two parts: that which is limited to a single domain; and that which relates to multiple domains, such as the relationship between concepts in different domains. Each of them has to be dealt with differently. We now elaborate on this in more detail.

4.2.2 Domain-Specific Information

As mentioned before, domain-specific information represents concepts and their relations within the context of a domain. In this respect, the information available as a model can be directly used to extract traceability information. On the other hand, to define and collect missing domain-specific traceability-related information –not available either in the form of a model or in any other format– one of the following alternatives could be applied.

1. **Extend existing metamodels.** In some cases, it is reasonable and possible to extend existing models to include missing information. For example, the missing info is a number of simple attributes. In these cases, the correspondence metamodels are extended by defining or adding new objects, attributes, and relationships (as associations or objects), and describing required constraints. Consequently, the target models have to be updated or regenerated according to the new metamodel. Nevertheless, depending on the type and size of changes in metamodels, particular practice need to be applied to update models with valid additional or new data.

2. **Define new metamodels.** In some cases, it is not possible to work with existing metamodels: 1. there is no artefact (in any format) which covers even partially related information to the missing one, 2. there are non-model artefacts covering missing information, for example within other tools (e.g. requirements management tools such as DOORS) or arbitrary structured documents (e.g. spreadsheets), and 3. the engineer cannot alter the metamodels with the new concepts (e.g. due to limited access, regulations). In these cases, new metamodels have to be defined (from scratch) and instantiated as new (domain) models which include the missing information.

   In the first case, depending on the type of the information and also possible and available ways for collecting data, required information could be captured by any heuristic method. The new models thereafter can be used and manipulated similar to other artefacts in the project.

   In the other two cases, different model transformation techniques (M2M, M2T, T2M) can be applied to transform the non-model artefacts or models into the new models which also include the complementary new data (missing information). For example, a spreadsheet, which is a common format for preparing documents in a project, can be transformed into desired models with user-defined T2M transformations or available tools specifically developed for this purpose (e.g. [Francis et al., 2013]). Most of the existing tools provide the feature to export their data into spreadsheets. For example, DOORS, as the most widely used tool in industry to manage requirements and record traces between them, allows users to export the description of requirements and their relations to different formats including spreadsheet. The generated spreadsheets can be used to generate or populate the new models representing the missing information.

   In these cases, maintaining the consistency between original artefacts or models with the new models is a challenge which is extensively studied in the context of MDE under co-evolution subject (e.g. [Rose, 2011]). Automatic model transformations partially address this issue, as they can be re-executed whenever any of the original artefacts change. This is possible when it does not invalidate the extra data previously and independently inserted into new models.

4.2.2.1 Example

Regarding the available information and their format (mostly provided in textual format in tables) in the example project, we define a new metamodel for each domain which defines the information in that domain. Existing information (tables) is then transformed into new domain models. Figure 4.9 and Figure 4.10 depict the main artefacts in safety and requirements engi-

neering domain as the metamodel for these domain models. The metamodels also show the association relationships between concepts in each domain which are later on translated into trace link types in the TIM.

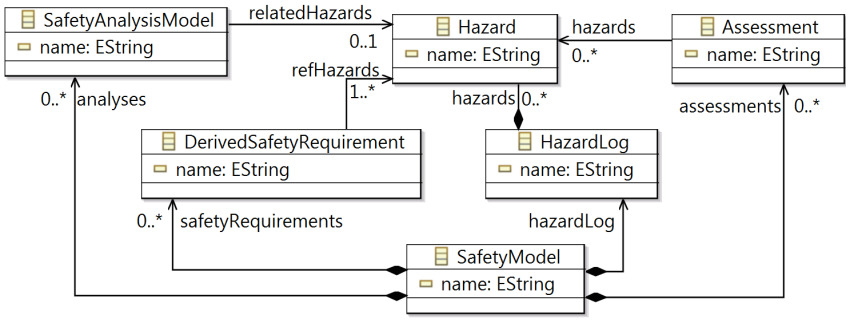


Figure 4.9: The metamodel for the safety domain

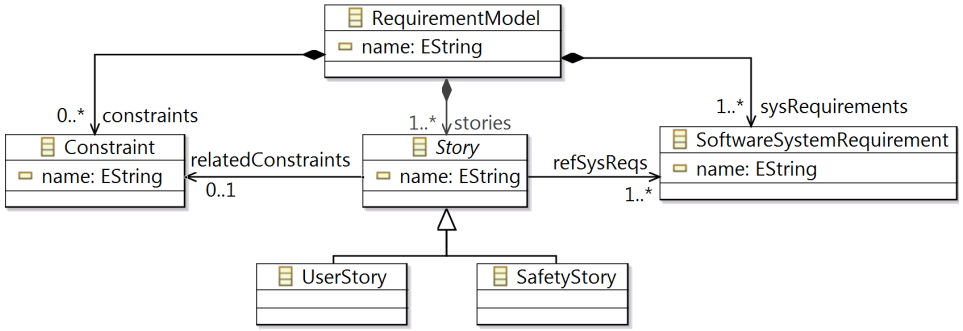


Figure 4.10: The metamodel for the requirements engineering domain

4.2.3 Multi-Domain Information

After investigating the required information in each domain and providing the information as models, the multi-domain information is considered. Multi-domain information encompasses concepts from different domains. It mainly represents the relationships existing between two domains. One of the main unavailable traceability-related data is a precise and formal definition of the relationships between domains. The available traceability-related information (mainly trace links) is usually limited to one domain. In this respect, we focus on the relations between domains.

To represent the relationships between domains, so called inter-domain trace links, we propose building a traceability model –so called partialTM–

between pairs of domains to keep record of inter-domain trace links. We define a traceability information model for each partialTM. Each traceability information model –called partialTIM– formally or explicitly defines the inter-domain trace link types. Basically, a partialTIM is defined by extending the CoreTIM; each partialTIM can be considered as a slice of the main TIM only containing those elements that are directly related to each other (with a trace link). Figure 4.11 conceptually shows the relationship between partialTIMs and the main TIM.

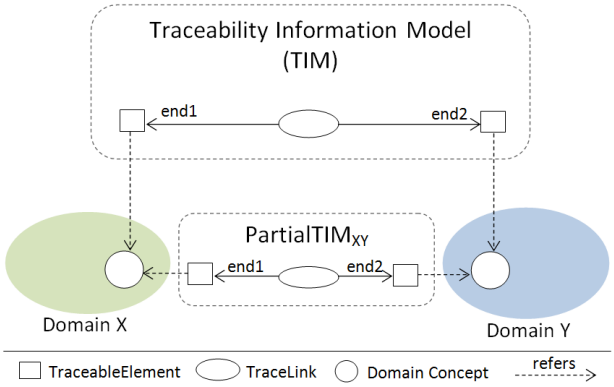


Figure 4.11: How a TIM and partialTIMs are related

A partialTIM clearly specifies the required trace link types and the type of trace link ends for each trace link type. There are two kinds of inter-domain traces:

1. *Equivalence trace:* This type of trace shows equivalent concepts in two domains. In this case, a concept from one domain is redefined and reused in another domain (possibly with new name), though they represent an identical concept.

2. *Relationship trace:* These traces are used to show relationships between elements from two different domains that are related to each other, in some way, but they are not equivalent.

Although, in this thesis, partialTIMs are defined manually, they can be defined semi-automatically by using different type of model management operations, for example, by comparing traceable entities, defined in a TIM and linked directly with a trace link type, with available entities defined in different domains. Through such comparison, it is possible to identify potential relationships between two domains and, hence, a partialTIM. A partialTIM can also be defined based on the available onthologies, which provide description of concepts in different domains, and available knowledge

of how concepts of anthologies are related. These would specify the potential partialTIMs.

Inter-domain trace links can be identified and captured manually or semi-automatically (i.e. through model comparison). In the manual case, the user manually defines the traces between domains based on the correspondent partialTIM, while in the latter form traces are identified with model comparison. In this case, the users might need to check the identified traces to select valid traces to be recorded.

4.2.3.1 Example

In the IADDS project, we define the required traceability metamodels for the inter-domain trace link types. Figure 4.12 shows the partialTIM between the requirements engineering and safety engineering domains which defines a *relationship* trace link type, named 'UserStoryToAssessmentTL', between 'UserStory' and 'Assessment' in addition to an *equivalence* trace link type, named 'SafetyStoryToDerivedSafetyReqEL', between 'SafetyStory' and 'DerivedSafetyRequirement'. In this example, the trace links are captured and recorded manually.

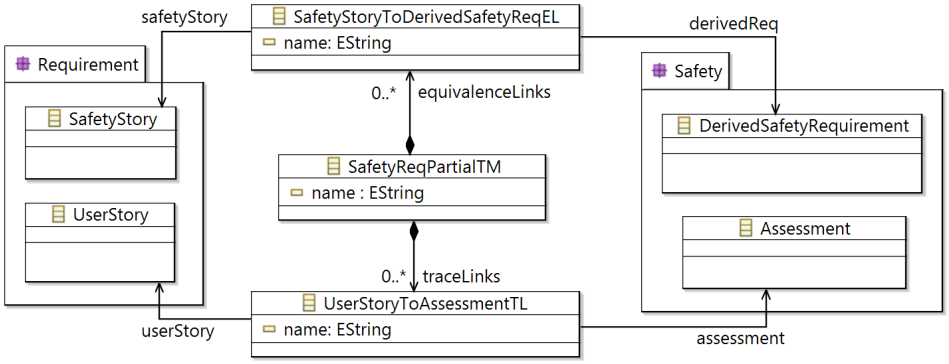


Figure 4.12: The partialTIM between requirement and safety engineering domain (ReqSafety-partialTIM)

4.3 Traceability Information

Once the domain-specific and inter-domain models (which provide required traceability-related information), are available and ready to use, it is time to extract the information from them and generate a *single* project-wide traceability model; a model which is built on top of the other models, generated automatically by model operations and does not contain any information that cannot be regenerated automatically. The TM, as described earlier, is

like a *view*; theoretically it should be generated on-demand, and not updated directly.

A single TM provides a coherent view of the traceability information spread in different models. As [Mäder and Cleland-Huang, 2010] state, using a diverse set of traceability information sources (usually represented in heterogeneous formats) is one of the main problems in working with trace links. A TM also unifies the way in which the traceability information can be used to perform traceability analyses. For example, it lets us define a *Traceability Analysis Language* to describe traceability analyses, based on traceability goals, regardless of the diversity of the underlying models.

Having prepared and completed the domain-specific and inter-domain models, which are called *source models* in the rest of this document, the project-wide TM is generated through the following two steps. Figure 4.13 illustrates the conceptual overview of how a TM is generated.

1. **Locate source elements.** In this step, the relationships between the TM and other models are identified. Each relationship specifies that each type in the TM maps to what type from source models. It also defines precisely how two elements are related. These relationships are defined in a *mapping model*.

2. **Generate the TM.** In this step, using model operations, the required information is extracted from source models based on the mapping model, and the traceability model is generated and populated with the collected information.

4.3.1 Locating Source Elements

In this step, the model elements in the TM are mapped into model elements in source models. The *mapping* formally specifies how each concept (traceable element or trace link type) defined in the TIM relates to model elements in the other models. It is used to collect information from other models in order to automatically generate the project-wide TM. It could also be used to interpret the result of traceability analyses in terms of the source models in different domains. The *mapping* is expressed in a model which conforms to a mapping metamodel. The mapping model is similar to a Correspondence Model (CM) or Weaving Model (WM) in model composition [Bézivin et al., 2006]. A CM or a WM is a model that explicitly describes the relationships between elements of different models, but is constructed specifically for model comparison or merging processes.

We captured the specific requirements for the mapping model regarding the context of this work and how it would be used in the proposed traceability solution. Accordingly, the metamodel

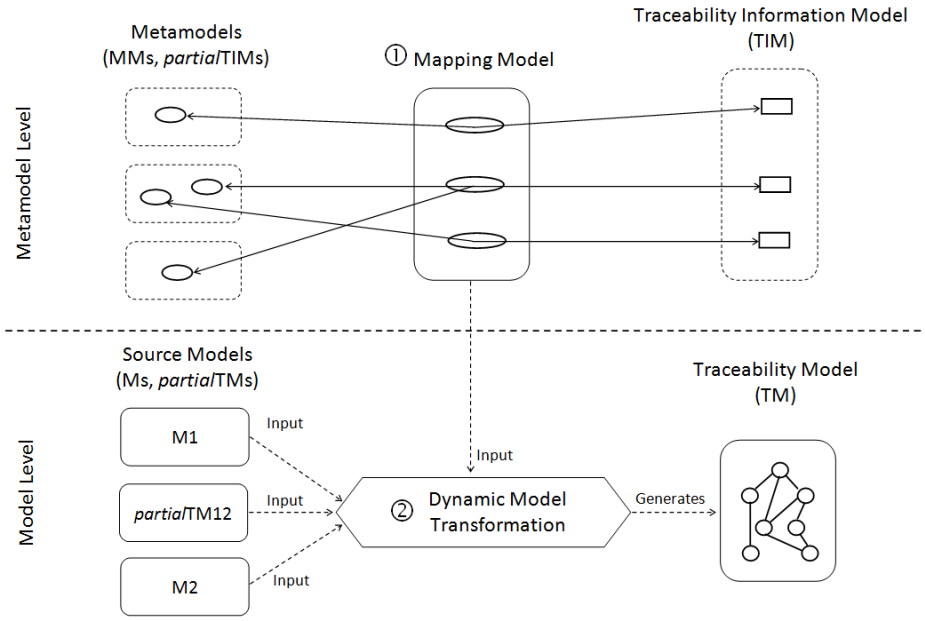


Figure 4.13: Generating the traceability model

- is defined at the metamodel level and refers to (Ecore) metamodels (R1)
- is used either to generate the TM and translate the result of trace queries in terms of the source models (R2)
- defines what element maps to what element (R3)
- explains precisely how two elements are related (R4)

Additionally, we determined that in order to have complete information to generate the TM, the mapping model should capture two types of relationships between models:

1. Relationships between a TM and other models, called *Mapping Entry* in the model; they define how each element in TM is related to elements in source models. For example, regarding the TIM for the IADDS project (Figure 4.8) and the partialTIM between requirement and safety domains (Figure 4.12), the 'Assess' trace link type, in the TIM, maps to the 'UserStoryToAssessmentTL' between 'UserStory' and 'Assessment', in the partialTIM between requirements and safety engineering domains.

2. Relationships between models except TM, called *Equivalence Entry* in the model; they define equivalent elements which are those elements that represent a common concept in different domains. For example, in the IADDS project, 'SafetyStory' in the requirements engineering domain (Figure 4.10) is equivalent to 'DerivedSafetyRequirement' in safety domain model (Figure 4.9).

Figure 4.14 shows the examples described above for each type of relationships. A mapping relationship (highlighted in red) is defined between the TIM and the ReqSafety-partialTIM (the partialTIM between the requirements engineering and safety engineering domains). Two equivalent elements in requirements engineering and safety engineering domains are shown and highlighted in blue.

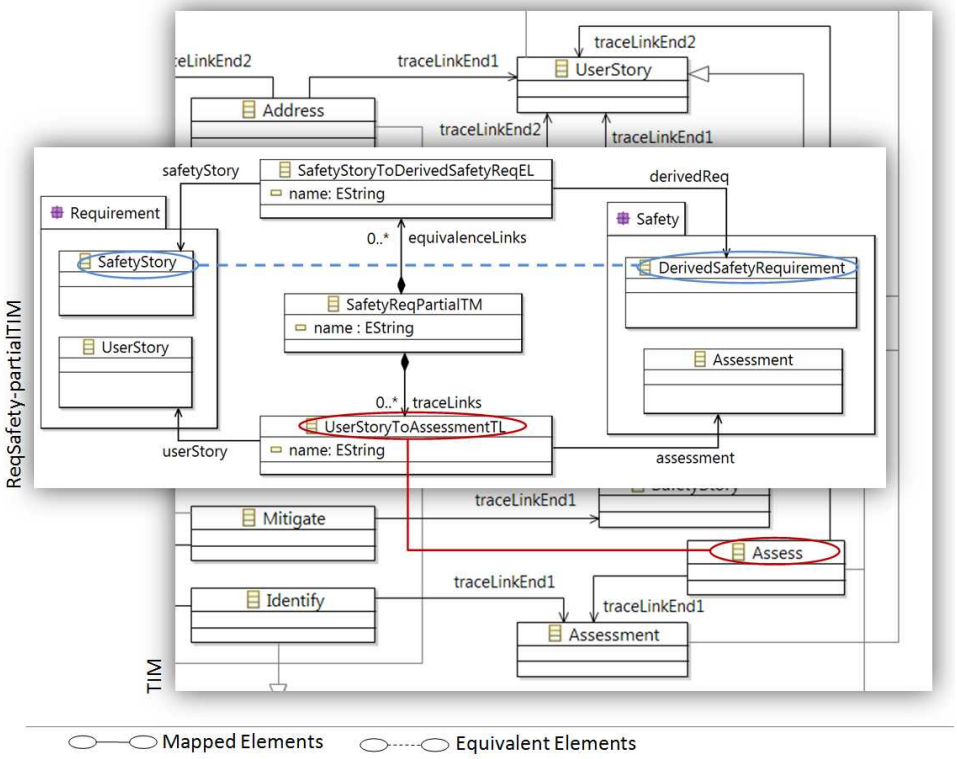


Figure 4.14: Graphical view of example relationships between the TIM and the ReqSafety-partialTIM

According to the above mentioned requirements and properties, the mapping metamodel has been defined using Ecore, as depicted in Figure 4.15.

4.3.1.1 The Mapping Metamodel

In this section the concepts of the mapping metamodel are explained in detail.

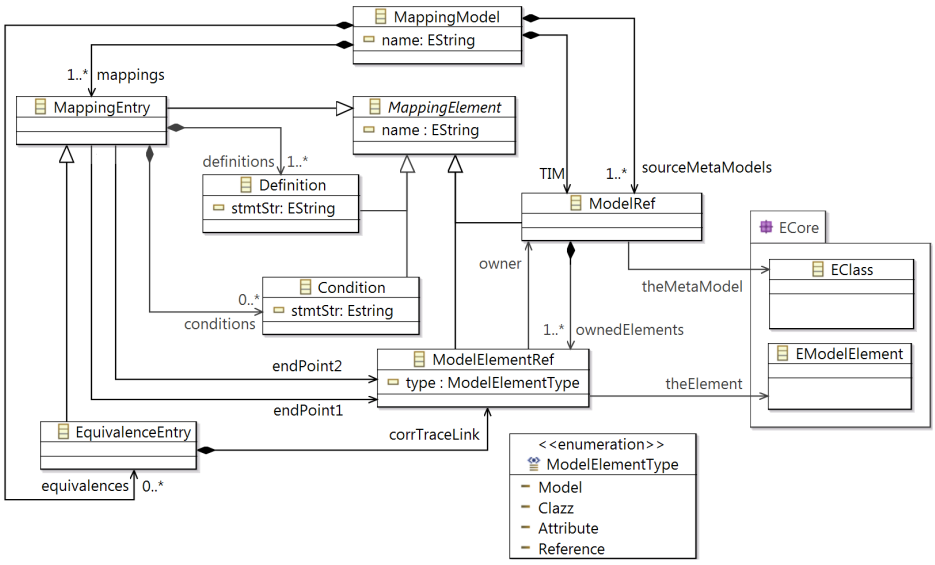


Figure 4.15: Mapping Metamodel

MappingModel: Acts as the root of the mapping model. It defines a name and contains at least two ModeRefs which represent the TIM and other involved metamodels (Ecore metamodels). It also contains a number of MappingEntries and EnquvalenceEntries.

ModelRef: Represents a metamodel of the model used to generate the final TM. It defines name and contains a number of ModelElementRefs defining the elements from this metamodel which are used in the mapping model.

ModelElementRef: Represents an element of a metamodel which is used in the mapping model. It can be a model, a class, an attribute, or a reference.

MappingEntry: Represents a mapping between a model element in TIM with a model element in another metamodel. Each mapping defines two ModelElementRefs (as end points of the mapping) and a number of Definitions and Conditions.

EquivalenceEntry: Defines an equivalence relationship between two model elements from two different metamodels (other than the TIM). An EquivalenceEntry is a kind of MappingEntry with an additional reference (*corrTraceLink*) to the related *partial*TIM (which defines the equivalence traces between two related domains). This reference is required to find equivalent objects when generating project-wide TM.

Definition: A statement indicates how the two end points of a mapping are exactly related to each other (e.g. endPoint1.name=endPoint2.ID)

Condition: A statement specify the condition in which an entry is valid and used.

A ModelRef refers to metamodels and a ModelElementRef represents a type defined in the metamodels (requirement R1). The MappingEntry and EquivalenceEntry address the requirement R3 and the two relationship types necessary to generate the TM. With Definition and Condition users express how two elements are related (requirement R4) which is used to automatically populate the TM with correct and valid values for the model elements.

A mapping model, which conforms to the above metamodel, is a high-level specification of the model transformation rules which are required to extract information from the source models, create the TM, and populate it with this information. The transformation is obtained by translating the mapping model into low-level, executable model operations. Section 5.2.4 explains how the mapping model is used to generate the TM.

4.3.1.2 Example

Figure 4.16 shows an example of mapping and equivalence entries from the case study which were explained above. As shown in the figure, the mapping model works with four metamodels: the IADDS TIM, the metamodel for requirement and safety engineering domains, and the partialTIM between Requirements and Safety domain, and so has four elements of type ModelRef representing each of them. The mapping entry, named 'Assess-US2ASS', maps the 'Assess' trace link type, in TIM, to the 'UserStory-ToAssessmentTL', in the partialTIM between requirement and safety engineering domain. It contains definitions indicating how to initialise the 'Assess' object in the generated model (reserved words start with '_'). For example, the first definition indicates that the name of the 'Assess' object would be the same as the name of the source model element and the second definition says that the 'traceLinkEnd1' of the new object is equal to the 'assessment' property in the source model element. The equivalence entry, named 'SafetyStory-DerivedSafetyReq', shows that the 'SafetyStory' and 'DerivedSafetyRequirement' are equal and the correspondent trace link type is 'SafetyStoryToDerivedSafetyReqEL' which is defined in ReqSafety-PartialTIM (refer to Figure 4.12). The definitions for the equivalence entry indicate how each end point of the entry relates to the corresponding trace link.

4.3.2 Generating the Traceability Model

In this step, the project-wide traceability model is generated and populated with the information extracted from source models, including domain-

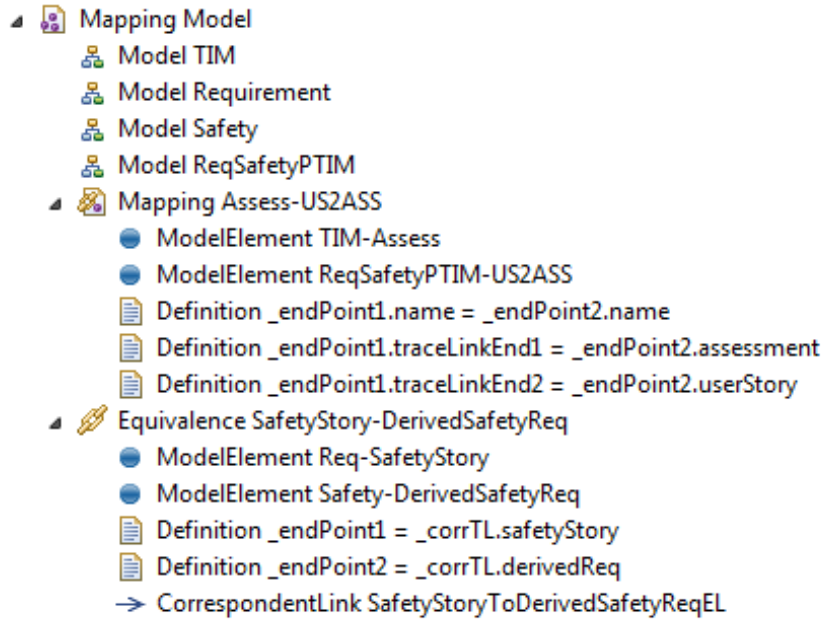


Figure 4.16: Part of the mapping model for IADDS project

specific models and partialTMs, based on the mapping model, as illustrated in Figure 4.13. The TM is created and populated automatically by a domain-specific model transformation –a so called *dynamic model transformation*– which takes the mapping model and the other source models as input and produces the final TM as output.

The transformation is dynamic as it does not contain the low-level (executable) transformation rules, which explicitly define which model elements are transformed to which model elements. In the dynamic model transformation, rules are determined dynamically and executed as the transformation runs. The transformation can be considered to be a higher-order transformation (HOT). A HOT is a model transformation such that takes a transformation model as input and produces a transformation model as output [Tisi et al., 2009]. In comparison to HOT which generates executable transformations rather than executing them, the dynamic transformation performs the transformation *and* generates the target model.

Accordingly, the dynamic model transformation is a transformation engine which performs the transformation based on the two recurring patterns of the rules that we identified in generating the TM. These two patterns define the semantics of the mapping model:

1. Create an object of subtype of 'TraceableElement' or 'TraceLink' in the TM in respect to a specific object of type EClass in a source model.

2. Create an object of subtype of 'TraceLink' in the TM for a given association between two objects in a source model.

The transformation iterates over the mapping entries (defined in the mapping model) and, for each entry, identifies the matching pattern for the entry and, accordingly, carries out the transformation. For each entry, it finds the corresponding model element from source models to the type defined in the second endpoint of the entry (*endPoint2* in the Figure 4.15). Then, it creates or updates the corresponding model element (in the TM) to the type defined in the first endpoint (*endPoint1* in the Figure 4.15) of the mapping entry. Finally, it initialises the output element based on the definitions expressed in the entry and puts a reference to the original model element in the source model.

The project-wide traceability model is created so as to minimise redundancy and inconsistency; it keeps the minimum information needed. For example, it contains reference to the source elements in the source models instead of completely redefining these elements and each element only includes necessary information for traceability analysis.

The transformation is tested by common techniques (e.g. defining test cases particularly for unusual situations). Therefore, the output traceability model would be a valid model regarding the specification of the transformation; the recovered links are complete and correct (i.e. include all the required links and there are no traces between unrelated elements) according to the provided mapping model (assuming the source models are valid). On the other hand, the project-wide traceability model is also validated with respect to the validity constraints associated with the TIM, which represent the semantics of traceability in the project. In this context, an invalid trace model identifies a problem in the source models (e.g. inconsistency, incompleteness) which should be resolved.

4.4 Maintaining Traceability Model

One of the main activities in traceability is *maintenance*. Traceability maintenance refers to those activities associated with updating pre-existing traces as changes are made to the traced artefacts and the traceability evolves, and creating new traces where needed to keep the traceability relevant and up to date. However, traceability maintenance is also required following changes to the requirements and constraints that drive the overarching traceability strategies, which consequently results in change in the traceability solution and the traceability information [Gotel et al., 2012b].

As discussed in Section 2.1.3.4, traceability maintenance is largely considered in order to keep the integrity of the relationships while the referenced entities continue to change and evolve. Accordingly, in this section, we focus

on the traceability model and discuss how it is updated whenever any of the models to which it refers change.

Models (domain-specific and inter-domain) change as the system (or software) artefacts evolve. In our approach we assume that changes in domain models are captured and managed within the scope of the domain possibly with domain-specific tools and techniques. For inter-domain models (partialTMs), which are basically created for a solution, specific techniques should be applied. Considering the way these models are created for the first time (Section 4.2.3), they are updated through (re-)creating them again (manually or semi-automatically).

As explained in Section 4.3.2, the traceability model is created by the dynamic model transformation. The transformation uses these models as input, extracts information from them, and generates the TM automatically based on the mapping model. Therefore, whenever one of the source models changes and the change was propagated completely, the traceability model can be regenerated automatically by the dynamic model transformation with the updated models as input. This transformation happens when a user want to access the traceability information for example to perform an analysis.

In order to control the cost of managing a change, a change could be investigated in more detail to determine if it will affect the TM or not, before regenerating the TM. This is because some of changes, mainly in domain-specific models, happen on those parts of models that are not involved in traceability and so they do not relate to the TM and do not affect it. Doing so, the mapping model can be searched to find out if there is any object of type ModelElementRef which refers to any of the altered elements (the main element and the affected elements). If so, the TM has to be regenerated otherwise, there would be no update for the TM. Although, in this way, we need an additional model operation (to inspect the mapping model with respect to a change) to observe if the TM should be regenerated or not, it would be helpful for cases in which the TM has been enriched with additional information such as transitive links. In such cases, it is recommended to keep the TM unchanged as much as possible.

Nevertheless, in the context of this work, the change and evolution in the traceability goals and their effects on the other traceability-related models have to be also studied to provide a *flexible* traceability solution. In Section 7.5.3, we identify other change scenarios in the context of the proposed approach and discuss how those scenarios could be managed with potential model-based approaches.

4.5 Traceability Analysis

Traceability information is captured and recorded to support traceability goals. As mentioned before, traceability goals are usually explained in high

level terms. For example, they can be expressed as 'traceability of designs against requirements' or 'track the allocation of requirements to system components'. To be able to support the goals, we need to define concrete traceability analyses which can be directly applied to the traceability model to determine whether traceability goals are satisfied. Doing so, each traceability goal may result in one or more traceability analyses.

In the context of this thesis, the analyses are formal descriptions of those questions which were defined (through GQM) to identify traceability-related concepts in a project (Section 4.1.2). A traceability analysis is then a trace query or a constraint which could be expressed with a model management language in terms of the concepts defined in the TIM and applied on the TM.

Generic query languages (e.g. SQL) and model management languages can be used to express traceability analyses, but these require knowledge of the underlying structures in which the traceability information is stored. For example, in Sparx Enterprise Architect^{TM} [Sparx Systems Pty Ltd., 2014] queries are modelled as SQL statements on the underlying database; this requires substantial knowledge of internal data models. To address the aforementioned challenges, research has identified languages and notations to support traceability queries or adopting standard query languages such as SQL or XQuery. One of the main goals of such languages is to allow users to specify queries at an abstraction level that focuses on the traceability perspective of the project. For example, [Schwarz et al., 2008] uses graph-based querying approach to extract traceability information. In this approach, graphs constitute an abstract representation of artefacts and their traceability relationships. [Maletic and Collard, 2009] introduce a Traceability Query Language (TQL) based on XML. In their approach, TQL is build on top of XML addressing language (XPath) and TQL queries are transformed into XQuery. [Mäder and Cleland-Huang, 2010] also present a Visual Trace Modelling Language (VTML) which allows users to model queries visually within their proposed approach to provide goal-oriented traceability.

Considering the context of this work, general-purpose model management languages, such as EOL [Kolovos et al., 2006b] and OCL [Object Management Group, 2012], similarly require knowledge of the structure of the models and also they are not at an appropriate level of abstraction recommended to describe traceability analyses. This is because using such imperative languages, directly, requires users to explicitly specify how to apply an analysis on the TM and generate an arbitrary output model. However, traceability users are just interested in expressing an analysis regardless of how it is implemented and how it generates an output model.

Accordingly, we suggest defining a task-specific analysis language to express traceability analyses at the traceability abstraction level, called the Traceability Analysis Language (TAL). TAL is a task-specific analysis language –bound to the TIM– which hides the complexity of the underlying

information and how it is stored and represented. The TAL is compatible with the implementation infrastructure with which we have worked throughout this thesis.

4.5.1 Traceability Analysis Language (TAL)

As mentioned above, TAL is a task-specific language to specifying traceability queries or constraints at the traceability abstraction level. Additionally, it is a TIM-specific language as it lets users to express analyses by using the traceability domain terminology and project-specific terms (defined in the TIM).

The TAL is a textual language which supports two types of analysis:

1. Query: to find specific traceable elements (or a number of them) that satisfy given conditions

2. Constraint: to check if the traceability model satisfies specific conditions

Through experiments and analysis examples, we found out that the following existential conditions are encountered very often in analyses:

ForAll: all the elements satisfy the condition

Exist: at least one element satisfies the condition

ExistOne: there is only one element which satisfies the condition

4.5.1.1 Concrete Syntax

Regarding the above mentioned requirements for a suitable traceability analysis language, we define the concrete syntax of the TAL which enables users to describe queries and constraints in their level of abstraction. Listing 4.2 displays the concrete syntax in general.

```
(<comment>)?
Query <name> (<inputParameter> : <traceableElement>)* {
  find | count <tracableElement> as <alias>
  (where (<condition>))?
}

(<comment>)?
Constraint <name> (<inputParameter> : <traceableElement>)* {
  forAll <traceableElement> as <alias> (where <condition>)? : (<
      condition>)
  | exist <traceableElement> as <alias> where (<condition>)
  | existOne <traceableElement> as <alias> where (<condition>)
}
```

Listing 4.2: The concrete syntax of TAL

Each Query or Constraint has a name and may have a number of input parameters. Queries *find* or *count* the number of model elements with the given condition described for the target model element type (*traceableElement*). Constraints express a validation constraint on specified model elements, using *forAll, exist, existOne* analysis types. Input parameters are defined in order to be enable to execute a query or check a constraint with respect to a given element of specified types (e.g. finding related hazards to a specified safety story, 'SS1', in context of IADDS project).

A Query can define a condition which includes a number of statement following the *where* keyword. Each constraint defines a condition following the *where* keyword, in case of using *exist* and *existOne*, or a colon (:) in case of using *forAll*. A constraint with *forAll* may also restrict the elements on which its condition has to be applied. This restriction is defined as a condition following the *where* keyword and before the colon (:). A condition can be described with one or more expressions. An expression could be a simple comparison (e.g. greater than, less than, and equal) or a constraint defined with one of the validation functions (*forAll, exist, existOne*). Additionally, the expression may use built-in functions (e.g. size(), isDefined(), and includes()), which are provided in the language.

Additionally, TAL provides an expressive syntax in the context of traceability. For example, to express the existence of a paticular trace link type between two elements the following syntax is provided, which can be used in conditions:

<traceableElement1> <***traceLinkX***> <traceableElement2>

The above statement means there exists a trace link of type 'traceLinkX' with 'traceableElement1' as *traceLinkEnd1* and 'traceableElement2' as *traceLinkEnd2*.

On the other hand, considering the concrete syntax, it is easily observed that the valid values for the *TracableElement* and *TraceLink* are exactly the TraceableElements and TraceLinks defined in the TIM. Therefore, it seems possible to explicitly specify these values in the language. The grammar can be generated based on a given TIM and so defines the permissible values for these two terms. In this thesis, the proposed TAL is defined with Xtext [The Eclipse Foundation, 2013b], which is a textual grammar language. Accordingly, the TAL grammar is automatically generated based on the TIM using M2T model transformations. We will explain how the grammar of the TAL is defined and generated in more detail in Section 5.2.5.

As a result, the TAL is a project-specific analysis language (TIM-specific) and provides early validation for analysis scripts to increase precision and consequently reduce errors. The main feature of a TIM-specific TAL is to clearly specify the terms permitted in an analysis script which contains the name of traceable objects (TraceableElements), trace link types (TraceLinks), and possibly their features.

4.5.1.2 Abstract Syntax

The abstract syntax of the TAL is defined based on the grammar language used to describe a language namely Xtext. Figure 4.17 illustrates the abstract syntax of the TAL (generated with Xtext). We now describe the language in detail.

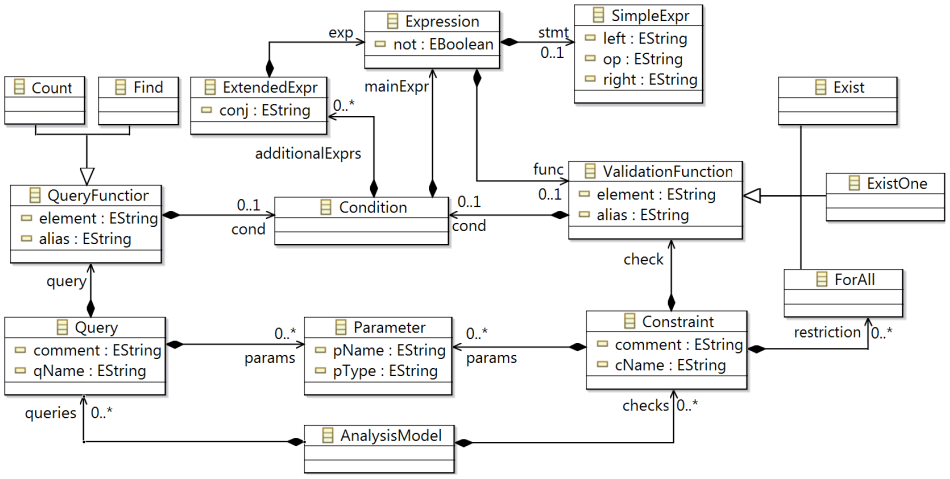


Figure 4.17: The abstract syntax of TAL

AnalysisModel: Acts as the root of the analysis model. It defines a *name* and contains a number of Queries and Constraints.

Query: Represents a traceability query to find specific elements. It defines a *name* and *comments*, and contains a QueryFunction. A Query might have a number of Parameters as input parameters.

Constraint: Represents a constraint on the traceability model. Similar to Query, it defines a *name* and *comments*, contains a ValidationFunction, and might have input Parameters.

QueryFunction: Defines the function for the owner Query. It defines the target model *element* (on which the query is executed) and an *alias* to refer to the element, and may contain a Condition.

ValidationFunction: Defines the function for the owner Constraint. It defines the target model *element* (on which the query is executed) and an *alias* to refer to the element, and may contain a Condition.

Condition: Represents the condition for queries or constraints. It contains a main Expression and may have a number of additional ExtendedExprs.

Expression: Represents an expression to explain the condition for queries or constraints. It contains a SimpleExpr and can have a ValidationFunction to express a constraint (such as exist, exist one, and for all)

on model elements.

SimpleExpr: Is a simple statement over model elements. It is used to specify existence of a trace link between two traceable elements or to describe a desired relationship between two model elements, such as less than and equal. It can also be used to evaluate the result of calling a built-in function on a model element, such as 'includes()' and 'size()'. Each Statement defines the one or two model elements that it works with (left and right operands) and the *operator* (e.g <, >, and =).

ExtendedExpr: Is an extended Expression which has 'And/Or' in addition to an Expression.

Parameter: Represents a parameter in Queries or Constraints. It defines the name and type of the model element that it would refer to.

Find, Count: A Subtype of the QueryFunction.

Exist, ExistOne, ForAll: Subtypes of ValidationFunction.

4.5.1.3 Example

Listing 4.3 shows an example TAL query, named 'unMitigatedHazards', to find those 'Hazard' (a traceable element in IADDS project) which are not mitigated in the project by a 'SafetyStory'. As shown in the listing, 'SafetyStory', 'Hazard', and 'Mitigate' are examples of TIM-specific terms that are allowed to be used in trace queries or constraints.

```
//Find hazards which are not mitigated
Query unMitigatedHazards {
  find Hazard as HZD where not (
    exist SafetyStory as SS where (SS Mitigate HZD)
  )
}
```

Listing 4.3: Example TAL query

In Listing 4.4, a TAL constraint is defined to check that all 'SystemRequirements' are at least addressed by one 'UserStory'.

```
//Each system requirement has to be addressed at least with one
    user story
Constraint allSysRequirementsAddressed {
  forAll SystemRequirement as SR :
   (exist UserStory as US where (US Address SR))
}
```

Listing 4.4: Example TAL constraint

4.5.2 Analysis Result

Each TAL script is executed on the project-wide TM. The result of executing a traceability analysis on the TM is a *Result Model*. The model contains the

output model elements and shows the overall result of the analysis mostly in case of checking a constraint in the TM. The result model can be used to support extended user-defined usage scenarios (e.g. generate reports) which can be implemented with model management languages.

Figure 4.18 depicts the metamodel of the result model which is explained in the following.

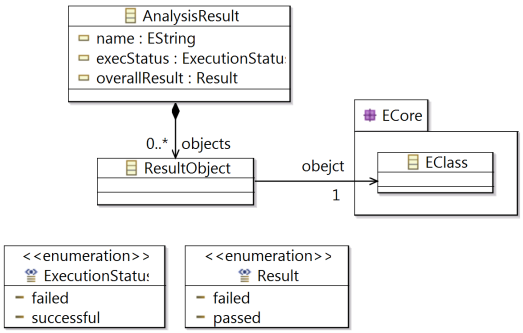


Figure 4.18: Analysis result metamodel

AnalysisResult: Acts as the root of the result model. It defines a *name* and contains a number of ResultObject (in case of querying the TM). It also shows the overall result of the analysis, in case of checking a constraint, indicating if the model passes or fails to satisfy the constraint. The result model shows the execution status of the script representing if it is failed (due to technical errors or problems) or successful.

ResultObject: Represents the model elements satisfied the query. It associates with the original model element in the TM.

ExecutionStatus: Provides the valid values for the execution status of the EOL script.

Result: Provides the valid values for the overall result of the analysis.

4.6 Chapter Summary

In this chapter, we presented the main contributions of this thesis. In Section 4.1, we introduced a three-step method to define a project-specific TIM, which includes determining traceability goals, identifying related concepts in the project, and finally representing the TIM in a formal way. Section 4.2 explained how to investigate existing models in a project to identify available traceability-related information which could explicitly or implicitly support traceability. In Section 4.3, we presented a systematic approach to extract traceability-related information from other models in the project and generate a project-wide traceability model. Also, in Section 4.4, we demonstrated

how the traceability model is maintained when the relevant models change and evolve over the time. Finally, Section 4.5 introduced the Traceability Analysis Language (TAL) and it was discussed how it facilitates traceability analyses.

Tables 4.1, 4.2, 4.3, 4.4, 4.5, and 4.6 show the characteristics of the proposed approach with respect to the parameters defined in Section 3.1.

Table 4.1: Overview of the proposed approach - Artefacts

	Artefact			
	Type		Format	
	General	Specific	General	Specific
The Approach	✓			Models

Table 4.2: Overview of the proposed approach - TIM

	TIM		
	General-Purpose	Case-Specific	
		Static	Customisable
The Approach			✓

Table 4.3: Overview of the proposed approach - Tooling

	Tooling			
	General	specific		
		Partial	Complete	Interoperable
The Approach			✓	✓

Table 4.4: Overview of the proposed approach - Planning and Management

	Planning and Management			
	Identity Req.	Define TIM	Define Process	Assessment
The Approach	✓	✓	✓	

Table 4.5: Overview of the proposed approach - Trace Creation

	Trace Creation									
	Automation			Acquisition					Storage	
				Capture		Recovery				
	Manual	Semi-automatic	Automatic	Transform	Event-based	IR	Rule	Misc	Model	Repository
The Approach		✓						✓	✓	

Table 4.6: Overview of the proposed approach - Maintenance and Usage

	Maintenance					**Usage**						
	Automation			Mode		Visualisation			Retrieving			
										Query		
	Manual	Semi-automatic	Automatic	Reactive	Proactive	Matrix	Graph	Link	Predefined	API	General	Specific
The Approach		✓		✓								✓

5

Implementation

Tool support for a traceability framework is essential to maximise the return on investment in supporting traceability. A traceability tool is an integral part of a traceability solution to assist or automate any part of the traceability process [Gotel et al., 2012b]. Accordingly, a prototype of the required infrastructure for the proposed traceability solution has been developed which is explained in this chapter.

For the implementation, EMF [Eclipse Foundation, 2013], Epsilon [Kolovos and Paige, 2013], Xtext [The Eclipse Foundation, 2013b], and Xtend [The Eclipse Foundation, 2013a] have been used. EMF has been used as the modelling framework for this work. In addition to the familiarity of the author with EMF, it has been chosen because it is considered as a de facto standard modelling framework being supported by open-source high quality tools and editors.

In addition to EMF, the Epsilon framework has been selected for developing the various model management tasks, such as model transformations and comparisons. Similarly to choosing EMF as modelling framework, Epsilon has been chosen because of its familiarity to the author and due to the high level of re-use available in the platform and its presence in a research community. Epsilon is an ideal host for the rapid prototyping of languages for better supporting model management activities.

Finally, a combination of Xtext and Xtend has been used to develop the traceability analysis language and generate the infrastructure required to use the language to express and run traceability analyses. Xtext is a framework for development of programming and domain specific languages [The Eclipse Foundation, 2013b]. Xtend is a programming language which is supported with the provided code generation facilities in Xtext. Xtext and Xtend have been used as they are well integrated with Eclipse and EMF modelling framework.

Accordingly, EMF and Epsilon are appropriate choices to implement the prototype for the traceability solution. However, in principle, any other

modelling framework and model management framework could have been used for developing a tooling support for the proposed approach in this thesis.

This chapter provides an overview of the infrastructure, used to implement a prototype of the tooling required to support the introduced traceability solution. It also introduces and explained the prototype in detail and highlights its main parts.

5.1 Infrastructure

5.1.1 Eclipse Platform

Eclipse is an open-source software development environment, which consists of an integrated development environment (IDE) and an extensible plug-in system. The Eclipse plug-in system allows developers to extend the core of the tool by contributing functionality in the form of plug-ins. Due to this extensibility mechanisms, a wide range of tools have been developed atop Eclipse, such as editors and execution engines for programming languages, as well as modelling tools such as EMF and GMF.

Eclipse is the platform of choice for MDE tools as the majority of contemporary MDE languages and frameworks provide Eclipse-based development tools [Kolovos, 2008]. Moreover, due to the aforementioned extensibility mechanisms, the development of new tools and their integration with existing tools can be done with little effort. Given these reasons, Eclipse was chosen as the platform atop which the prototype is developed.

5.1.2 Eclipse Modelling Framework

Eclipse Modelling Framework (EMF) is a modelling framework and code generation facility for building tools and other applications based on a structured data model [Steinberg et al., 2009]. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. Models can be specified using annotated Java, XML documents, or modelling tools like Rational Rose, then imported into EMF. Most important of all, EMF provides the foundation for interoperability with other EMF-based tools and applications.

EMF provides a metamodelling language, named Ecore, for defining the structure of models. Ecore is an implementation of the Meta Object Facility (MOF) 2.0 specification [Object Management Group, 2011b].

5.1.3 Epsilon

Epsilon (Extensible Platform for Specification of Integrated Languages for mOdel maNagement) [Kolovos et al., 2010] is a platform for building consistent and interoperable task-specific programming languages which can be used to interact with EMF models to perform common model management tasks such as model transformation, model comparison, and validation.

The purpose of Epsilon is to consolidate the common features of the various task-specific languages in one base language and then develop the various model management languages atop it. The base language is EOL. Moreover, Epsilon provides the Epsilon Model Connectivity (EMC), which abstracts over the different modelling frameworks and enables the Epsilon task-specific languages to uniformly manage models of those frameworks. Apart from the existing model management languages, Epsilon provides appropriate extensibility mechanisms to implement new task specific languages with minimal replication. The architecture of the Epsilon framework and its family of languages are illustrated in Figure 5.1.

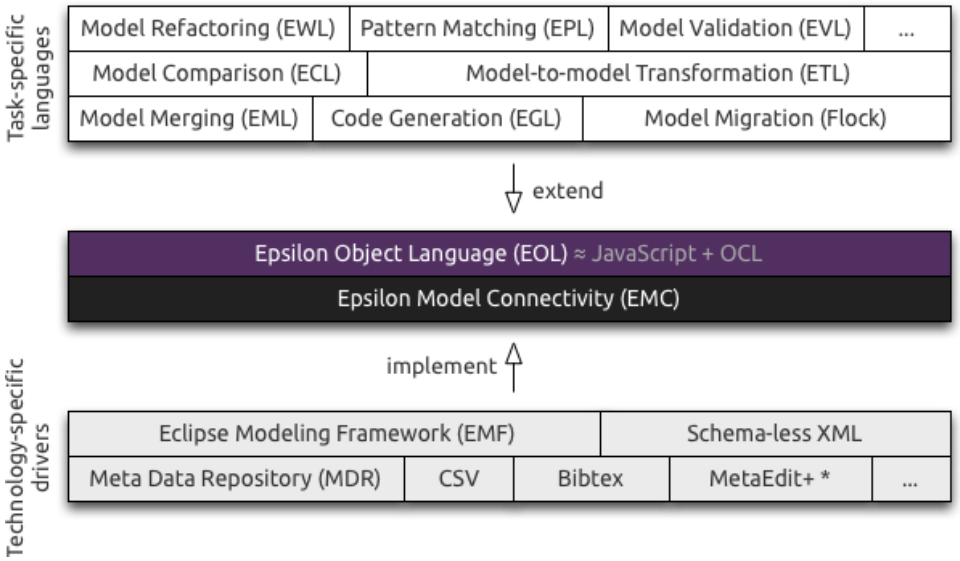


Figure 5.1: The architecture of Epsilon [Kolovos and Paige, 2013]

At the bottom is the EMC, which abstracts the various modelling technologies such as EMF and Metadata Repository (MDR) [Matula, 2003]. Atop EMC, EOL is developed, which is the basis for the current and future task-specific languages. Since Epsilon framework is built in Eclipse, it is well integrated with other Eclipse development tools, through the use of the plug-in system provided by the Eclipse platform [Kolovos et al., 2006a]. The task specific languages used for this work are briefly described in the

following.

5.1.3.1 Epsilon Object Language (EOL)

At the core of Epsilon is the Epsilon Object Language (EOL) [Kolovos et al., 2006b], an imperative model-oriented language that combines the procedural style of JavaScript with the powerful model querying capabilities of OCL [Object Management Group, 2012]. EOL provides a reusable set of common model management facilities, atop which task-specific languages can be implemented. However, EOL can also be used as a general-purpose standalone model management language for automating tasks that do not fall into the patterns targeted by task-specific languages. It supports statement sequencing, model modification capabilities, simultaneous access to multiple models which conform possibly to heterogeneous metamodels. In this work, EOL has been used in different parts. It is used to develop the dynamic model transformation to generate the traceability model (Section 5.2.4). Additionally, traceability analyses are ultimately translated into EOL programs which are then executed on the TM (Section 5.2.5).

5.1.3.2 Epsilon Transformation Language (ETL)

ETL [Kolovos et al., 2008] is a hybrid, rule-based, model-to-model transformation language built on top of EOL. It is capable of transforming an arbitrary number of source models into an arbitrary number of target models. Moreover, as ETL is based on EOL, it reuses its imperative features to enable users to specify particularly complex, and even interactive, transformations. In the context of this work, ETL is not used explicitly. However, it is one of the options to impelement and perform the required transformation to generate the TM (discussed in Section 5.2.4).

5.1.3.3 Epsilon Validation Language (EVL)

EVL [Kolovos et al., 2009] is the validation language of the Epsilon platform to specify and evaluate constraints on models of arbitrary metamodels and modelling technologies. It is an OCL-like validation language, which additionally supports dependencies between constraints, customizable error messages and specification of fixes, which are invoked to repair inconsistencies. Similarly to the other languages of the Epsilon framework, EVL builds on top of EOL. This enables it to evaluate inter-model constraints. In the proposed approach, EVL is used to express correctness constraints which apply to metamodels and models.

5.1.3.4 Epsilon Generation Language (EGL)

EGL [Rose et al., 2008] is a template-based, model-to-text transformation language for generating various types of textual artefact, including executable code, documentation, and other textual artefacts from models. EGL offers model-to-text specific features such as protected regions for mixing generated code with hand-written code and extensible template system. Similarly to ETL, EGL is built atop EOL, therefore it has access to general model management support. In the context of this work, EGL is used to generate the Xtext grammar for the TIM-specific TAL which is explained in Section 5.2.5.

5.1.3.5 Epsilon Wizard Language (EWL)

EWL [Kolovos et al., 2007] is a language, whose aim is to support interactive in-place update transformations on user-selected model elements. The niche of EWL is the automation of recurring model editing tasks such as model refactorings. EWL is used in this thesis in the context of building the mapping model. Two EWL wizard have been developed to provide a list of the metamodels, attached to the mapping model, and a list of all the EModelElements in these metamodels to the user, in order to choose and assign the *target* for ModelRefs or ModelElementRefs in the mapping model. They will be explained in Section 5.2.3.

5.1.4 Xtext & Xtend

Xtext [The Eclipse Foundation, 2013b] is a framework for development of programming and domain specific languages. It provides a *grammar language* which is a domain-specific language, designed for the description of textual languages. The grammar language allows to describe the concrete syntax and how it is mapped to the semantic model of the language. The Xtext language generator generates the parser and serialiser and some additional infrastructure code. For example, it generates an IDE plug-in which provides the generated entity editor with various functionalities including code completion, syntax highlighting, syntactic validation, linking errors, the outline view, and find references. Figure 5.2 shows a screen shot of the generated editor for the IADDS project.

One of the Xtext artefacts is a code generator stub which is added to the language project. The code generator can be used to generate arbitrary types of code for the new language. The Xtext language generator uses Xtend [The Eclipse Foundation, 2013a] and generates a Xtend class which is used to generate the code. Xtend is a statically-typed programming language which translates to comprehensible Java source code. Syntactically and semantically Xtend has its roots in the Java programming language

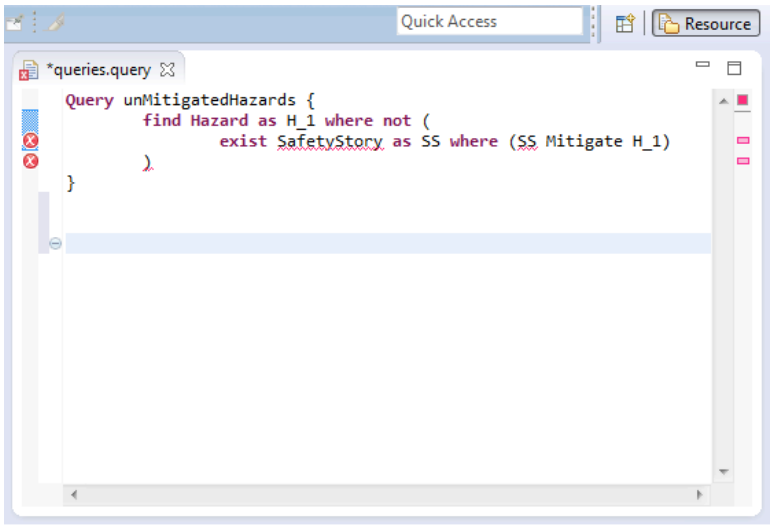


Figure 5.2: A screen shot of the generated editor for the TAL

but improves on many aspects including extension methods, lambda expressions, and template expressions. Xtend is well integrated with Xtext and provides template expression which is an important feature for code generation. Xtend's capability to describe templates allows users to define code templates regarding the various element types defined in the grammar. The code generator is called by the language compiler whenever a statement in the new language is created.

In this work, we describe the analysis language using Xtext grammar language and then generate the language infrastructure required to use the language to define traceability analyses. With code generation facilities provided with Xtext, each TAL script is automatically transformed into an EOL script to be executed on the TM.

5.2 Prototype

This section introduces the prototype implemented as the tool support for the proposed approach in this thesis. It explains the main elements and parts of the prototype which are related to and support the novel parts of the approach.

The prototype is a model-based environment which provides the foundation required to develop a traceability solution according to the proposed approach and use it to support usage scenarios. As mentioned before (Figure 4.1), a solution has two aspects: infrastructure and operational. In this context, the tooling initially allows engineers to build the required infras-

tructure for a solution: explain metamodels, create utility models, define or perform model management operations, and generate the analysis language and its editor. As a result, an operational environment is prepared and provided to traceability users which allows them to create (or regenerate) the traceability model, by executing the provided operation (dynamic model transformation), describe traceability analyses, and use their results.

In the following, we focus on the novel parts of the approach and facilities accordingly provided to engineers in order to build an infrastructure. We describe in detail how a mapping model is created (Section 5.2.3), the dynamic model transformation is defined and works (Section 5.2.4), and the analysis language is generated (Section 5.2.5). We also explain how metamodels and models are generally defined, manipulated, and visualised in the context of this implementation (Section 5.2.1). Additionally, it is demonstrated how an extension to the CoreTIM can be defined as an example extension to the proposed approach (Section 5.2.2).

Figure 5.3 gives an overview of the implementation infrastructure and technologies used to implement each part of the prototype.

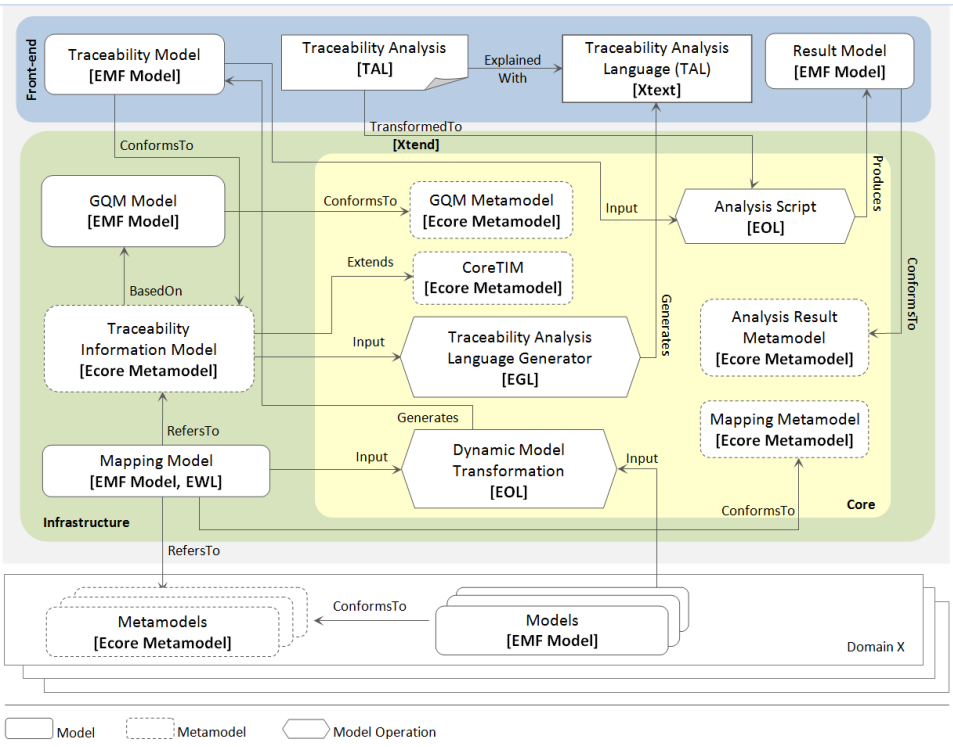


Figure 5.3: Technologies used to implement the tooling for the approach

5.2.1 Metamodels and Models

Metamodels can be represented with any metamodelling language, such as a UML class diagram [Object Management Group, 2010b] or Ecore metamodel [Steinberg et al., 2009]. We use Ecore metamodelling language to describe all metamodels in the traceability solution. The TIM and other metamodels, including GQM metamodel, mapping metamodel, domaind-specific metamodels, and partialTIMs, are defined and described as Ecore metamodels. The metamodels are accompanied with EVL scripts to express and check complex validity constraints on their related models.

For our model management operations, we need to locate the main EClass of each metamodel, which is the EClass that contains all the other model elements, and we call it the *entry point*. This allows us to use the entry point, as the container of the other elements, to access the inner elements in order to do model operations. Doing so, we use EAnnotation to specify the entry point. The details of annotations have the form of key-value mappings and their purpose is to capture useful details for the model management task of interest. In this respect, the following EAnnotation is added for the main class of each Ecore metamodel.

```
1 @traceability(type="ModelEntryPoint")
```

We use basic EMF tree editors to create models (instantiate metamodels) and then manipulate them accordingly. However, in some cases, the editor has been customised by using the EXtended Emf EDitor (Exeed) [Kolovos, 2007]. This is achieved by adding Exeed specific EAnnotations to the EClasses of the metamodels. These annotations provide instructions about how to format labels and icons of the editor and then Exeed uses this information to visualise the models accordingly. For example, the editor has been customised for the mapping model with user-defined icons and labels for each model element. A screen shot of the Exeed editor for a mapping model is illustrated in Figure 4.16.

5.2.2 Extended CoreTIM

As mentioned before in Section 4.1.3.2, traceability metamodels can define or contain other concepts in addition to what is defined in the CoreTIM, depending on the specific requirements for traceability information or the preferences of engineers who define the TIM. In this context, we suggested to define ExtendedCoreTIM which is an extension to the CoreTIM and can be used as the base metamodel for defining project-specific TIMs similarly to the CoreTIM.

In our prototype of the required tool support, we focused on the GQM models and tried to attach it to a TIM. This relationship helps engineers to understand why specific elements have been defined and also helps them in later stages to make decisions particularly in case of change. Accordingly, we

defined an ExtendedCoreTIM of which the main feature is that it relates the GQM model developed in a project to its traceability metamodel in order to demonstrate the rationale behind the definition of the metamodel. The abstract syntax of the ExtendedCoreTIM is illustrated in Figure 5.4.

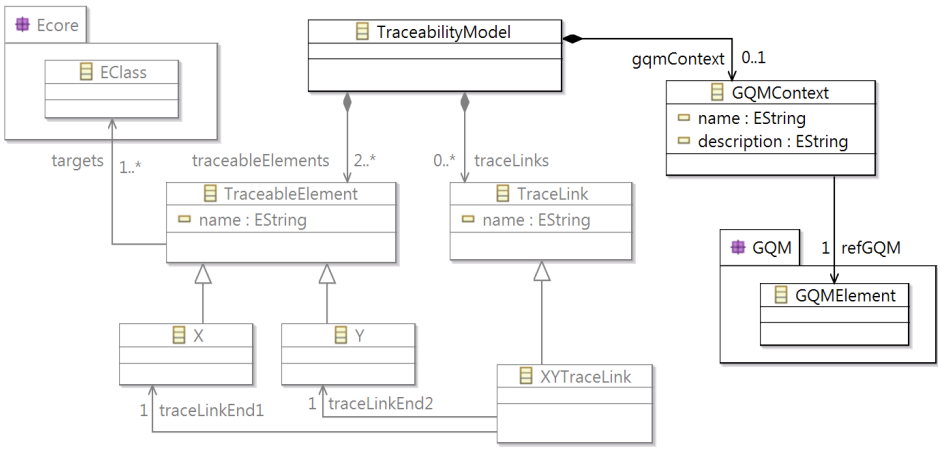


Figure 5.4: Abstract syntax of ExtendedCoreTIM

Basically, it extends the CoreTIM with GQMContext, which represents the context within which its associated element is defined. A GQMContext defines a name and a description, has a reference to a GQMElement (any element in a GQM model), named *refGQM*, and is associated with TraceabilityModel. Accordingly, a TraceabilityModel contains a GQMContext, called *gqmContext*, which in practice points to the GQM model of the project (the element of type GQMModel in the model). When a TM is generated, the *gqmContex* attribute of its main class (entry point) is assigned to the main class of the GQM model.

Note that a GQMContext could not be associated with TraceableElements and TraceLinks as the supportive information provided by a GQM model is defined in the metamodel level. It demonstrates that why a specific *type* is defined in the metamodel not why *each instance* is defined in a model. Therefore, a GQMContext is just associated with TraceabilityModel which has one instance a TM. However, the GQM model can be used to annotate the project-specific traceability metamodel.

To annotate the traceability metamodel, we developed a wizard with EWL which works on GQM models and allows users to locate the related TraceableElements or TraceLinks for a GQMElement. Then, the related element in the TIM is annotated accordingly. The EWL wizard, called *annotateTIM* is shown in the Listing 5.1.

```
1 wizard annotateTIM {
```

```
2  guard : self.isKindOf(GQMElement)
3  title : "Annotate TIM"
4  do {
5   var traceElements = getTraceElements();
6   var theElement = UserInput.choose("Select Element",
        traceElements);
7   if (theElement.isDefined()) {
8    var emfTool = new Native("org.eclipse.epsilon.emc.emf.
        tools.EmfTool");
9    var ecoreUtil = emfTool.ecoreUtil;
10
11   //The resource representing the TIM Ecore
12   var timResource = theElement.eResource().getResourceSet().
        getResources().get(1);
13   //To be able to modify the TIM
14   timResource.setTrackingModification(true);
15
16   //Annotate the element
17   ecoreUtil.setAnnotation(theElement, "GQMContext", "type",
        self.eClass().name);
18   ecoreUtil.setAnnotation(theElement, "GQMContext", "name",
        self.name);
19   ecoreUtil.setAnnotation(theElement, "GQMContext", "
        elementRef", self.asString());
20  }
21 }
22 }
```

Listing 5.1: EWL wizard to annotate a TIM according to a GQM model

As shown in the listing, the wizard is called and executed when the current model element is of type GQMElement (the *guard* in line 2 specifies this). The main part of the wizard is the *do* block in which a collection of the trace elements (all subtypes of TraceableElement and TraceLink defined in the TIM) is provided to the user to select the related element to the current GQMElement. The operation *getTraceElements* returns a collection of trace elements and the TIM is defined as a resource for the GQM model. Then, the selected element, from the TIM, is annotated with the specific values. The annotation has the following format and the parameters are replaced with the type of the current GQMElement (goal, question, or concept), its name, and a reference to it.

```
1  @GQMContext(type="<elementType>", name="<elementName>",
       elementRef="<elementRef>")
```

According to the proposed approach, in practice, engineers extend the ExtendedCoreTIM (or the CoreTIM) to define a project-specific TIM, which contains specific TracableElements and TraceLinks. Then, they need to *locate source elements*; determine how the TIM is related to other models and *create a mapping model*. The next section (5.2.3) explains how a mapping model is created, particularly with the use of facilities provided in Epsilon.

5.2.3 Mapping Model

As explained in Section 4.3.1, the mapping model formally specifies how each traceable element or trace link type defined in the TIM relates to model element types in the other metamodels. Accordingly, it refers to metamodels, including the TIM, domain-specific metamodels, and partialTIMs.

The mapping model is created as an EMF model by instantiating the mapping metamodel (Figure 4.15). Then, the metamodels with which the model works are determined and, so, the ecore metamodel of the TIM and other related metamodels are defined as additional resources for the model. In this way, it is possible to refer to the metamodels and the model element types, defined in them, while populating the mapping model. For each of these metamodels an element of type ModelRef is added to the mapping model. To assign the reference of each ModelRef element to the corresponding metamodel, an EWL wizard has been developed which allows an engineer to easily select the metamodel.

The EWL provides a list of all the metamodels (already defined as additional resources) and assigns the value for the attribute *theMetaModel* in the ModelRef model element to the selected metamodel. The EWL wizard, called *listMetaModels*, is shown in the Listing 5.2.

```
1  wizard listMetaModels {
2   guard : self.isTypeOf(ModelRef)
3   title : "MetaModels"
4   do {
5    var metaModels = getMetaModels();
6    var theMetaModel = UserInput.choose("Select Metamodel",
         metaModels);
7    if (theMetaModel.isDefined()) {
8     self.theMetaModel = theMetaModel;
9    }
10  }
11 }
```

Listing 5.2: EWL wizard to list metamodels: *listMetamodels*

As shown in the listing, the wizard is called and executed when the current model element is of type ModelRef (the *guard* in line 2). The main part of the wizard is the *do* block in which a collection of the metamodels (returned by operation *getMetaModels*) is given to the user to select the desired metamodel. The ModelRef model element has a reference to an EClass (depicted in figure 4.15). This EClass is the entry point of a metamodel, the EClass annotated with @traceability(type="ModelEntryPoint"). In this respect, the operation *getMetaModels*, called in line 5, returns a collection of the entry points of the metamodels. Listing 5.3 shows the opertion *getMetamodels*.

```
1  operation getMetaModels() : Collection (EClass) {
2   var metaModels = new Set;
3   var allEClasses =  EClass.AllInstances;
```

```
4   metaModels.addAll(allEClasses.select(cl | cl.getEAnnotation(
        "traceability").isDefined() and cl.getEAnnotation("
        traceability").getDetails().get("type") = "
        ModelEntryPoint"));
5
6   return metaModels;
7  }
```

Listing 5.3: EOL operation *getMetaModels*

Figure 5.5 shows a screen shot of the list of the EModelElements provided to users.

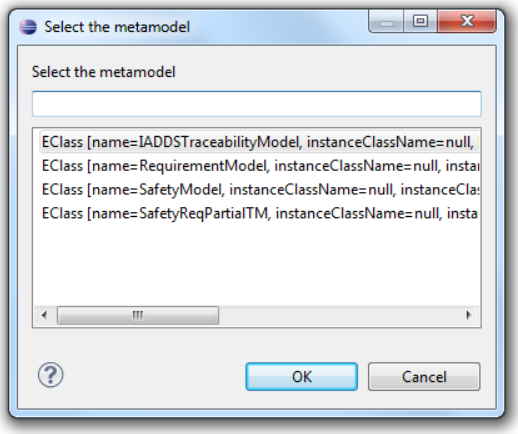


Figure 5.5: The list provided to users to select a metamodel refered by a ModelRef

Having defined the metamodels in the mapping model, the relationships between the TIM and the other metamodels (mapping entries) and the equivalence relationships (equivalence entries) between metamodels other than the TIM should be defined. Each entry has two references to the model elements involved in the relationship. To assign these values in an entry, we have developed an EWL wizard which lists all the EModelElements defined in the metamodels (EClasses, EAttributes, and EReference), allows the user to select the target, and finally assigns the right value for the entry.

Listing 5.2 shows the EWL wizard called *listModelElementTypes*. Similarly to the previous wizard, the wizard is called on the elements of type ModelElementRef (the *guard* in line 2). The operation *getModelElements* returns a collection of all the EModelElements defined in the metamodels, which is provided to the user to select the required element. Then, in line 12 and 13, the metamodel and the model element are assigned for ModelElementRef in the entry.

```
1  wizard listModelElementTypes {
2   guard : self.isTypeOf(ModelElementRef)
3   title : "EModelElements"
4   do {
5    var modelElements = getModelElements();
6    var modelElementStr = UserInput.choose("Select
         EModelElement", modelElements);
7
8    if (modelElementStr.isDefined()) {
9     var theModel = getModel(modelElementStr);
10    var modelElement = getModelElement(modelElementStr);
11
12    self.theElement = modelElement;
13    self.owner = theModel;
14
15    if (modelElement.isTypeOf(EClass)) {
16     self.type =  ModelElementType#Clazz;
17    } else if (modelElement.isTypeOf(EAttribute)) {
18     self.type =  ModelElementType#Attribute;
19    } else if (modelElement.isTypeOf(EReference)) {
20     self.type =  ModelElementType#Reference;
21    }
22   }
23  }
24 }
```

Listing 5.4: EWL wizard to list model element type: *listModelElementTypes*

Each entry would have a number of Definitions in order to explicitly define how two end points are related. The definitions are used in initialising the new model element in the output model (the TM). Definitions are explained textually. There are some keywords (starting with '_') which can be used in the statement for referring to the endpoints of the relation. For example. '_endPoint1' is used to refer to the first end point in the entry, and '_source' is used to refer to the source of a ERefernce element.

Figure 5.6 shows a screen shot of the list of the EModelElements provided to users.

As mentioned in Section 4.3, having defined the mapping model, a project-wide traceability model is generated and populated by a dynamic model transformation which extract traceability-related information form other models based on the mapping model. The next section (5.2.4) explains the transformation in detail.

5.2.4 Dynamic Model Transformation

As explained in Section 4.3.2, the TM is generated automatically by a model transformation operation. The transformation takes the source models and the mapping model, as its input models, and generates the project-wide TM

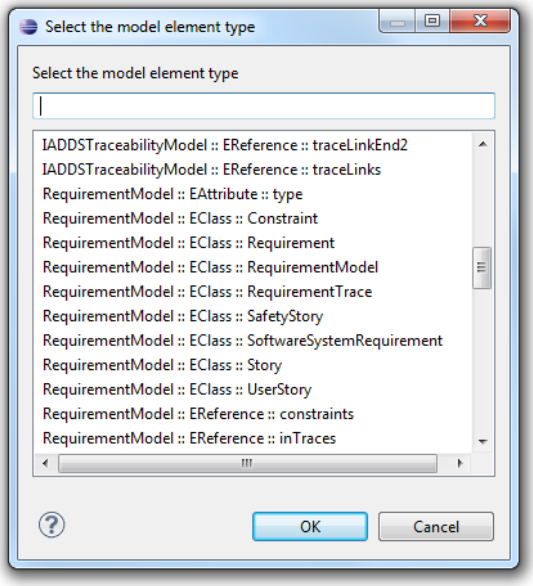


Figure 5.6: The list provided to users to select the endpoints of an entry

as the output model. The dynamic model transformation is published in the final tooling and provided to the traceability users to create the TM.

Basically, there are two options for how to implement and perform the transformation as depicted in Figure 5.7.

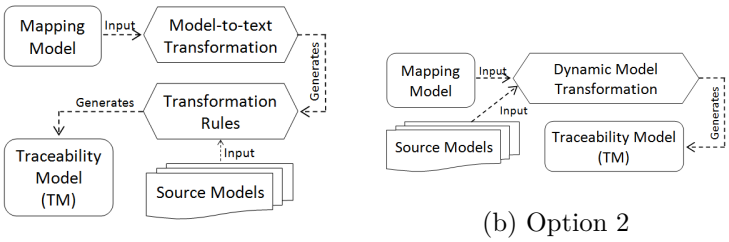
1. Generate transformation rules, based on the mapping model, and then use them to generate the TM

   This option is a higher-order transformation (HOT), which takes a transformation model as input and produces a transformation model as output [Tisi et al., 2009]. In the context of this work, the mapping model is the transformation model which is transformed into an executable transformation model. Regarding the target transformation model (e.g. ETL in our infrastructure), a M2M or M2T transformation language can be used to generate the target transformation, which is thereafter executed on the source models and generates the TM.

2. Develop a domain-specific transformation which performs the transformation dynamically (on-the-fly)

   A domain-specific transformation performs customised transformation for particular purposes, in comparison to usual transformation. In this case, a domain-specific transformation is developed with a general model operation language (e.g. EOL). The transformation dynamically infers the transformation rules from the mapping model and

executes them at run-time. Rules are applied on the source models and the TM is accordingly populated. This way, the TM is generated gradually while the transformation is executed.



(a) Option 1

(b) Option 2

Figure 5.7: The two options to generate the TM

We chose the second option and decided to implement and perform the required model transformation using EOL, which we call it *dynamic model transformation*, instead of transforming the mapping model into an ETL module (HOT). The first reason for this decision was to minimise the additional models and model operations as much as possible. This is because even if they are created or executed automatically they increase the complexity and cost of dealing with change to keep them consistent and updated. In the first option, there would be two model operations: an EGL program used to generate transformation rules in ETL which are thereafter used to generate the TM. In the second case, there is just an EOL program which does the transformation to build the TM.

Additionally, we discovered two recurring patterns for the transformation rules:

1. Create an object of subtype of TraceableElement or TraceLink in the TM in respect to a defined object of type EClass in a source model.

2. Create an object of subtype of TraceLink in the TM for a given association between two objects in a source model.

In this respect, we determined that although the first pattern can be easily implemented by ETL, it would be too complex to explain the rules for the second pattern in ETL and we need to explain some parts of the transformation within user-defined operations which are implemented using EOL. Although ETL is a hybrid language that implements a task-specific rule definition and execution scheme and also inherits the imperative features of EOL to handle complex transformations, we found that describing the required transformation in context of ETL would increase the complexity of

the solution. This is because, as we wanted to generate the transformation rules automatically, we needed to develop a complex EGL program to generate a complex ETL program which should be tested and verified to check if it generates the correct output or not. On the other hand, the second rule pattern is important and frequently used in the context of this work. This is because, fundamentally, we intend to derive trace links from general relationships between elements and objects in other models. Therefore, we decided to develop an EOL program to perform the transformation to generate the TM.

Listing 5.5 shows part of the EOL program used for dynamic model transformation. The complete code of the transformation is provided in Appendix E. The core of the transformation is iterating over mapping entries (Listing 5.5 the *for* statement in line 13) and for each entry creating and initialising a new element in the output traceability model. At each iteration, the source metamodel and its model is determined based on the value of *endPoint2* for the entry (line 16 and 23). Then, according to the pattern of the mapping, the correspondent element is created or updated in the TM. For example, in line 25, the *if* statement shows the case in which an EClass in the TIM has been mapped to an EClass in one of the source metamodels. In this case, for each object of type the EClass in the metamodel, an object of type the correspondent EClass in created and added to the TM (lines 29-34). The output of the transformation is a project-wide TM which conforms to the project-specific TIM (defined in line 6 in Listing 5.5).

Alongside the transformation, a utility model, so called *equivalences*, is generated which shows each model element in the TM is related to which model element from other models. This model is used to find the equivalent element to an element in the source models and use it in the transformation. For example, assume in model M, there are two model elements X and Y and there is also a 1-1 association between them. Both X and Y have an equivalent model element in the TM: X' and Y'. Now, if we have to define a trace link between X' and Y' because of the existing association between X and Y, we need to find X' and Y', the equivalent elements to X and Y which have already been created in the TM, and then create a trace link between these two elements (X' and Y'). The equivalence model is used to find X' and Y'.

```
1 ...
2 //input: mapping model
3 var mappingModel : Any = inputMappingModel!MappingModel.all()
      .first();
4
5 //output: traceability model
6 var traceabilityModel = ecoreUtil.create(TIM.theMetaModel);
7 traceabilityModel.name = mappingModel.name;
8 emfTool.createModel(traceabilityModel, "
      outputTraceabilityModel");
```

```
9  ...
10
11 //iterate over mapping entries
12 var entry : inputMappingModel!MappingEntry;
13 for (entry in mappingEntries) {
14  //endPoint1 always refers to TIM
15  var endPoint1 : inputMappingModel!ModelElementRef = entry.
       endPoint1;
16  var endPoint2 : inputMappingModel!ModelElementRef = entry.
       endPoint2;
17  var definitions = entry.definitions;
18  var conditions = entry.conditions;
19
20  //Select the elements belong to the specified source model
21  var inModelElements = sourceModelElements.get(endPoint2.
       owner.name);
22
23  //select the elements from the source model
24  var inModelElements = sourceModelElements.select(el | el.
       eContainer().eClass().name = sourceMetaModel.
       theMetaModel.name).asSet();
25
26  if (endPoint1.type = inputMappingModel!ModelElementType#
       Clazz and endPoint2.type = inputMappingModel!
       ModelElementType#Clazz) {
27    inputElements = inModelElements.select(el | el.eClass().
         name = endPoint2.theElement.name);
28
29    var outElements : Sequence = new Sequence;
30    for (element : Any in inputElements) {
31      var outElement = outputTraceabilityModel.createInstance(
           endPoint1.theElement.name);
32      ...
33    }
34    ...
35  }
```

Listing 5.5: Part of the dynamic model transformation to generate the TM

The next section (5.2.5) explains how the introduced traceability analysis language is defined and implemented in the prototype. Additionally, it demonstrates the execution semantics of the language in detail.

5.2.5 Traceability Analysis Language

As explained in Section 4.5.1, the traceability analysis language (TAL) is a textual language, which is also bound to the TIM (TIM-specific), to express traceability analyses at traceability abstraction level.

In this implementation, the TAL is defined on top of EOL which means that each statement is transformed into an EOL program to be executed on the TM. The generated EOL program takes TM as input, runs the analysis,

and produces the output result model. This way, the TAL would be compatible with the implementation infrastructure with which we have worked through out the research. However, it also allows us to exploit existing model management tools which already operate on modelling and metamodelling standards (such as Ecore).

The grammar of the TAL have been explained with Xtext which allows us to describe the concrete syntax and how it is mapped to the semantic. Additionally, it generate the basic infrastructure required to use the language to define traceability analyses.

As mentioned in Section 4.5.1, the TAL grammar is generated based on a given TIM. Basically, the TAL grammar consists of two parts:

1. *General part* which does not have any TIM-specific elements and covers the syntax for defining a query or a constraint, describing a condition using ForAll, Exist, and ExistOne, and explaining an expression in TAL. This part is provided in Appendix D.

2. *Project-specific part* which contains the *project-specific* terms which are defined in the TIM as TraceableElements and TraceLinks. These terms are used in the TAL as the valid values for the *TracableElement* and *TraceLink* rules in the grammar.

We also found out a number of other rules are required to describe more complex uses of these project-specific terms, which depends on the part of a statement in which they are used (e.g. elements as an operand which requires access to its feature). All the additional rules follow a set of recurring patterns for all elements, and, hence, can be generated automatically based on the TIM using M2T model transformations.

In this context, we have developed an EGL program which takes a TIM and generates the TIM-specific part of the TAL grammar which is thereafter appended to the general part of the grammar in order to have a complete grammar for a TIM-specific TAL. For each element in the TIM (subtypes of TraceableElement and TraceLink), we identified five grammar rules which have to be generated to reach to a precise and complete grammar. These rules define how to use the

1. alias for the element (specific syntax)

2. element in the query or constraint

3. features of the element

4. features of the element as an operand (used to navigate to the feature of the elements referred in a trace link)

5. element as an operand in an expression (normally, we need to access its features)

Listing 5.6 shows the core part of the EGL program in which the above grammar rules for each subtype of TraceableElement and TraceLink (defined in the TIM) are generated. The *elements*, in line 2, is a collection of TraceableElement and TraceLink. In lines 16-17, the first type of rules in above list is generated; the rule which defines the specific syntax to use an alias for an element.

```
1  [%
2  for (element : Any in elements)
3  {
4   //Element name
5   var elName = element.name;
6   //Determine the alias for the element
7   var elAlias = elName.toCharSequence()
8           .select(ch | ch.matches("[A-Z]"))
9           .concat();
10
11  //Initialise required variables
12  ...
13  //Generating grammar rules
14 %]
15
16 terminal [%=elAlias%]ALIAS :
17  "[%=elAlias%]_"INT;
18
19 [%=elName%]:
20  name = '[%=elName%]' 'as ' alias = [%=elAlias%]ALIAS
21 ;
22 ...
23 }
```

Listing 5.6: Part of the EGL program to generate the TIM-specific part of the TAL grammar

Figure 5.8 shows an example of the grammar rules generated in the IADDS project for the traceable element UserStory and trace link Mitigate.

With the complete grammar for the analysis language, the Xtext language generator is executed which will derive the various language components. We mainly use the editor, because of syntax highlighting and syntactic validation, and the code generator stub (the Xtend class) which is used to translate TAL scripts into EOL. The code generator stub is called by the language compiler whenever a statement in the new language is created.

The code generator stub is a Xtend class which implements a *doGenerate* method. This method is the main method of the code generation and it is overrided to generate any arbitrary code in the desired format. Listing 5.7 shows our implementation of the *doGenerate* method to generate EOL code for TAL scripts.

```
1  override void doGenerate(Resource resource, IFileSystemAccess
       fsa) {
```

```
terminal USALIAS :
   "US_"INT;

UserStory:
   name = 'UserSroty' 'as ' alias = USALIAS
 ;

UserStoryFeature:
   'name' | 'description'
 ;

ExtUserStoryFeature:
   'name' | 'description'

 ;

UserStoryAsOperand:
   USALIAS'.'ExtUserStoryFeature
 ;


terminal MALIAS :
   "M_"INT;

Mitigate:
   name = 'Mitigate' 'as ' alias = MALIAS
 ;

MitigateFeature:
   'name' | 'description' | 'traceLinkEnd1' | 'traceLinkEnd2'
 ;

ExtMitigateFeature:
   'name' | 'description' | 'traceLinkEnd1' | 'traceLinkEnd2'
   | 'traceLinkEnd1.'ExtSafetyStoryFeature
   | 'traceLinkEnd2.'ExtHazardFeature
 ;

MitigateAsOperand:
   MALIAS'.'ExtMitigateFeature
 ;
```

Figure 5.8: TAL grammar rules for UserStory and Mitigate

```
2   for (q: resource.allContents.toIterable.filter(typeof(Query)
        )) {
3     fsa.generateFile(q.QName +"_Query.eol", q.compile);
4   }
5
6   for (c: resource.allContents.toIterable.filter(typeof(
        Constraint))) {
7    fsa.generateFile(c.CName +"_Constraint.eol", c.compile);
8    }
9  }
```

Listing 5.7: The *doGenerate* method in the Xtend Class

First of all, the contents of the *resource* are filtered down to queries and constraints. Then, for each query and constraint, an EOL file (with *.eol* extension) is generated and the output of compiling Query and Constraint

(calling function *compile* on the query or the constraint) is written into the generated file. The function *compile* implements the code generator and defines the template code for the input type (e.g. query), which is implemented with specific syntax (EOL code in our implementation).

Listing 5.8 shows part of the function *compile* which generates the EOL code for a Query. In line 1, the code generator function *compile* is defined for type Query. The tag pair « » is used to delimit a dynamic section of the Xtend code. The contents of dynamic sections are executed and used to control the text that is generated, such as explaining conditional statements and calling the other functions to generate required text for the nested elements. In lines 11-13, if the Query has any input parameter the function *compile* for each parameter is called (shown in lines 36-41). For each input parameter, a list of all the model elements of specified type is provided to users to select one of them which is then assigned to a variable in the context of the EOL. In line 15, the code generator calls function *compileQFunc* for the *query* element in a Query which generates the template code for the QueryFunction (defined in the TAL grammar). All the other texts (outside of « and ») are directly written into the output.

```
1  def compile (Query q) {
2  '''
3   "**********************".println();
4   "Query: <<q.QName>> .... ".println();
5   "**********************".println();
6
7   //output model
8   var resultModel = new outputModel!QueryResult;
9   resultModel.queryName = queryName;
10
11  <<IF q.params != null && !q.params.isEmpty()>>
12    <<FOR p: q.params>>
13      <<p.compile>>
14
15  result = <<q.query.compileQFunc>>;
16
17  //build the output model
18  if (result.isEmpty()) {
19   resultModel.overallResult = outputModel!Result#failed;
20   "Failed: No Element Found!".println();
21  }
22  else {
23   resultModel.overallResult = outputModel!Result#passed;
24   ("Passed: " + result.size() + " Element(s) Found.").println
         ();
25   for (r: Any in result) {
26    var resObj : outputModel!ResultObject = new outputModel!
          ResultObject;
27    resObj.object = r;
28    resultModel.objects.add(resObj);
29   }
```

```
30   }
31
32   resultModel.execStatus = outputModel!ExecutionStatus#
         successful;
33  '''
34  }
35
36  def compile (Parameter p) {
37    '''
38    //Input parameter: <<p.PName>> : <<p.PType>>
39    var <<p.PName>> = System.user.choose("Choose Parameter: <<p
          .PName>>", inputModel!<<p.PType>>.all());
40    '''
41  }
```

Listing 5.8: Code template for the Query

The template code for all of the elements in the language have been implemented in the Xtend class similarly to the one defined for the Query element.

Accordingly, when a trace query or constraint is described an EOL file is generated automatically. Listing 5.9 shows the generated EOL for the trace query defined in Section 4.5.1.3. The EOL script creates a model of type QueryResult (line 9), named resultModel, and initialises its name to the name of the given trace query (line 10). The core part of the EOL is the statement executed to query the input model and collect the desired elements (line 13). The input model is called *inputModel* which refers to the traceability model and defines in the run configuration for the EOL. Finally, based on the result of the main statement, the found elements are added to the resultModel (the *for* statement in lines 23-27), the overalResult, and the execStatus of the model are assigned to the right value (lines 21 and 30 respectively).

```
1  "*******************************".println();
2  "Query: unMitigatedHazards ....  ".println();
3  "*******************************".println();
4
5  //temporary result
6  var result : Any;
7
8  //output model
9  var resultModel = new outputModel!QueryResult;
10 resultModel.queryName = "unMitigatedHazards";
11 resultModel.execStatus = outputModel!ExecutionStatus#failed;
12
13 result = inputModel!Hazard.all().select(HZD | not (inputModel
       !SafetyStory.all().exists(SS | inputModel!Mitigate.all().
       exists(M | M.traceLinkEnd1.name = SS.name and M.
       traceLinkEnd2.name = HZD.name))));
14
15 //build the output model
```

```
16 if (result.isEmpty()) {
17  resultModel.overallResult = outputModel!Result#failed;
18  "Failed: No Element Found!".println();
19 }
20 else {
21  resultModel.overallResult = outputModel!Result#passed;
22  ("Passed: " + result.size() + " Elements Found.").println();
23  for (r: Any in result) {
24   var resObj : outputModel!ResultObject = new outputModel!
        ResultObject;
25   resObj.object = r;
26   resultModel.objects.add(resObj);
27  }
28 }
29
30 resultModel.execStatus = outputModel!ExecutionStatus#
       successful;
```

Listing 5.9: Generated EOL file for the trace query defined in Section 4.5.1.3

5.3 Chapter Summary

In this chapter, an overview of the prototype of the tooling support for the traceability solution was provided. The implementation enables the author to use and evaluate the proposed approach. Throughout this chapter, important implementation decisions such as the choice of specific development environments or of specific programming and developing languages is discussed.

It is important to note that although EMF and Epsilon were chosen as the model management framework to implement the prototype for the tooling support, in principle, any other modelling framework and model management framework could have been used instead for implementation and this decision should not affect the genericity of the proposed approach. Of course, depending on the framework it might be needed to implement some parts from scratch, regarding the framework, or appropriate adjustments to the implementation should have been made. For example, using EWL facilitates the creation and definition of the mapping model as the model refers to many metamodels and element types and EWL wizards allows users to access the element types and assign them easily and correctly. In this respect, if we had chosen another framework which does not support update transformations in small explicitly (similar to Epsilon with EWL), we would have had to find and implement other methods to support users in this step.

To conclude, the main concept of the proposed approach (i.e. the fact that requirements traceability is a multi-domain concern and traceability solutions are required to work with various, heterogeneous models) should be generic and in principle applicable no matter which modelling technologies

are used to implement the tooling to support the approach. In the next chapter, a complex and large case study, using the implemented prototype, is provided. The case study helps us to evaluate the validity of the hypothesis and the tool, in addition to the requirements defined in Section 3.4.

6

Case Study

The EUR RVSM Programme

This chapter presents a case study of applying the introduced approach in a safety-critical project. The case study is large due to the size of the project and the number of the traceability-related metamodels and models as well as the nature of the traceability relationships. First, a brief introduction to the project is provided, which focuses on safety issues. Then, the application of the proposed approach in a traceability scenario is demonstrated.

6.1 The Programme: Introduction

The European Reduced Vertical Separation Minimum (RVSM) Programme [EUROCONTROL, 1999a][1] provides six additional cruising levels to air traffic in the airspace of 39 RVSM States. This facilitates the task of Air Traffic Services (ATS) in maintaining a safe, orderly and expeditious flow of traffic and results in increased capacity of the Air Traffic Management (ATM) system and reduced inflight delays and fuel economies for the users.

The RVSM Programme requires specific training for aircrew and Air Traffic Control (ATC) staff which was performed prior to the start of RVSM operations. The Programme also requires ATC equipment and procedures to be modified according to specific Programme requirements prior to the start of RVSM operations.

The Programme involves activities from a wide range of stakeholders and

[1] The resources for the project, which have been used in this case study, are no longer available officially.

covers different areas including the followings. Definitions have been taken from EUR RVSM Master Plan [EUROCONTROL, 1999b].

Safety. The introduction of RVSM must be achieved in conjunction with a thorough assessment of the safety implications of this change, the establishment of clear safety objectives, and safety evaluations showing the attainment of these objectives, before and after RVSM introduction.

Airspace Aspects. The definition of the extent of the European RVSM area has been based on the operational requirement for a homogeneous area without significant gaps. Within RVSM airspace, sectorisation and ATS routes will need to be reviewed in the context of the availability of the additional RVSM Flight Levels.

ATC Procedures. ATC Operational Procedures for the European RVSM airspace will need to be developed and implemented, including Flight Planning Procedures, Contingency Procedures, Transition Procedures, Procedures for handling non-RVSM State aircraft. These procedures will be reflected in the ATC Manual for Reduced Vertical Separation Minimum (RVSM) in Europe. Additionally, ATC training syllabi will be developed to support RVSM ATC training by the ATS providers.

ATC Systems. In order to accommodate and support the provision of ATC in an RVSM environment, ATC systems will need modification. The modifications are related to the controller and ATC training simulators to accommodate new needs. Further, the Central Flow Management Unit (CFMU) will need to adapt the system for RVSM, including modifications to the Integrated Initial Flight Plan Processing System (IFPS).

Aircraft Requirements. For operations in RVSM airspace, flights are required to be RVSM approved regarding the RVSM requirements. The RVSM requirements are reflected in the ATC Manual for RVSM in Europe, as basis for National regulation.

Monitoring. In line with International Civil Aviation Organisation (ICAO) Guidance Material, introducing RVSM should have appropriate monitoring in place to confirm that the height keeping performance requirements are being met. The monitoring programme requires the availability of height monitoring systems, both ground based (HMUs) and portable for on-board measurements (GMUs).

The identified activities have been organised into five main sub programmes and projects which have been further divided into a number of Work Packages (WPs) (tasks with objectives/deliverables).

1. Project P0: Programme Validation & Management. The main deliverables of P0 are the detailed RVSM Master Plan [EUROCONTROL, 1999b].

2. Sub Programme P1: Airspace User Preparation & Performance Verification. It ensures that the technical, operational and regulatory means will available for airspace users and States to enable RVSM approvals. P1 also assists and monitors the approval process. The monitoring programme will provide the technical data to confirm that safety objectives are met

3. Sub Programme P2: ATM Preparation. This will ensure all Air Traffic Service (ATS) provider units are prepared and ready for the introduction of RVSM. It identifies the tasks and deliverables required to make airspace changes including introducing RVSM related ATC procedures (ATC Manual for RVSM in Europe), modifying ATC systems (Agreed Operational Requirements for System Support, Agreed Interface Specifications, and software and procedures to fulfill the requirements), providing ATC training (ATC training syllabus), and resolving legal issues.

4. Sub Programme P3: RVSM Safety Assurance. It constitutes the safety assessments necessary prior to implementation, just after implementation, and at the end of the RVSM Programme to ensure that the agreed safety objectives are met. It includes the development of an agreed Safety Policy, a report on the RVSM Functional Hazard Assessment (FHA), a report on the Collision Risk Assessment (CRA), National Safety Plans, and Pre/Post-Implementation Safety Case.

5. Project P4: Awareness and Marketing. P4 caters development, delivery and coordination of an awareness programme through actions, products and packages supporting RVSM milestones.

Figure 6.1 gives an overview of the areas covered in the Programme and WPs with the associated deliverables/objectives. WPs have been categorised regarding the main areas and the general relationships between them are specified. An additional area, *Management*, has been defined (for our case study) which shows the managerial aspects of the Programme. Obviously, managerial activities (Project P0) and their outcomes determine other activities and monitor them. Accordingly, the deliverables of this area, including Programme Support Office (PSO), RVSM Master Plan, and RVSM Management Plan, are related to all other WPs, which is shown as relationships between this area and other areas rather than between WP (shown as dashed arrow). Additionally, some of the WPs involve more than one area and they are shown on the boundary between related areas.

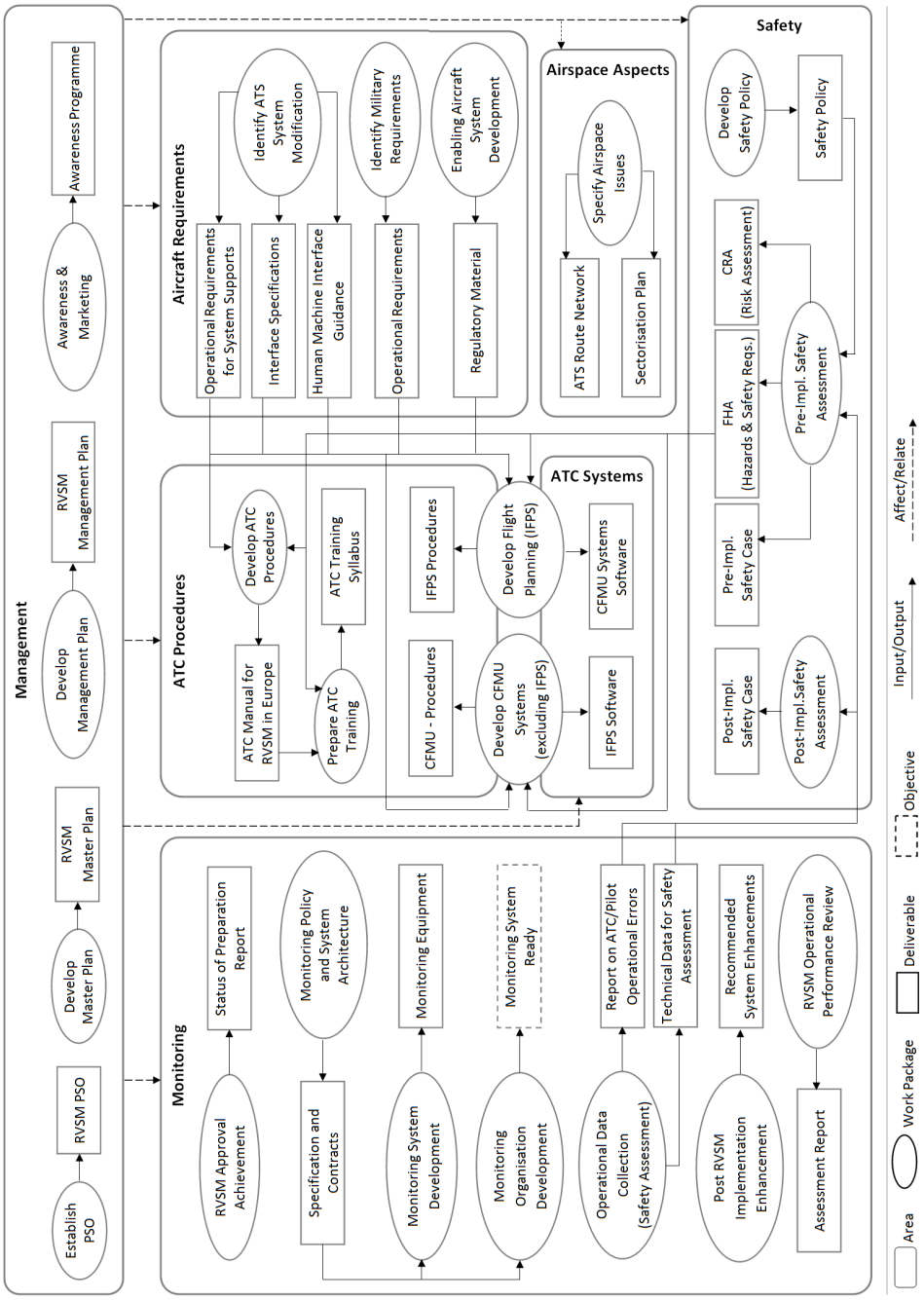


Figure 6.1: Conceptual overview of the EUR RVSM Programme

6.2 Traceability in the EUR RVSM

In this Programme, similarly to any safety-critical project, traceability is both a mandatory requirement of safety standards and essential in addressing the challenges and requirements inherent in the context of safety. In terms of the proposed approach in this thesis, both domain-specific traceability and multi-domain traceability are required. In particular, the project requires multi-domain traceability as it involves different areas (domains) and a wide range of activities. As shown in the Figure 6.1, activities (and so artefacts) in different areas are relevant to each other and, consequently, must trace to their related activities. For example, safety activities must trace to system modification activities, or trace to activities carried out to define and provide ATC procedures.

In many domains, traceability is supported in the scope of that domain (e.g. with domain-specific traceability solutions) or existing models implicitly provide traceability-related information. For example, traceability between the specific requirements of a supporting system and system acceptance tests is provided in the context of the system development process. In the safety domain, the relationships between safety requirements, hazards, and system elements is completely and explicitly defined and provided. However, to have a comprehensive and integrated view of the interrelation between different domains and their artefacts, a multi-domain approach to traceability is required, beneficial, and useful.

On the other hand, the nature of the involved areas acknowledges and supports the motivation of the research presented in this thesis; this case fits well with the proposed multi-domain approach. The areas are completely different from each other. Activities in each domain are usually performed without significant knowledge of other domains with totally different teams. The interrelation between activities is limited to preparing specific artefact to be used in other activities which is somehow the only way to relate them. This is because it is not possible or feasible to carry out these activities through a single execution or operational environment (system) or seamlessly integrate them in order to provide a coherent and comprehensive view of the information in the project.

Nevertheless, the traceability in the safety domain also has similar characteristics to multi-domain traceability to some extent. Safety activities are usually carried out independently by teams specialised in particular area, though they all conceptually serve for a same purpose (safety). In the scope of each activity, the relationships between concepts are well-defined and provided completely. But, the relationship between them are defined implicitly or incompletely. Moreover, in some cases different vocabulary are used to represent same concepts which would result in later misunderstanding and inconsistency. For example, in the RVSM FHA, for each hazard a *safety objective* is defined which is referred to as a *safety integrity requirement* in

the context of the PSSA. The term 'safety objective' refers to a different concept in the RVSM Safety Policy.

In this case study, we are particularly interested in traceability required for or related to safety; traceability between safety artefacts (implicitly provided in the safety area) and traceability between safety activities and activities/artefacts in other areas. Accordingly, in the following, a brief description of safety activities and their relationships with other activities is provided, which is however limited to activities for which there are available and enough resources in public. In addition to safety activities and artefacts, two activities including 'modification of ATC equipments' and 'provision of ATC Manual' are introduced highlighting their relation with safety activities.

Available resources for this project are documents (texts) which generally describe accomplished activities and represent the input/output of them. The metamodels and models for EUR RVSM that are used in this thesis have been developed by the author specifically for the purposes of this case study, based on existing available documents.

6.3 Safety Activities

The EUR RVSM has to demonstrate to the international aviation community that the Target Level of Safety (TLS) set out by ICAO [ICAO, 2002] for the vertical collision risk will not be exceeded in the European RVSM Airspace.

6.3.1 Safety Policy

The EUR RVSM Safety Policy [EUROCONTROL, 2000] has been developed to meet the requirements of ICAO standard. The Safety Policy is described through a set of safety statements based on which the EUR RVSM safety objectives are defined. It is the function of these objectives to ensure that the safety statements have been complied with.

Figure 6.2 shows the main concepts involved in the Safety Policy in the form of a metamodel for a model representing the Safety Policy.

6.3.2 Functional Hazard Assessment (FHA)

A detailed FHA is conducted to determine how safe the system would be by specifying the minimum requirements to be achieved by the system related to the identified hazards –called *safety objectives* in the context of FHA [EUROCONTROL, 2001b]. The identification is carried out in structured brainstorm sessions. For each session a number of scenarios were developed. The aim of scenarios is to ensure that all RVSM-related aspects of flight and air traffic control operations are critically examined.

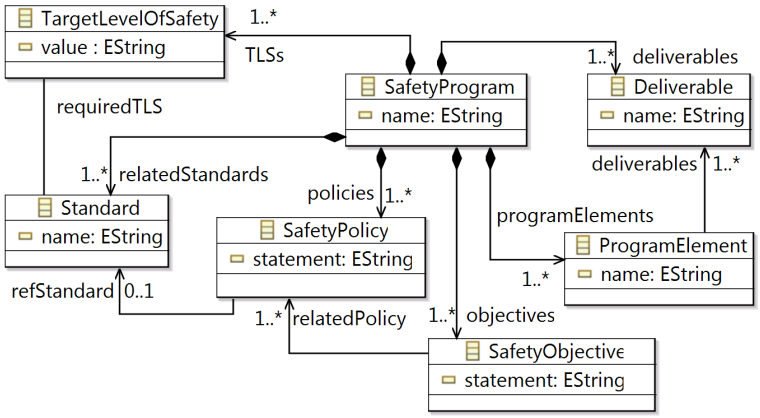


Figure 6.2: Concepts and their relationships in the RVSM Safety Policy

Once all possible hazards have been identified, each of them is being assessed to determine the consequences on operation and safety. After assessment of operational and safety consequences, the identified hazards are assessed in regard to *severity* and *probability*. The safety objective for each hazard is derived from the severity classification assigned to the hazard. Based on the specific severity classification, the safety objective specifies the maximum tolerable probability of the failure condition occurrence; that is "how safe the system needs to be".

Additionally, the mitigation to reduce the effect, and/or probability of occurrence, of each hazard is identified. Then (based on the mitigation), the identified hazards are grouped in two categories: safety-critical (not tolerable) and not safety-critical. Safety-critical hazards are those that do not achieve the related *safety objective* after RVSM mitigation (the identified mitigation is not sufficient).

Figure 6.3 shows the main concepts in the FHA and the relationships between them (the metamodel of the FHA model).

6.3.3 Preliminary System Safety Assessment (PSSA)

Throughout PSSA, further development is made in the context of safety and safety requirements are defined [EUROCONTROL, 2001c]. Initially, the safety objectives, which express seven *attributes* of the system, are translated into *High-level Safety Requirements* which are allocated to different RVSM *system elements* and subsequently broken down into *safety requirements* for them. System attributes include Function, Accuracy, Capacity, Overload Tolerance, Robustness, Reliability, Maintainability.

Additionally, the identified hazards are allocated to the system elements in order to ensure that they are appropriately addressed and their risk be-

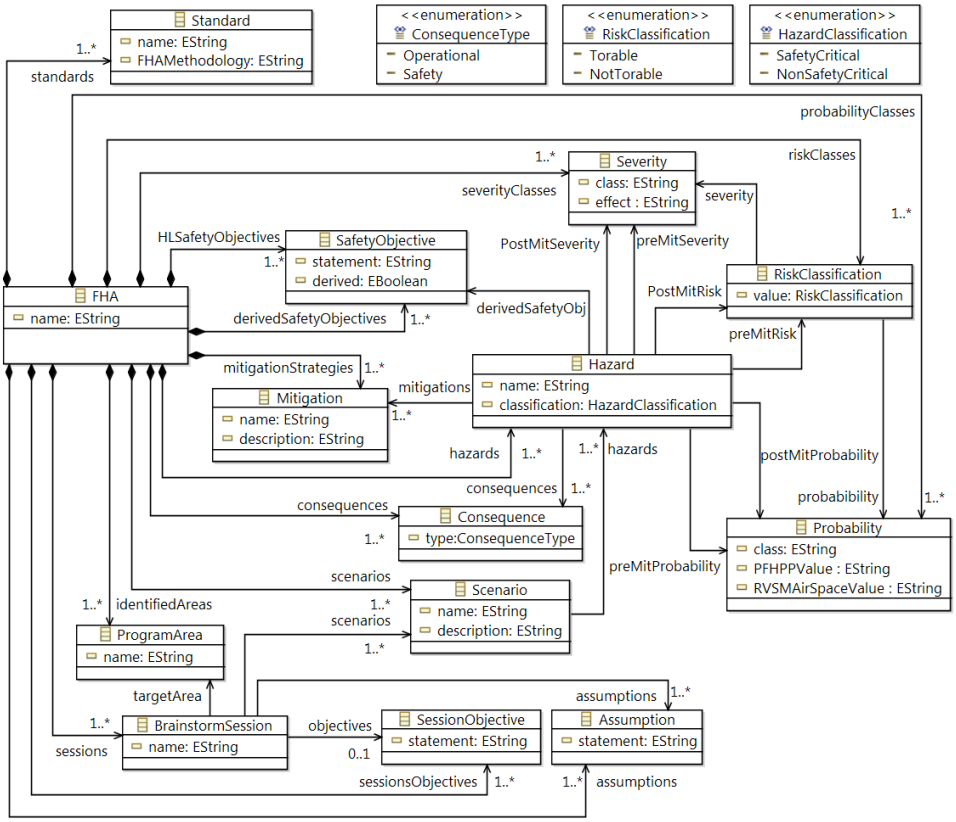


Figure 6.3: Concepts and their relationships in the FHA

ing managed. Doing so, *safety integrity requirements* are defined which are corresponding to the *safety objectives* for identified hazards in the FHA. Then, these requirements, with an indication of what (if any) mitigation is available, are allocated to already defined *safety requirements* and, consequently, to the related system element(s). Therefore, if any mitigation is available, an explicit (functional and detailed) *safety requirement* is derived for the relevant system element(s), in order to specify clearly the mitigation required. Otherwise, the *safety integrity requirement* from the FHA is directly allocated to the relevant system element(s), in order to limit the risk to a tolerable level.

Safety requirements (all types) also specify how they are realised and which *actions* are required for the individual stakeholders participating in EUR RVSM. *Actions* are particularly related to (affect) the tasks carried out under the Sub Programme P2. For example, they require specific training for the aircrew or ATC staff, particular requirements and specification for ATC systems, or specific ATC procedures to follow.

Figure 6.4 shows the main concepts in the PSSA and the relationships

between them (the metamodel of the model of the PSSA).

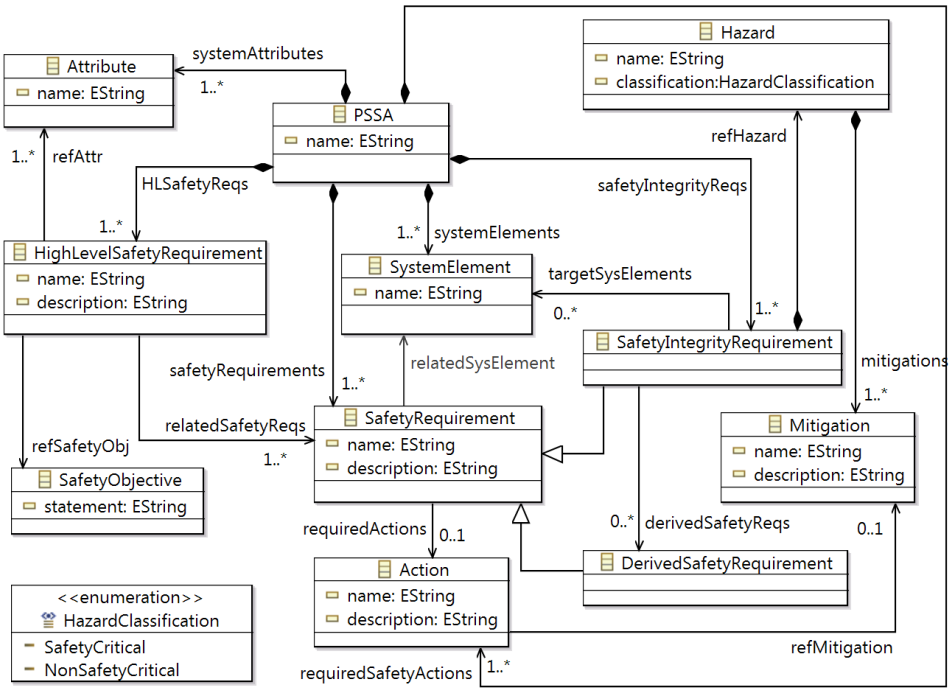


Figure 6.4: Concepts and their relationships in the PSSA

6.3.4 System Safety Assessment (SSA)

In SSA, the target system is checked against the FHA derived safety objectives or more accurately the PSSA derived safety requirements. The Pre-Implementation Safety Case (PISC) [EUROCONTROL, 2001c] establishes all the arguments and evidence necessary to demonstrate that the implementation of RVSM will be tolerably safe when assessed against the requirements of the EUR RVSM Safety Policy. The principal safety arguments are that

- the safety requirements fully address all the functionality, performance and integrity requirements necessary to ensure that the safety risks under RVSM will be tolerable.

- the RVSM Concept fully satisfy the RVSM safety requirements.

- the Implementation of the RVSM Concept fully satisfies the RVSM safety requirements.

– That the Switch-Over from the current vertical separation minima to the RVSM will not adversely affect the safety of the on-going air traffic operations.

Nevertheless, the evidence of the successful implementation of RVSM, to ensure that the agreed safety objectives are met, depends on the satisfactory completion of the actions specified by *safety requirements* in other areas.

Accordingly and considering the above mentioned safety arguments, it is required to be able to link safety requirements to other parts of the system and deliverables to be able to demonstrate that they are satisfied. For examples, if an action results in a specific requirement in one of the supporting systems, it has to be shown that the requirement is implemented in the system, for example by requirements traceability in the development of that system. In case an action requires a specific procedure or training, there should be correspondent parts (procedure or training material) in the ATC Manual or ATC training syllabus. These relationships have to be defined and available to be able to establish valid safety arguments. Therefore, traceability is essential to be able to comprehensively demonstrate that the EUR RVSM Programme is operationally safe.

6.4 Safety-Affected Activities

Safety activities affect many of the other activities and, therefore, safety artefacts are used in these activities, as the agreed requirements or specifications for that activity. In this section, two activities including 'modification of ATC equipments' and 'provision of ATC Manual' are introduced focusing on how they are affected by safety activities and how safety requirements are addressed in them.

6.4.1 Modification of ATC Equipment

ATC equipment is modified to provide the additional support to controllers regarding the specification of the RVSM and specific safety requirements. Generally, the RVSM Programme provides an agreed technical specifications for each equipment to the related stakeholders. For example, a document entitled 'RVSM Requirements for IFPS' was developed and accepted as the formal agreement between CFMU for the modification to the IFPS. In some cases, operational specifications for functionalities are also detailed in the RVSM ATC Manual.

Additionally, a series of Systems Acceptance Testing (SAT) are carried out which are conducted by a dedicated test team and the CFMU Quality Assurance section. Successful completion of the operational evaluation will therefore ensure that safety requirements have been satisfied.

Figure 6.5 shows the main concepts in modifying ATC equipments (based on the available resources) and their relationships.

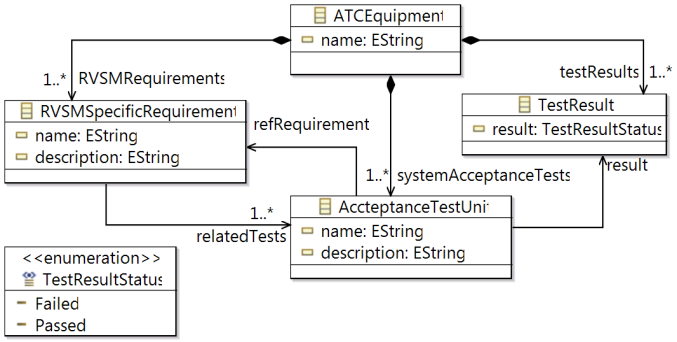


Figure 6.5: Concepts and their relationships in the ATC supporting system

6.4.2 Provision of ATC Procedures

RVSM is a change to a fundamental ATC separation. Its application needs to be supported by appropriate operational instructions and procedures which cover all possible scenarios encountered by controllers. These instructions and procedures are provided in the ATC Manual for RVSM in Europe [EUROCONTROL, 2001a]. The manual represents an operational reference document intended for the use of ATS personnel involved in the planning, implementation and application of a RVSM in Europe.

The safety requirements applicable to ATC procedures for RVSM are intended to ensure that the procedures cover the precise and clear description of the new vertical separation minimum, the criteria and requirements for its use, and enable the safe and expeditious flow of traffic. Accordingly, they are satisfied by procedures issued by EUROCONTROL and provided in the ATC Manual for RVSM.

Figure 6.6 shows the main concepts involved in and related to ATC procedures and the relationships between them. It mainly represents the ATC Manual concepts as the metamodel of a model representing the ATC Manual.

6.5 An Example Traceability Scenario

In this section, a multi-domain traceability scenario, in the context of safety assessment, is introduced and the application of the proposed approach to support the scenario is demonstrated and discussed.

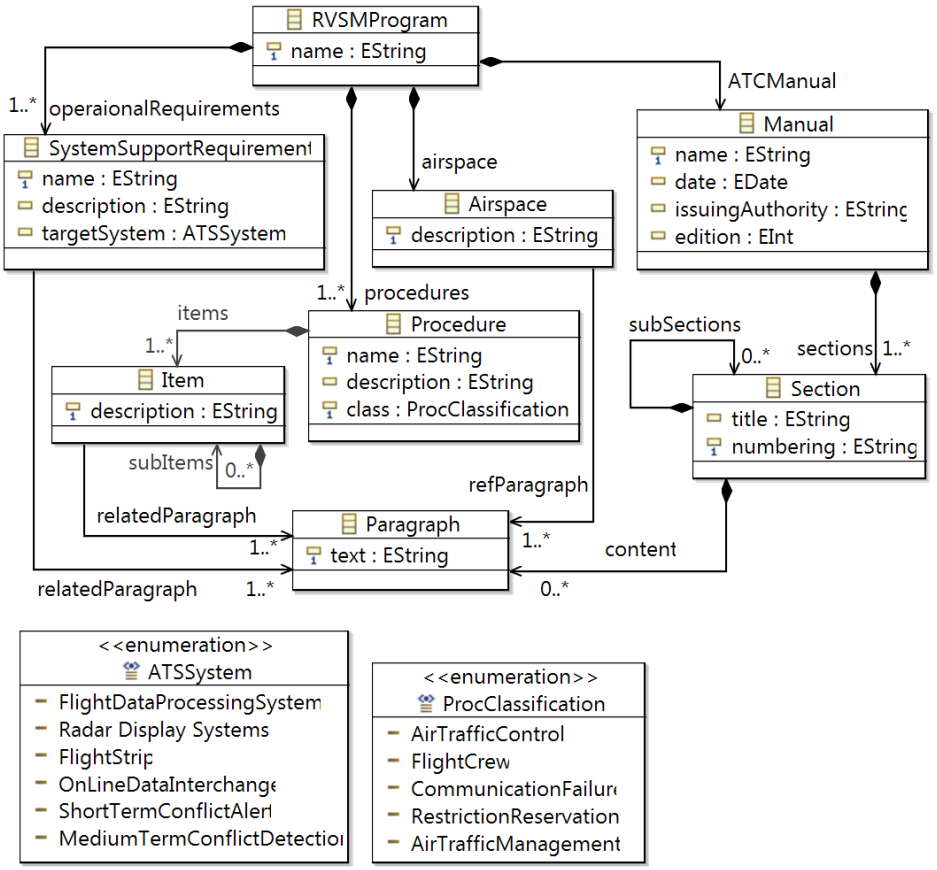


Figure 6.6: Concepts and their relationships in the ATC Manual

The scenario is defined as the following which, in particular, supports the first and the second safety arguments provided in the PISC.

'Safety requirements are valid, correct, and realised in Concept.'

The main concerns in this scenario are to show that specified safety requirements are

- *Valid:* they are either directly relevant or related to the RVSM safety objectives or defined to address identified hazards and mitigations.
- *Correct:* they covers all safety objectives and all identified hazards.
- *Realised in Concept:* they are explicitly addressed and covered in other related parts of the system.

A traceability solution is essential to address these issues; a solution which allows to define, create, and retrieve those traces representing the required relationships between objects or artefacts. As it can be seen, a part of the required information is in the safety domain and considers relationships between different safety concepts and models. The other part of the information requires relationships between safety models and models in other domains, such as supporting systems and ATC procedures. It might also require relations between other domains (excluding safety), for example, between system models and procedures.

6.5.1 The TIM

According to the proposed approach, traceability goals are determined and a GQM model is developed in order to define a TIM for the solution. The GQM model illustrates how the goals are refined into more detailed traceability questions and further into related concepts in the project. Figure 6.7 depicts the GQM model (the EMF model is provided in Appendix F.1).

Three traceability goals are defined corresponding to the aforementioned concerns for safety requirements. The first goal (G1) is to show that safety requirements are valid; for each safety requirement there is at least one safety objective or one hazard. The goal is refined into a question (Q1) which finds the related safety objective or hazard for a safety requirement. This question involves the following concepts: safety requirement, safety objective, high level safety requirement, and hazard.

Goal G2 demonstrates that all safety objectives and identified hazards are addressed by at least one safety requirement. Three traceability questions are defined in the context of this goal. The questions Q2 and Q3 find the safety requirements related to a safety objective or a hazard. The third question Q4 checks if the actions required by a safety requirement represents the mitigations identified for its related hazard. As shown in Figure 6.7, this question requires 'mitigation' and 'action' in addition to the above mentioned concepts.

Finally, the third goal (G3) focuses on the realisation of the safety requirement in the Programme. Doing so, three questions are defined. The question Q5 is defined to find the system element to which a safety requirement is allocated. The question Q6 determines more precisely how a given safety requirement is addressed. Considering the scope of this study, Q6 finds the operational requirements or specific procedures defined based on a safety requirement. Additionally, it is required to find out if these operational requirements are implemented in supporting systems and if these procedures are explicitly presented in the manual. Accordingly, the question Q7 focuses on the acceptance tests defined for operational requirements and the content of the ATC Manual provided for the RVSM Programme.

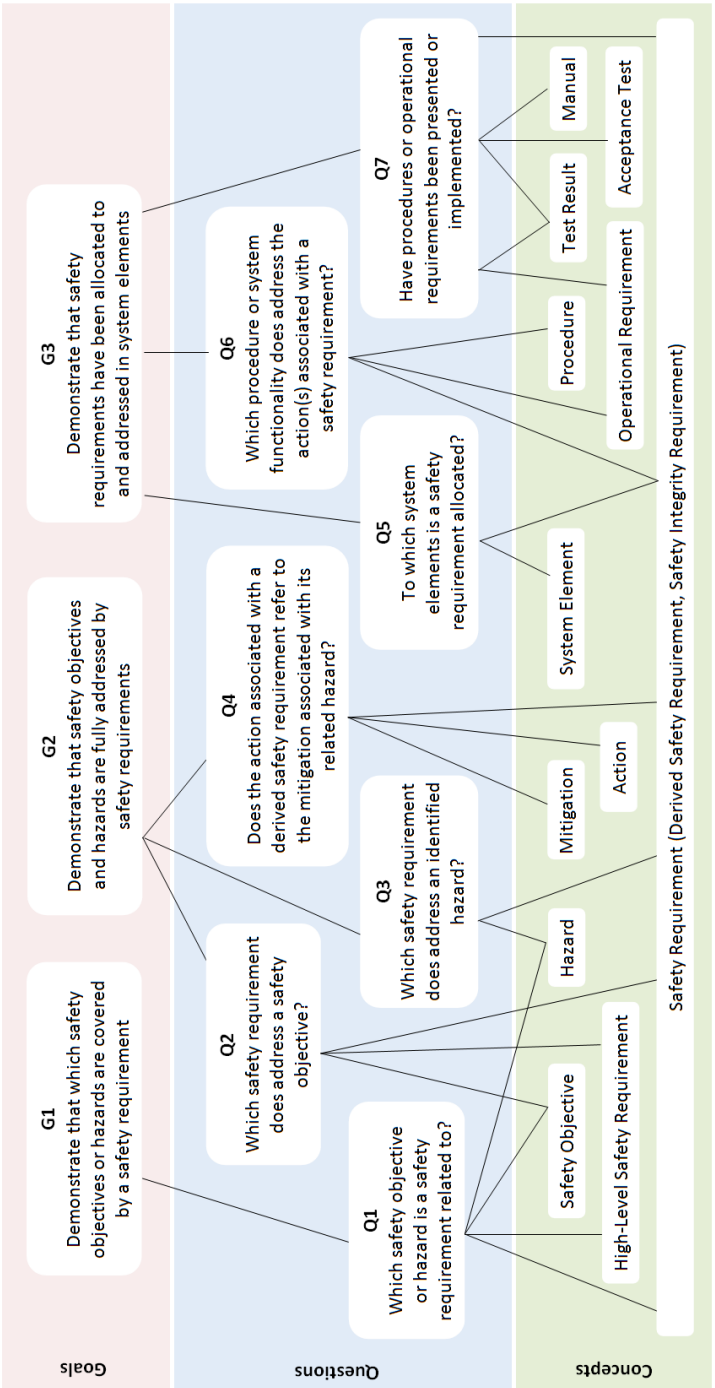


Figure 6.7: The GQM model for the example traceability scenario in RVSM

Based on the GQM model and the context of the project, the TIM is defined which is an extension to the Extended CoreTIM (introduced in Section 5.2.2), and, therefore, it is annotated with respect to the GQM model. The TIM is shown in Figure 6.8. *TraceableElements* are highlighted in yellow and the part of the TIM which have been introduced in the ExtendedTIM are coloured in gray in order to focus on the part which is specific to the RVSM (*TraceableElement* and *TraceLink* types).

Conceptually, the TIM represents an integrated and concise overview of the relationships between safety concepts and other concepts required to support specified traceability goals. The focus is on safety requirements and how they are related with other elements.

'Safety Requirements' are *allocated to* 'System Elements' and each of them defines specific 'Actions' to be done in the RVSM. The 'Safety Objectives' are *translated into* 'High Level Safety Requirements' which are then *decomposed to* 'Safety Requirements'. On the other hand, a number of 'Safety Requirements' are *derived from* those 'Safety Integrity Requirements' which are *defined for* NonSafetyCritical hazards. For SafetyCritical hazards, the defined 'Safety Integrity Requirements' is *directly allocated to* related 'System Elements' (shown as 'DirAllocatedTo' trace link type).

The actions *required by* derived safety requirements have to be defined based on the 'Mitigation' identified for the related 'Hazard' and, therefore, they *refer to* related mitigations. All the actions have to be explicitly considered in different areas of the RVSM. In the scope of this case study, we focus on how they are considered in the systems support or through provided procedures. Accordingly, depending on the type of an action, it has to be *addressed* by the 'Operational Requirements' defined for ATC equipments or *covered* by ATC 'Procedures' which are *presented in* the ATC Manual ('SectionOfManual'). For operational requirements, it is required to show that they are satisfied in the developed system which is possible through developing 'Acceptance Tests' for each requirement and recording their execution results ('Test Results').

Additionally, a traceability model, conforming to this TIM, has to satisfy more validity constraints, in addition to structural constraints which are described by a model validation language (EVL in our case study). These constraints could represent specific characteristics of a project or could be required based on the traceability goals. A traceability question, in the GQM model, may explain a constraint on particular trace link types.

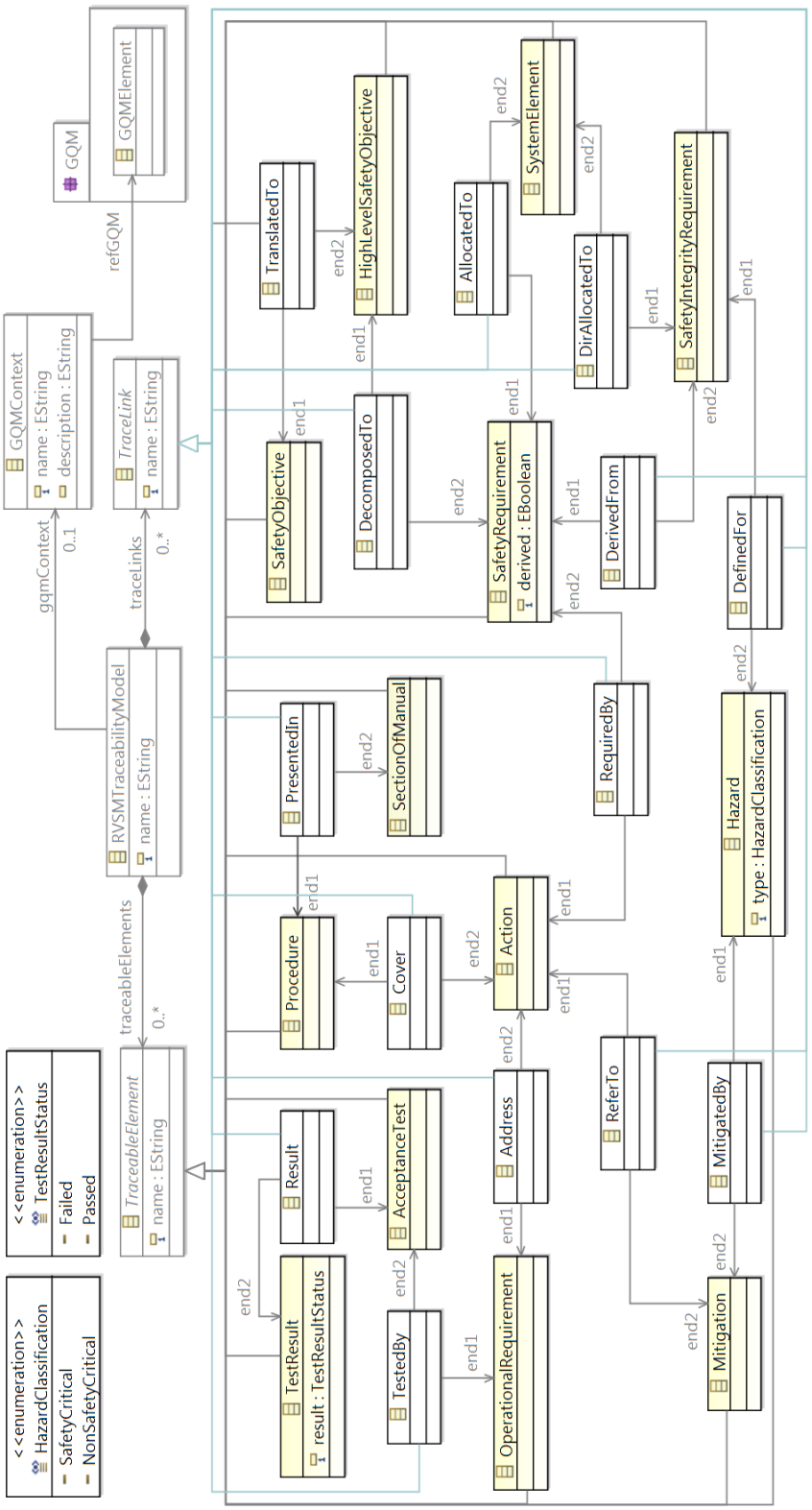


Figure 6.8: The TIM for the EUR RVSM Programme.

For example, the first validation constraint demonstrated in Listing 6.1 explains that a *derived* safety requirement is derived from a safety integrity requirement which its associated hazard is classified as NonSafetyCritical (there is no derived safety requirement for a SafetyCritical hazard). The second constraint in the Listing 6.1 has been defined based on the traceability question Q1, in the GQM model. It specifies that each safety requirement has to trace to a high level safety requirement or has to be derived from a safety integrity requirement. All the EVL constraints for the TIM are provided in Appendix F.2.

```
1  context SafetyRequirement {
2    guard : self.derived
3    constraint DerivedForNonSafetyCriticalHazard {
4    check :
5       var safetyIntegrityReq = DerivedFrom.all().selectOne( df
            | df.traceLinkEnd1 = self).traceLinkEnd2;
6       return not DefinedFor.all.exists(df | df.traceLinkEnd1 =
            safetyIntegrityReq and df.traceLinkEnd2.type =
           HazardClassification#SafetyCritical);
7
8    message : "Safety Requirement `" + self + "' traces to a
        SafetyCritical hazard";
9    }
10 }
11
12 context SafetyRequirement {
13   constraint NoDanglingSafetyRequirement{
14   check : DerivedFrom.all.exists(df | df.traceLinkEnd1 = self
15                     and
16                      df.traceLinkEnd2.isDefined())
17      or
18      DecomposedTo.all.exists(dt | dt.traceLinkEnd2 = self
19                     and
20                     dt.traceLinkEnd1.isDefined())
21
22   message : "All Safety Requirement have to be traced to a
        safety objective or a hazard";
23   }
24 }
```

Listing 6.1: Example EVL constraints in the TIM for RVSM

6.5.2 Traceability-related Information

Considering the models provided in the context of safety activities and activities affected by them (introduced in Section 6.3 and 6.4), all the traceable elements are defined and provided in these models. Trace link types which are defined between two elements within a same model can also be derived from that model and accordingly created in the TM. The main challenge is

about those trace link types which involve two elements from two models. According to the proposed approach, in such cases, partialTIMs are defined and partialTMs are generated.

As explained in Section 4.2.3, model elements in different domains are related to each other in two ways. Two elements are *equivalent* when they represent a same concept (common in two domains). In contrast, two elements are *relevant* when they do not represent an identical concept but they are related to each other in some way.

In the scope of this example traceability scenario, the following inter-domain relationships between model elements are identified which are required to be specified and provided.

- 'Safety Objective' in Safety Policy is equivalent with 'Safety Objective' in FHA when it is not a derived objective.
- derived 'Safety Objective' in FHA is equivalent with 'Safety Integrity Requirement' in PSSA.
- 'Safety Objective' in the Safety Policy is equivalent with 'Safety Objective' in the PSSA.
- 'Action' for derived safety requirements in PSSA is related to 'Mitigation' in FHA.
- 'RVSM Specific Requirement' in ATC systems supporting is related to 'Action' in PSSA.
- 'System Supporting Requirement' in ATC Manual is related to 'Action' in PSSA.
- ATC 'Procedure' in ATC Manual is related to 'Action' in PSSA.

Accordingly, five partialTIMs have to be defined to formally and explicitly specify these inter-domain relationships; partialTIMs between Safety Policy and FHA, between Safety Policy and PSSA, between FHA and PSSA, between PSSA and ATC Manual, and between PSSA and ATC systems supporting. Figure 6.9 shows the partialTIM between FHA and PSSA which includes an equivalent trace link type between 'SafetyObjective' and 'SafetyIntegrityRequirement' and a relationship between 'Mitigation' and 'Action'. Other parialTIMs are provided in Appendix F.3.

6.5.3 The Traceability Model

Based on the TIM, traceability-related information is located and the mapping model is defined to extract required information from existing models and create the TM. The TM provides a comprehensive and coherent view

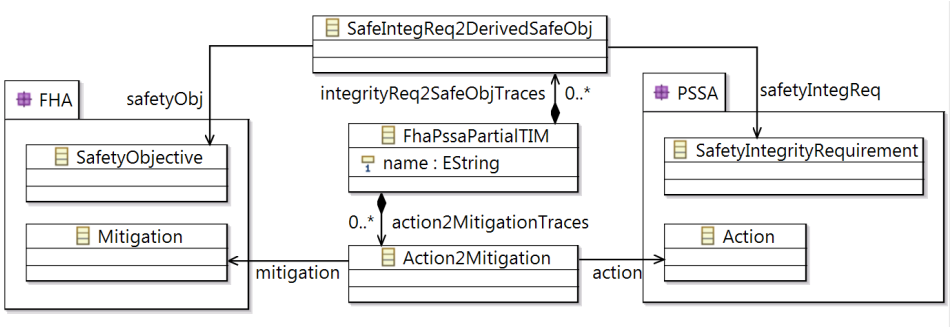


Figure 6.9: The partialTIM between FHA and PSSA

of the required (traceability) information over involved areas in the RVSM Programme to support the specified traceability goals.

Figure 6.10 shows part of the mapping model in which some of the mapping entries are defined. Mapping entries specify that each element in the TM is mapped to which element or association in other models. The mapping entries shown in the figure are limited to TraceableElements and describe how they are mapped to other model elements. The complete mapping model is shown in Appendix F.4.

6.5.4 Trace Queries

A TIM-specific traceability analysis language is generated based on the TIM defined for the example traceability scenario. The language is used to define arbitrary trace queries in the scope of the TIM. For example, the traceability questions identified in the GQM model demonstrate example queries or constraints to define. Each question might lead to more than one query and constraint. In the following, a number of queries and constraints defined with respect to these questions are explained as examples to show how the analysis language is used.

Q1: To which safety objective or hazard is a safety requirement related?

This question leads to two queries to find the safety objective and the hazard related to a safety requirement. These two queries support the question considering the validity constraint already defined on the TM: all safety requirements have to be related to a safety objective or a hazard. Listing 6.2 shows these queries. Safety requirements and safety objectives are traced to each other via high level safety requirements; a high level safety requirement is decomposed to a number of safety requirements while it is translated from a safety objective. Accordingly, this safety objective and these safety requirement are related. The first query explains this relationship with the

TAL and finds the related safety objective for a given safety requirement.

```
Query relatedSafetyObj4SafetyReq (SR : SafetyRequirement) {
  find SafetyObjective as SO where (
   exist HighLevelSafetyRequirement as HLSR where (
    SO TranslatedTo HLSR
    and
    HLSR DecomposedTo SR
   )
  )
}

Query relatedHazard4SafetyReq  (SR : SafetyRequirement) {
  find Hazard as HZD where (
   exist SafetyIntegrityRequirement as SIR where (
    SIR DefinedFor HZD
    and
    SR DerivedFrom SIR
   )
  )
}
```

Listing 6.2: Queries for the first traceability question (Q1)

Q4: Does the action associated with a derived safety requirement refer to the mitigation associated with its related hazard?
The constraint shown in Listing 6.3 addresses this question.

```
Constraint actionReferToMitigation (SR : SafetyRequirement)
    {
  forAll Action as ACT where (ACT RequiredBy SR) : (
   exist SafetyIntegrityRequirement as SIR where (
    SR DerivedFrom SIR
    and
    exist Hazard as HZD where (
     SIR DefinedFor HZD
     and
     exist Mitigation as MIT where (
      HZD MitigatedBy MIT
      and
      ACT ReferTo MIT
     )
    )
   )
  )
}
```

Listing 6.3: The constraint for the fourth traceability question (Q4)

Q6: Which procedure or system functionality does address the action(s) associated with a safety requirement?
This question decomposed to two queries to find the related procedures or operational requirements which address or cover the action required by a given safety requirement. These queries are shown in Listing 6.4.

```
Query relatedProcedure (SR : SafetyRequirement) {
  find Procedure as PROC where (
   exist Action as ACT where (
    PROC Cover ACT
    and
    ACT RequiredBy SR
   )
  )
}

Query relatedFunctionality (SR : SafetyRequirement) {
  find OperationalRequirement as OR where (
   exist Action as ACT where (
    OR Address ACT
    and
    ACT RequiredBy SR
   )
  )
}
```

Listing 6.4: Queries for the traceability question Q6

Q7: Have procedures or operational requirements been presented or implemented?
One of the queries in the context of this question is to find those safety requirement which are related to supporting systems but have not been implemented successfully (looking into test cases and test results). Listing 6.5 demonstrates this query.

```
Query unRealisedSafetyReqsInSystems {
  find SafetyRequirement as SR where (
   exist Action as ACT where (
    ACT RequiredBy SR
    and
    exist OperationalRequirement as OR where (
     OR Address ACT
     and
     not
     exist AcceptanceTest as AT where (
      OR TestedBy AT
      and
      exist TestResult as TR where (
       TR Result AT and TR.result=Passed
```

```
            )
          )
        )
      )
    )
  )
}
```

Listing 6.5: A query for the traceability question Q7

6.5.5 Discussion

As mentioned before, traceability is a mandatory requirement in the EUR RVSM Programme. The Programme involves a wide range of stakeholders from various areas and contains different kinds of activities. In this context, most of the traceability scenarios involve different programme areas and consequently require relationships between them (their artefacts). In particular, multi-domain traceability is essential in the context of safety assessment to provide valid evidence for safety arguments.

Using the approach presented in this thesis, we can obtain a coherent view of the traceability information required in the context of the RVSM Programme. The approach allows us to collect traceability-related information from different areas in a systematic, semi-automatic way rather than manually in an ad hoc way.

In addition to the main benefit of this approach, we can identify incompleteness between artefacts. For example, in the safety domain, the hazards identified during the FHA are investigated in more detail in the PSSA. It is required to ensure that all identified hazards are covered in the PSSA and each covered hazard is an identified hazard (a valid hazard). Accordingly, considering the metamodels shown in Figure 6.3 and Figure 6.4, each 'Hazard' defined in the PSSA model has to trace to a 'Hazard' defined in a FHA model and vice versa. In the context of the developed traceability solution, this property can be expressed as a validation constraint on the partialTIM between PSSA and FHA (Figure 6.9).

Additionally, the developed solution, mostly partialTIMs between models, provides a way to deal with inconsistencies between them specially when they are involved in the traceability. For example, as mentioned before in Section 6.2, the term *safety objective* is used in FHA as the derived safety requirement for a hazard, while the term is used differently in the context of the RVSM Safety Policy. On the other hand, safety objectives defined in FHA are referred to as *safety integrity requirement* in PSSA. This inconsistency, in using the term *safety objective*, is managed within the traceability solution as it is involved in the traceability scenario. 'Safety Objective' is defined in the TIM and, in the mapping model, it is described that to which concept in other models it is mapped. Also, in the partialTIM between PSSA and FHA, it is specified that safety objectives in FHA are equivalent

to safety integrity requirements in PSSA. This way such inconsistencies are managed in a systematic way.

6.6 Chapter Summary

This chapter demonstrated the application of the approach presented in this thesis in a large safety-critical project. Following a brief introduction to the project and its activities, it discussed the role of traceability in the project and outlined why the proposed approach to traceability is helpful and beneficial in this case. A multi-domain traceability scenario, in the context of safety assessment, was defined and, according to the proposed approach, a traceability solution was developed to support the traceability goals defined in this case study. Main parts of the solution were considered and described in detail. Finally, Section 6.5.5 discussed the benefits of using the proposed approach to develop a multi-domain traceability solution.

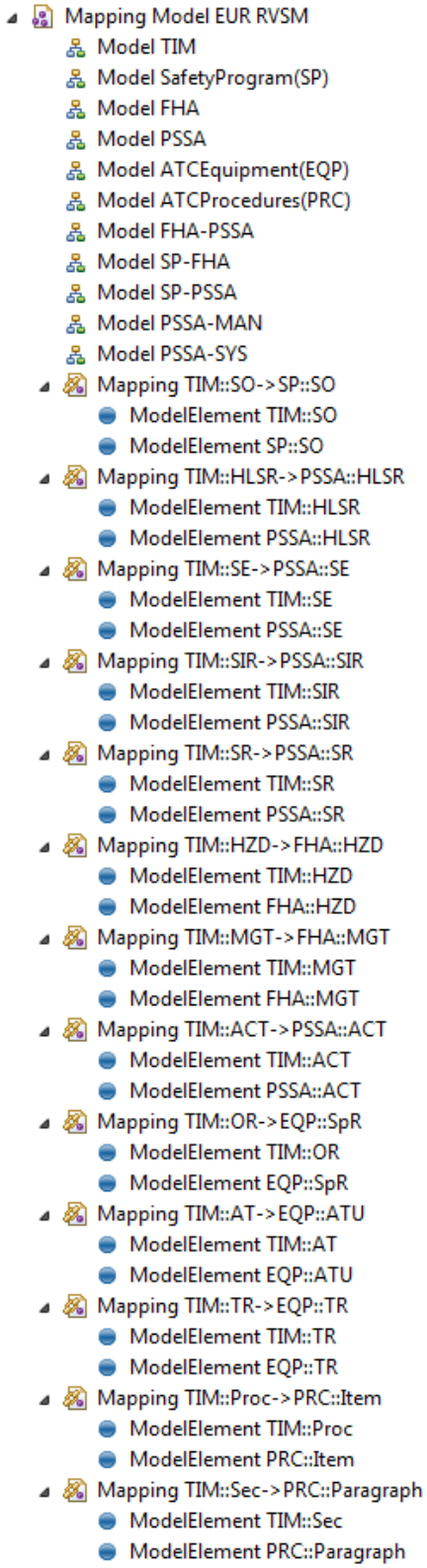


Figure 6.10: Part of the mapping model for RVSM case study

7

Evaluation and Discussion

This chapter evaluates the approach presented and explained in this thesis with respect to the research hypothesis outlined in Section 1.2. The evaluation explores whether the proposed tasks and structures are effective in developing a multi-domain traceability solution. It demonstrates how the proposed approach fulfils the defined requirements for such solutions and addresses identified challenges.

In addition to evaluating the proposed approach with respect to the defined requirements in Section 3.4, we focus on the validity and feasibility of this approach by analysing the prototype implementation, investigating the case study, and introducing publications directly related to this work. At the end, the limitations are outlined and we discuss why the approach has the potential to be useful in practice in spite of its limitations and the cost of developing a solution accordingly.

In Section 7.1, the approach is assessed against the identified requirements. Section 7.2 evaluates the EUR RVSM case study focusing on the feasibility and benefits of the approach. In Section 7.3, the papers representing the outcomes of this research are introduced. The limitations and shortcomings of the approach are identified and discussed in Section 7.4. Finally, the chapter is concluded with a discussion of using the proposed approach in practice (Section 7.5).

7.1 Evaluating the Approach Based on Requirements

This section focuses on the requirements defined for a multi-domain traceability solution and evaluate the work presented in this thesis regarding them. In the following sections, we discuss in detail how requirements are addressed and, hence, high-level goals are met by the proposed approach.

Each section focuses on a specific element, feature, or aspect of the proposed approach and highlights the requirements mainly addressed by that

element, feature, or aspect. In terms of the GSN, each section demonstrates the evidence (represented as solutions) to justify how goals (requirements) are addressed. Accordingly, the GSN diagram of requirements, provided in Figure 3.4, is completed by adding these sections as solutions. In order to have a expressive and clear GSN diagram, discussions under requirements are highlighted and named as **[Sx.x.x]** which allows us to refer to them in the diagram. The complete GSN diagram, which illustrates a summary of the requirements-based evaluation, is depicted in Figure 7.1 and Figure 7.2.

7.1.1 Scenario-Driven Approach

"R1.1 - Traceability is requirements-driven"

According to the approach, the solution is designed and developed based on the specific usage scenarios for traceability and the context of a project. Usage scenarios represent stakeholder needs and imply the required information to answer project-specific questions or undertake traceability-enabled activities. As mentioned in Chapter 4, scenarios act as the input of the introduced process to develop a solution. Therefore, the developed solution is requirements-driven.

7.1.2 Project-specific Traceability Information Model

"R1.1.1 - Solution is customised to accommodate requirements"

Using this approach, engineers can focus on traceability requirements for a specific project and develop a solution which is tailored to the requirements of a given project. A TIM is the core element of any traceability solution and solutions are built upon them. Accordingly, defining a customised TIM is a preliminary task in developing a customised traceability solution.

Section 4.1 describes how to define a project-specific TIM which is customised to the specific requirements and context of a project. In the proposed approach, a TIM is defined through three steps: 1. determine traceability goals, 2. identify related project concepts, and 3. define the TIM.

"St1.1.1.1 - Identify necessary information"

[S1.1.1.1] The first and the second steps provide a systematic way to identify necessary information to support traceability. Traceability goals are determined according to given scenarios which are used to identify required information to support traceability (Section 4.1.1). Then a goal-oriented approach, Goal-Question-Metric (GQM), is used to identify traceability-related concepts in the project. Using this approach, traceability goals are refined into lower level traceability questions (trace queries) which are associated with concrete concepts in the project (Section 4.1.2).

"St1.1.1.2 - Select the granularity of information"

[S1.1.1.2] In the third step, a project-specific TIM (traceability metamodel) is formally defined according to the identified concepts in previous steps. It specifies concepts and the relationships between them which are required to answer traceability questions and ultimately meet the goals. As explained in Section 4.1.3, the metamodel is implemented as a DSML which defines TraceableElements (concepts), TraceLinks (relationships), and validation constraints.

7.1.3 Models and Model Management Operations

"R3.1 - The approach is applicable to models: Create models and use model management operations (St3.1)"

[S3.1] The proposed approach works with models: domain-specific models are used to extract required information, inter-domain traceability models (partialTMs) are defined and created to record the relationships between domains, and traceability information is captured as a model which conforms to a customised traceability metamodel. Moreover, model management tasks and operations are used to create, modify and update the traceability-related models including utility models which are generated to support the proposed tasks and steps.

"R3.2 - Non-model artefacts: Systematic way to deal with non-model artefacts (St3.2)"

[S3.2] Although the approach works effectively with already available models that provide traceability information, it also considers non-model artefacts (though partially). The approach acknowledges that non-model artefacts are essential in an effective traceability solution, particularly in the context of multi-domain projects. As explained in Section 4.2, it identified different cases and discussed potentials to deal with non-model artefacts in each case, particularly using MDE techniques. For example, it was realised that in many cases existing non-model artefacts are structured (i.e. spreadsheets) and, therefore, a series of model transformation techniques (T2M, M2M, and M2T) can be used to transform them into models.

However, supporting an end-to-end, effective traceability solution in a project requires an extended approach to deal with all kinds of non-model artefacts specially when they are not structured and it is not practical and feasible to transform them into models, such as natural language descriptions or executable code.

"R3.3 - Different Metamodels: Systematic way to deal with different metamodels (St3.3)"

[S3.3] The approach is not restricted to specific metamodels, excluding the ones that are introduced and defined in this approach. Existing models (domain-specific or inter-domain models) are not required to conform to specific metamodels or to contain specific concepts.

"R3.4 - Different notations: Systematic way to deal with different notations (St3.4)"

[S3.4] The metamodelling language chosen to describe metamodels in the context of a solution is essential in order to be able to work with arbitrary metamodels. This is because the approach uses existing metamodels regarding the language by which they are described. So, a common metamodelling language should be used to describe all the metamodels in a soltuion. For example, in this thesis, we use EMF modelling framework and Ecore metamodelling language to define metamodels. So, Ecore metamodels are defined and used in different parts and steps of the approach. However, the approach does not mandate to use any specific metamodelling language which is dependent on the modelling framework chosen to implement a solution.

7.1.4 Domain-Specific Models

"R2.1 - Domain-specific information: Systematic way to reuse domain-specific information (St2.1)"

[S2.1] As explained in Section 4.2.1, once the TIM was defined, existing models (mainly domain models), which explicitly or implicitly support traceability, are analysed to find out how much of the required information is available and in which format. It is realised how the traceability model is related with domain models. The relationships are described in a mapping model which defines how each concept in a TIM (TraceableElement or TraceLink) is related to concepts in other models (Section 4.3.1). Then, as described in Section 4.3.2, with the use of model transformations and based on the mapping model, traceability-related information is extracted from domain-specific models and a TM for the project is generated. Accordingly, the approach uses existing domain-specific models in creating the traceability model and the model-driven approach enables us to work with arbitrary engineering models and effectively (re-)use them.

7.1.5 Inter-Domain Traceability Metamodels and Models

"R2.2 - Inter-domain information: Formally identify and capture inter-domain relationships (St2.2)"

[S2.2] The relationships between domains are important to accumulate the traceability information in order to support multi-domain traceability scenarios. Accordingly, a precise and formal definition of the relationships between domains is an essential information in a project, though it is usually

defined informally or incompletely. Section 4.2.3 demonstrated how to keep record of inter-domain trace links between two domain. Doing so, a traceability information model (partialTIM) is defined which explicitly specifies the required inter-domain trace link types between pairs of domains and. Then, traceability models (partialTMs) are created to record the trace links between domains. Accordingly, throughout the process to create a TM, inter-domain relationships are identified and captured in partialTMs which are used to create the TM, similarly to other models.

7.1.6 A Project-wide Traceability Model

Traceability information is captured and represented in a single project-wide traceability model. The model is built on top of the other models (domain models and inter-domain models) and generated automatically by the *dynamic model transformation* described in Section 4.3.2.

"R2.3 - Diverse information in different format: Systematic way to deal with the diversity in format and context (St2.3)"

[S2.3] A single TM provides a coherent view of the traceability information spread in different models, and unifies the way in which the traceability information can be used to perform traceability analyses. This way, users can focus on traceability and use the TM to perform traceability analyses regardless of the underlying information complexity, data structures, and information representation format of artefacts involved in supporting traceability.

"R2.4 - Separation of Concerns: Allow users to use domain-specific models and tools (St2.4)"

[S2.4] The way in which traceability information is captured and the project-wide TM is created allows the traceability users to use domain-specific tools and models separately while traceability information is defined and captured. Different artefacts from different domains that are being traced do not need to be combined (possibly artificially) in one overall description. Accordingly, the proposed approach supports separation of concerns and provides a transparent traceability solution to those who are not a user of traceability.

7.1.7 Traceability Activities

"R1.2 - Comprehensive solution: Consider all traceability activities or define extension mechanisms (St1.2)"

[S1.2] The proposed approach covers the *planning and managing traceability* tasks. As presented in Section 4.1.1, engineers determine their needs for traceability and design a solution accordingly. Section 4.1.3 demonstrated how a TIM is defined and represented as a DSML (Ecore metamodel). 'Assessing the solution' task is partially considered by defining validation constraints on a TIM which have to be correct all over the time.

In the scope of the approach, *trace creation* activity is realised by extracting traceability information from other models and generating a project-wide traceability models (Section 4.3.2). The approach does not consider the traceability solutions which might be used in each domain. It just uses the output of such solutions and other artefacts available in these domains. Thus, different trace creation techniques can be used in each domain to capture traceability information limited to a single domain.

The analysis language introduced in this thesis (Section 4.5.1) supports the *using traceability* activity, particularly for retrieving traces from a traceability model. In the context of the model-based approach, traceability information is represented as a model, which is a graph-based techniques for visualising traces (introduced in Section 2.1.3.5).

Traceability maintenance –and consequently flexibility– have not been fully covered in this thesis. The approach particularly considers maintaining the integrity of traceability model as other models change. Additionally, change and different types of change which might happen within the scope of the solution have been studied and identified, in Section 7.5. Then, and, accordingly, potential model-based strategies have been suggested to manage them, which certainly need more work to reach to a set of practical techniques to completely support traceability maintenance.

"R1.3 - Automated solution: Activities are performed automatically or semi-automatically (St1.3)"

[S1.3] In terms of traceability, two activities are subject to automation: trace creation and trace maintenance. In the context of the proposed approach, a dynamic model transformation is developed to extract traceability information from existing models and, accordingly, create the traceability model automatically, which is explained in Section 4.3.2. In this way, the TM can be automatically regenerated and updated whenever any of the source models change. As mentioned above, traceability maintenance is not completely covered in this work. However, MDE provides promising techniques to support (semi-)automated methods to keep the solution relevant and up to date (discussed later in Section 7.5.3).

"St1.1.1.3 - Customised traceability process"

[S1.1.1.3] As mentioned above, this approach does not constraint other project activities and does not require any specific order or schedule for

them. This is because it is loosely coupled with them and only use the output of these activities. Various traceability solutions could be used in each domain to capture traceability information limited to a single domain, in addition to the domain-specific activities carried out in each domain. This way allows engineers to easily customise the traceability process and integrate it with the other activities performed in a project.

7.1.8 Tooling Support

"R1.4 - Tool Support: Provide required tools (St1.4)"

[S1.4] The proposed approach provides a strategy to design and develop a traceability solution with the use of MDE techniques and tools. It demonstrates how modelling and consequently MDE framework help to develop a practical solution and implement a tool to carry out traceability activities. The proposed approach is not restricted to a particular model management framework. It introduces structures and operations which can be implemented with any framework. The prototype implementation (Section 5.2) shows that how the required tooling could be implemented with EMF and Epsilon, as an example modelling framework and platform. Nevertheless, it also demonstrates that it is feasible to implement a tooling support for a traceability solution developed based on this approach.

"R1.5 - Interoperable solution: Support interoperability between different tools (St1.5)"

[S1.5] The proposed approach is model-based. It mainly focuses on metamodels and works with them. As mentioned in Section 2.2.2, metamodels facilitate model interchange [Gitzel and Korthaus, 2004] and, consequently, interoperability between modelling tools. Specifying metamodels with a common modelling language ensures consistency in the way in which modelling constructs are specified and supports the construction of interoperable MDE tools. Therefore, a model-based tool supports interoperability between modelling tools which are used by users in different tools. For example, considering the prototype, the proposed structures and model management tasks are interoperable with the Eclipse Modeling Framework, which is arguably the most widely used contemporary MDE modelling framework.

On the other hand, as discussed above, the proposed approach allows users to use the tools that they normally use in their domain. The approach uses the artefacts generated within these tools. If these artefacts are models they can be used directly within the solution. Otherwise, specific techniques need to be provided to transform these artefacts into models. This case was considered in Section 4.2 where we discussed how the approach deals with non-model artefacts. Therefore, considering that there are several types of model in different domains with which the approach can be work, the

proposed approach supports interoperability between its supporting tools and other tools in a project.

"St1.1.1.4 - Configurable Tools: Tooling is configured to support stakeholder requirements and their processes"

[**S1.1.1.4**] The presented approach is requirements-driven and, so, the supporting tool is totally implemented based on specified requirements for traceability. On the other hand, as explained above, a solution and consequently its supporting tool can be customised and configured to stakeholders needs and usual processes.

7.1.9 A Model-Based Solution for Multi-Domain Traceability

According to all the above discussions, the proposed approach demonstrates a strategy to build a project-specific, multi-domain traceability solution. It introduces detailed steps and dedicated structures to design, define, and implement a solution using modelling principles and techniques to carry out related activities and generate related artefacts (represented as models). Therefore, the approach addresses the three top-level goals including providing a traceability solution (G1), supporting multi-domain traceability scenarios (G2), and providing a model-based approach to build a solution and carry out activities (G3).

Figure 7.1 and Figure 7.2 summarises the requirements-based evaluation in a GSN diagram which is an extension to the GSN diagram provided in Figure 3.4 (a larger view of the diagrams is provided in Figure C.2 and Figure C.3). As mentioned at the beginning, the above discussions in each section are considered as solutions in the diagram, except the discussion in Section 7.1.1 which is defined as *context* (C1.1) for the requirement 'R1.1 - Requirements-driven'.

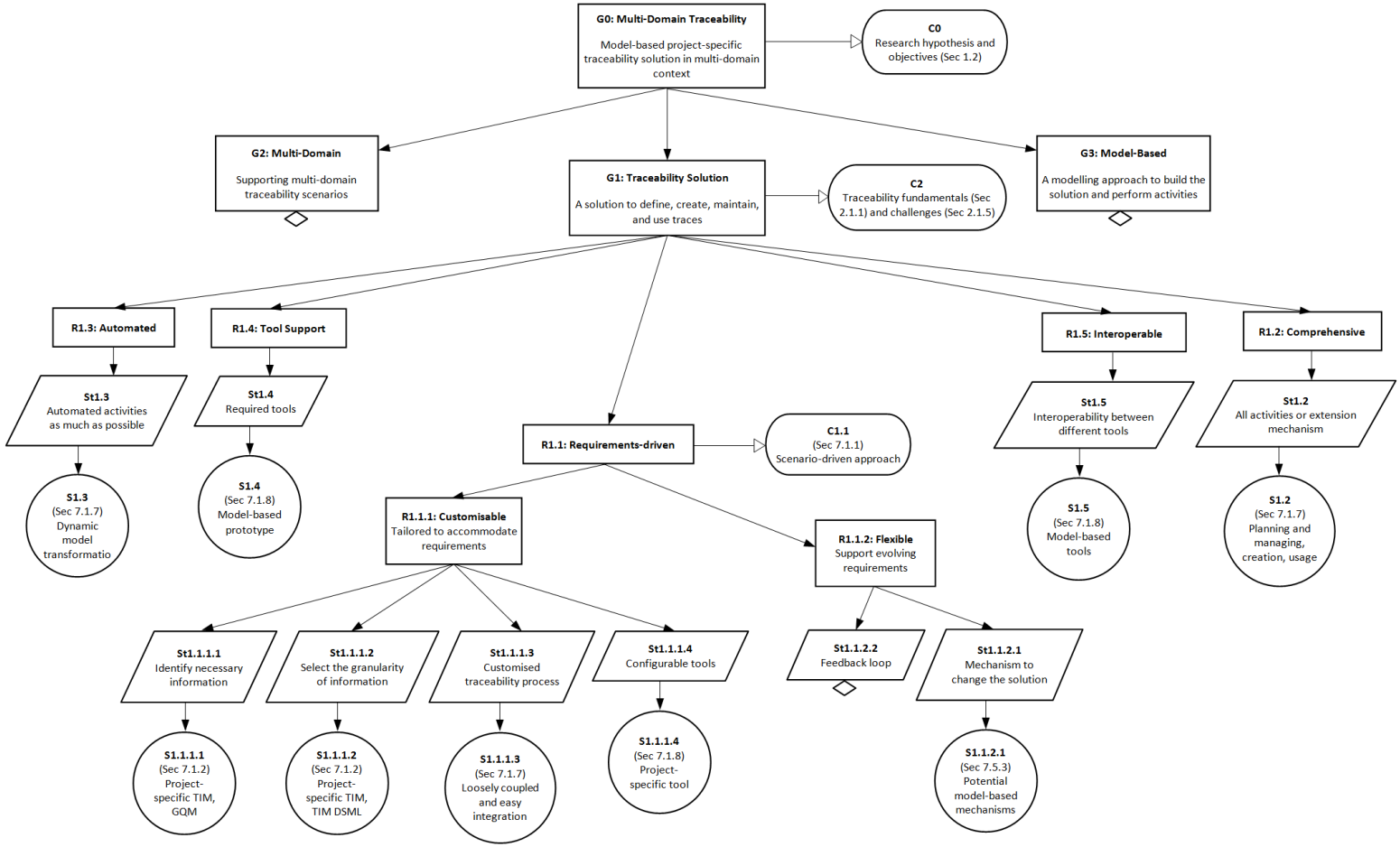


Figure 7.1: GSN diagram summarising the requirements-based evaluation for goal G1

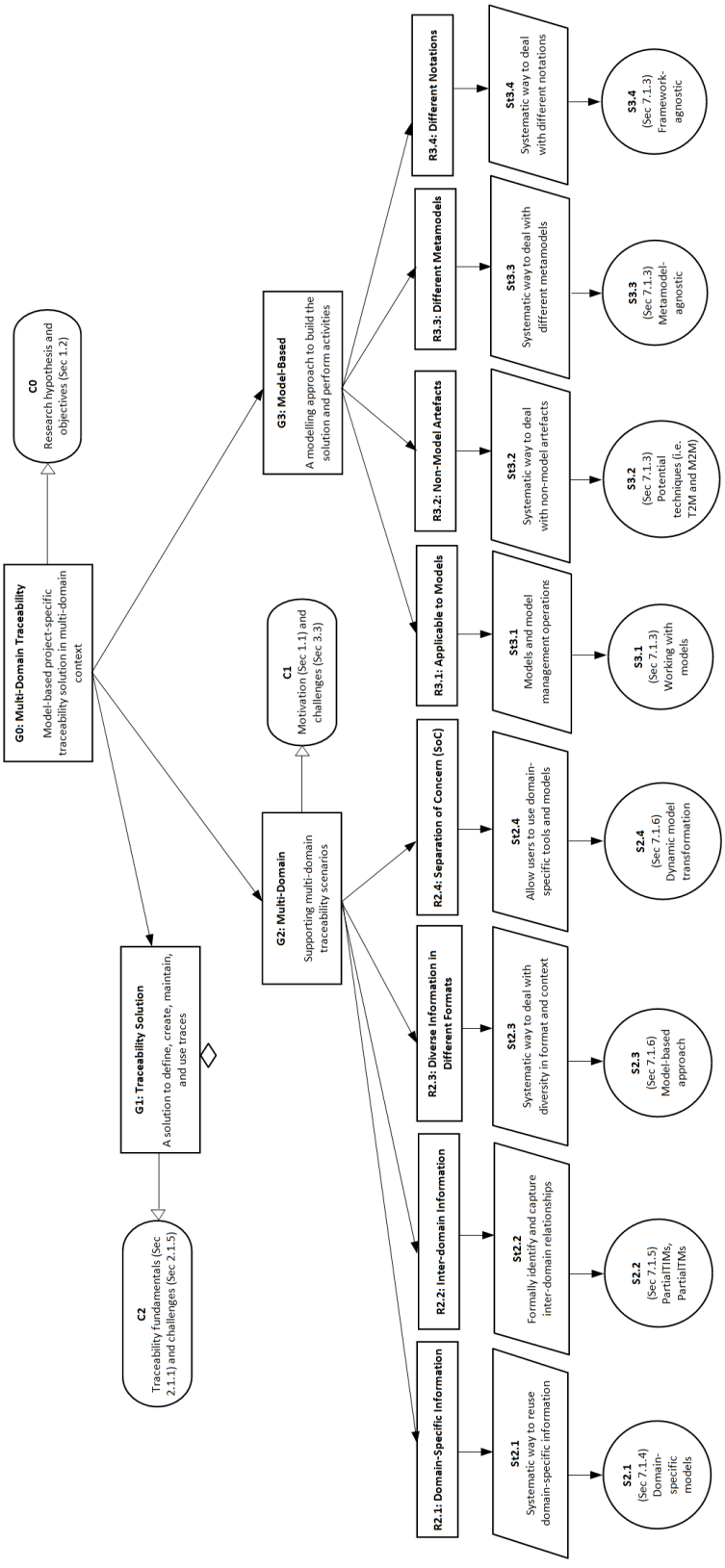


Figure 7.2: GSN diagram summarising the requirements-based evaluation for goals G2 and G3

7.2 The Case Study: EUR RVSM

In Chapter 6, the proposed approach was applied in a large-scale safety-critical project, EUR RVSM Programme. The case study explored the extent to which the proposed approach is beneficial, particularly in terms of multi-domain traceability scenarios.

In the case study, we defined an example traceability scenario in the context of safety assessment which covers different types of traceability. Then, using the proposed approach, we developed a traceability solution which supports specified goals.

The case study acknowledged that a multi-domain approach to traceability is essential in situations which involve wide range of stakeholders and various activities. We also found out that the approach can be generally helpful for any situation in which different activities are carried out even if they are all in a same domain. Usually, activities are performed independently, to some extent, and they are not (can not be) integrated properly (for any reason). Accordingly, the approach works well when there are a number of models and a concise view of the traceability information, which is implicitly or partially provided in these models, is required or recommended.

Additionally, the case study demonstrated that it is feasible to develop a traceability solution according to the proposed approach, various kinds of traceability usage scenarios can be supported with the approach, and finally why the benefits outweigh the cost of developing such solution. The case study also highlighted a number of limitations and potential improvements such as more automation support in building the infrastructure and additional features or capabilities to describe complex analysis scripts in the TAL.

7.3 Publications

Peer review is an important means of evaluation of the validity of an approach. Accordingly, publication in academic journals, and at international conferences and workshops ensure that our work is reviewed by experts, and is well-established and communicated in our field of research. The results of this research have been presented in a number of academic papers in journals, international conferences/workshops which are listed below.

– Masoumeh Taromirad, Nicholas D. Matragkas, and Richard F. Paige, *Towards Multi-Domain Traceability: a Model-Driven Approach*, submitted and under review in Journal of Software and Systems Modelling, Springer, 2014.

  This paper motivates multi-domain traceability and demonstrates how MDE principles and techniques could help to address its challenges.

Then, it provides a model-driven approach to define, design, and develop a complete multi-domain traceability solution.

– Masoumeh Taromirad, Nicholas D. Matragkas, and Richard F. Paige, *Towards a Multi-Domain Model-Driven Traceability Approach*, In Proc. of the 7^{th} International Workshop on Multi-Paradigm Modeling (MPM '13), Miami, Florida, USA, September 2013.

  In this paper, the concept of multi-domain traceability is introduced and a model-based approach to develop a multi-domain traceability solution is provided.

– Masoumeh Taromirad and Richard F. Paige, *Agile Requirements Traceability Using Domain-Specific Modelling Languages*, In Proc. of the 2^{nd} International Extreme Modeling Workshop (XM '12), Innsbruck, Austria, October 2012.

  This paper discusses how domain-specific modelling languages helps to provide light-weight traceability through defining traceability metamodels with respect to a project or case.

7.4 Limitations and Shortcomings

In this section, the main limitations and shortcomings of the proposed approach are outlined and discussed.

7.4.1 Limited Support of Non-Model Artefacts

The approach proposed in this thesis is applicable to models and focuses on defining and providing traceability between models. However, even in a MDE environment, other kinds of artefacts, such as informal, natural language descriptions of requirements, spreadsheets, or executable source code, are created and used during a development process. An end-to-end traceability solution requires to consider such artefacts as they are essential in providing required information.

Although non-model artefacts are partially considered, the suggested ways to deal with them are abstract and limited to cases in which it is feasible to use model transformations to transform these artefacts to models. There are situations in which non-model artefacts are not structured or it is not feasible to transform them into models, such as natural language descriptions or executable code. Therefore, in order to be able to provide a complete traceability solution, we need to extend our approach to consider all kinds of non-model artefacts and effectively deal with them. This would need to be done on a case-by-case basis, though we expect patterns will eventually be identified.

7.4.2 Partial Traceability Maintenance

Traceability maintenance refers to those activities associated with updating pre-existing traces as changes are made to the traced artefacts and the traceability evolves, and creating new traces where needed to keep the traceability relevant and up to date. Additionally, traceability maintenance is required following changes to the requirements and constraints that drive the overarching traceability strategies, which consequently results in change in the traceability solution and the traceability information [Gotel et al., 2012b].

A practical approach has to provide specific methods or techniques to support this activity or specify how existing practices for maintaining the integrity of the links could be applied within the approach. The way in which the TM is created allows users to regenerate the model whenever any of the related model change. Accordingly, the approach supports maintaining the integrity of the TM in the scope of the developed solution.

On the other hand, the approach needs to consider change and evolution in other parts of a solution (e.g. in traceability goals) and their effects on the other traceability-related models and discuss how to manage such changes.

Accordingly, the approach proposed in this thesis does not completely support traceability maintenance. However, Section 7.5.3 investigate change and evolution in detail. Different change scenarios, which might happen in a solution, are identified and discussed how they could be managed especially with the use of MDE techniques.

7.5 Discussion: Solution in Practice

The multi-domain traceability solution presented in this thesis provides a comprehensive and coherent view of the traceability information usually scattered across the project's domains heterogeneously. Particularly, the systematic way to collect traceability-related information and record it, using MDE tools, automates the creation and maintenance of a project-level traceability model. Additionally, the approach leads to a better way to manage the inevitable coupling and redundancy between traceability models and other models in the project. Moreover, a project-specific solution allows traceability users to focus on project's specific traceability goals and build an effective traceability solution.

However, there are still a number of concerns with the proposed approach which need to be considered and discussed to demonstrate why the approach could be used and be beneficial, in addition to identified limitations and shortcomings. The following sections elaborate on these issues.

7.5.1 Cost/Benefit

As mentioned previously, to our knowledge, there is no similar approach considering traceability as a multi-domain concern. Current research mostly focuses on traceability in a single context or, at most, similar contexts in which it is feasible to integrate them together for example by using a particular family of tools. The proposed approach allows users to benefit from a comprehensive solution which particularly operates across multiple domains and supports multi-domain traceability scenarios. Using this approach, users can express and answer questions (trace queries) across different domains regardless how the involved domains interact and the underlying information is provided. Supporting such scenarios is not easily possible with current tracing approaches and need more effort to achieve.

Preparing an MDE environment with the approach presented in this thesis requires additional effort and cost. Throughout the running example and the case study, we found out that the additional cost and expected effort is reasonable in comparison to normal activities in a project. For example, even for our case study, in which there were no models to be used and all the metamodels and models defined and generated from scratch, we built the solution in about two weeks, which is acceptable in the context of a large project. Moreover, this time could be considerably less if models and metamodels are already available. Moreover, the extra effort is largely required while a traceability solution is initially built. This is because although the solution might change, the main part of the solution remains unchanged. In the initial steps in building a solution, engineers and domain experts (traceability users) work together: domain experts specify the usage scenarios and goals, engineers define a TIM accordingly, and domain experts assist engineers to identify source models and the relationship between them to create the mapping model.

Finally, in situations where traceability is a mandatory requirement, it is required to establish an environment that capture the necessary traceability relationships [Pinheiro, 2003] and the benefits of traceability outweigh its costs [Cleland-Huang, 2006].

7.5.2 Non-model Artefacts

The approach works effectively when models have been already created and provided in a project. But, other kinds of artefacts, such as informal, natural language descriptions of requirements or spreadsheets, are created which have to be considered in a practical traceability solution.

Fundamentally, throughout our research, we found out that there are several types of model in different domains (e.g., requirements models, safety analysis models) which can provide substantial traceability-related information and, therefore, make the model-based approach applicable and benefi-

cial in many projects and situations.

On the other hand, as discussed in Section 4.2, in many cases different model transformation techniques (M2M, M2T, T2M) can be applied to transform non-model artefacts into models. Particularly when such artefacts are almost structured. For example, spreadsheets, as a common format for documentation, can be transformed into models using user-defined T2M transformations or available tools and techniques specifically developed for this purpose.

7.5.3 Change and Evolution

One of the main concerns with traceability implementations and tools is managing *change* which is formally considered in the context of traceability maintenance activities. Additionally, evolution is an identified concern in a MDE environment [Rose, 2011]; the evolution of models, modelling languages and other MDE development artefacts must be managed properly.

Accordingly, we study change in the context of our approach, from both traceability and MDE perspectives, and identify different change and evolution scenarios. Then, we discuss how these changes could be managed, specially with the use of MDE techniques, and suggest potential approaches or methods to maintain the solution relevant and updated.

7.5.3.1 Change Scenarios

Based on an analysis of the artefacts or elements defined, created, and used in the context of the proposed traceability solution and how they are related to each other, we identify the following scenarios which have to be managed to keep the solution relevant and up to date:

- **Change in the TIM.** A TIM will change when the requirements and constraints for the traceability change. This is because a TIM is defined based on these requirements and project contextual constraints in order to support traceability goals. On the other hand, a TIM is the core of any traceability solution and solutions are developed around it. Accordingly, a change in a TIM will affect the solution and traceability-related models in different ways and levels.

- **Change in metamodels.** This kind of change commonly happens and is extensively studied in the context of MDE. In this thesis, we particularly need to consider changes in domain-specific metamodels with respect to their effects on the models specifically generated for the solution. Basically, domain-specific models have to be updated due to change in their metamodels. But, in the context of this work, such changes are important from another perspective as they have

additional effects: the inter-domain metamodels/models (partialTIMs/partialTMs) and the mapping model are consequently subject to change. This is because they work with domain-specific metamodels and, hence, are related with them. We mainly study such effects in Section 7.5.3.3.

– **Change in the models.** This change is the one that is generally considered in the context of traceability maintenance; changes in the traced artefacts and the relationships between them which need to be managed and propagated correctly to maintain the integrity of the links. This change scenario has been discussed under maintaining traceability model in Section 4.4.

As mentioned before in Section 2.1.3.4, there are two main approaches to maintenance: proactive (or continues) and reactive (or on-demand). In the former approach, the changes are constantly monitored and the update of impacted trace links immediately follow changes to traced artefacts. In the latter approach, a dedicated and overall update of the trace set (in whole or in part), is done generally in response to some explicit trigger or preparation for an upcoming traceability use. We consider both types of approaches to maintenance in order to introduce potential practices to manage the aforementioned change scenarios.

7.5.3.2 Change in the TIM

This change mainly happens because of a change in traceability requirements (usage scenarios), constraints, and goals which drive the proposed strategy. This change would be the most expensive type of change in a traceability solution due to the role of a TIM in any solution. A TIM is the core element of a solution and its change would affect possibly all of the other elements of a traceability solution. To analyse the impact of a change in a TIM on the other parts, we go through the activities performed after defining a TIM and discuss in each case what should be done and how.

1. The first step after defining a TIM is investigating the available information to find the required traceability-related data. So, following a change in the TIM, the available information should be revisited. This job can be done manually or semi-automatically (i.e. through model comparison) as mentioned in Section 4.2.1.

2. Similarly to the main step, as the result of new investigation of available information, domain-specific and inter-domain metamodels/models might need to be changed and updated in order to reflect new requirements for the traceability-related data. Such changes, which mainly relate to making changes in metamodels and updating the correspondent models accordingly, are thoroughly investigated in MDE

domain, mostly in the context of model-metamodel co-evolution. In the next section (7.5.3.3), this type of change is discussed in more detail.

3. When the TIM, domain-specific and inter-domain models are changed, the mapping needs to be updated accordingly. Through several experiments in studying the effect of change in the traceability solution, we have identified the following cases in which the mapping model needs to be changed:
   - New mapping entry or equivalence entry is required to define a new relationship between the TIM and other metamodels. In this case the new entry is added to the model.
   - A model element type in a metamodel (either the TIM or a source metamodel), which has been referred by an entry (as 'ModelElementRef'), needs to be deleted. In this case, the entries which have references to the deleted type should be deleted from the mapping model. This can be done automatically by specifying required model migration strategies to update the mapping model consistently.
   - A model element type in a metamodel (either the TIM or a source metamodel), which had been referred by an entry (as 'ModelElementRef'), has to be renamed or moved. In this case, the entry might need to be updated to refer to the altered or new element. This update can be done automatically similarly to the previous case by describing particular model migration strategies to propagate the change.

4. The most important concern in managing change is to keep the TM up to date and valid. So, when the TIM changes, the TM has to be updated accordingly in order to conform to the TIM and represent valid trace links while the information evolves. In this case, the TM is regenerated by the dynamic model transformation with the updated inputs (TIM, domain-specific and inter-domain models, and mapping model).

5. Finally, the traceability analyses may need to be defined again, as they are defined in terms of TIM. Therefore, those trace queries and constraints which use a renamed or deleted model element type (in the TIM) should be rewritten, where as for the new model element types, new analyses are defined and expressed.

All of the above are the effects of a change in the TIM. As mentioned before, this change is the most expensive change as it leads to change in all

other elements of the solution. Considering the two approaches for traceability maintenance, those changes and steps which happen in metamodel level, such as updating a domain-specific metamodel and the mapping model, are performed and applied as the TIM changes (continuous traceability maintenance). But, the changes in the model level, including regenerating the TM, and updating the trace queries, are done whenever users need to use the traceability information or perform a traceability analysis (on-demand traceability maintenance).

7.5.3.3 Change in metamodels

This type of change would happen if there are some changes in system or software requirements. Additionally, a new traceability requirement might result in updates in metamodels. For example, when additional concepts are required to capture in order to support traceability goals. This way, as explained in Section 7.5.3.2, existing metamodels needs to be updated or new metamodels have to be created to define new concepts to be able to record them.

The first effect of change in metamodels is that their corresponding models needs to be updated to conform to the new metamodel. But, In the context of the developed solution, changes in domain-specific metamodels result in particular changes, rather than updating their related models. The mapping model is the main model which works with these metamodels. Therefore, it might need to be updated to reflect the changes in metamodels. As explained in the previous Section 7.5.3.2, a mapping model may have entries which refer to model element types which do not exist any more or have been renamed or moved. This way, the mapping model needs to be updated to show valid relationships between metamodels.

On the other hand, inter-domain metamodels (partialTIMs) are fundamentally linked with domain-specific metamodels. So, when one of these metamodel changes, related partialTIMs may need to be updated if they refer to the altered part of the source metamodels. Inter-domain models (pratialTMs) are accordingly required to be updated. This will need to deal with the model-metamodel co-evolution issues, in addition to identifying new trace links as discussed in Section 4.2.3. The TM has to be regenerated with the updated models, whenever users want to access traceability information and perform an analysis.

As mentioned before, such changes are an active research area in context of MDE [Mens and Demeyer, 2008]. There has been substantial research into model and metamodel change and evolution in MDE, which can be applied in these cases. Considering the classification of types of metamodel change presented in [Gruschko et al., 2007], some of the changes in a metamodel can be resolved in the model automatically, while there are some changes which needs human intervention to be resolved. Generally, researches pro-

pose different approaches for co-evolution of models with their respective metamodels. For example, [Gruschko et al., 2007] introduces an envisioned migration process focusing on the minimisation of manual effort, and [Rose, 2011] provides structures and processes for performing model-metamodel co-evolution. Using such approaches, the required migration strategies for the models can be defined and used to update the models according to the respective metamodels. However, updating the source models is out of the scope of this work. It is because we assume that source models are managed by users in each domain and the traceability solution just uses these models as its input just in the way they are provided.

Finally, there is an special issue with changes in metamodels: when a traceability-related concept, which has already been mapped to a concept in the TIM, is deleted from one of the metamodels. In this case, assuming that the TIM is the reference of the required traceability information to record and it is fixed, users need to find a way to deal with the missing information. Basically, they could follow what is done in the early phases of building the solution (Section 4.2). Also, they could inform the engineers of the related domain to consider the new situation, which might result in keeping the concept. These two approaches are based on the fact that the TIM is the reference and it is not supposed to be changed. However, there are situations in which the TIM has to be updated, though they happen rarely. Sometimes, such changes implicitly reflect a change in the context of a project and requirements for traceability which results in changes in the TIM and, consequently, in almost all other elements of the solution (the first scenario of change discussed in Section 7.5.3.2).

7.6 Chapter Summary

In this chapter, considering different evaluation methods, we demonstrated how the thesis proposal has been realised and the proposed approach addresses the identified challenges and requirements in the conxtet of this work. The evaluation of the proposed approach and the prototype implementation of the tooling support, against the specified requirements for a multi-domain traceability solution, showed that the approach meets almost all of the requirements specially those are related to the multi-domain nature of the solution. An overview of the case study illustrated the feasibility of the approach, in addition to the benefits achieved by using it, particularly in projects where providing traceability is a mandatory requirement. Additionally, publications (peer reviewed papers), representing outcomes of this research, supported the contribution and validity of the proposed approach. The evaluation led to identifying main limitations and shortcomings of the approach including limited support of non-model artefacts and supporting traceability maintenance activity and tasks. Finally, Section 7.5 discussed

the solution in practice with respect to the identified limitations and the cost of developing a multi-domain solution based on this approach. We demonstrated how they are addressed or could be managed and so the approach is beneficial.

8

Conclusion

This thesis has investigated traceability in multi-domain context because of its essential role in projects that span different engineering domains. In such projects, traceability is usually mandated and should be exploited and managed across the engineering lifecycle which is often required to capture the artefacts and trace links either within one or across many domains. However, there are several challenges that hinder the adoption of traceability in multi-domain projects, such as defining, identifying, and specifying relationships between heterogeneous models in different domains. The thesis has contributed to the research challenges and the research hypothesis stated in Section 1.2:

> *In many contexts, traceability is a multi-domain concern as it needs to operate across project's different domains. This thesis demonstrates that a modelling approach to develop a multi-domain traceability solution improves the precision and repeatability in such solutions. This will be via automating the challenging aspects of defining, creating, using, and maintaining heterogeneous traceability relationships between various domains in order to support multi-domain traceability scenarios.*

To explore the hypothesis, the following research objectives were defined:

1. Identify and analyse the concept of multi-domain traceability and its specific requirements and challenges.

2. Propose an approach to develop a multi-domain traceability solution which effectively deals with models across different domains.

3. Provide techniques to clearly identify requirements for traceability and develop a project-specific solution accordingly.

4. Support and automate the activity of maintaining the traceability solution to keep it relevant and effective over the time.

5. Provide semi-automatic techniques to specify and capture relationships (trace links) between multiple domains

The remainder of this chapter summarises the contributions of the thesis in relation to the thesis hypothesis and research objectives, and briefly describes the areas for further work which were identified during the course of this research.

8.1 Thesis Contributions

The main contributions of this thesis are summarised below.

- Motivating the concept of multi-domain traceability, identifying specific challenges in this context, and analysing existing literature on traceability accordingly, which consequently, led to a set of requirements for an effective multi-domain traceability solution.

- Demonstrating the process and structures to design and develop a model-based multi-domain traceability solution, which included a three-step method to define a project-specific TIM, defining and creating inter-domain models, locating related models, providing a systematic way to extract traceability-related information and create a traceability model, and a Traceability Analysis Language (TAL) for expressing traceability analyses.

- Application of the approach in a large safety-critical system which explored the extent to which the proposed approach is beneficial.

- Evaluating the approach with respect to the specified requirements for a practical traceability solution in multi-domain projects, which was illustrated using the Goal Structuring Notation (GSN) to give a concise view of the evaluation.

8.1.1 Traceability in Multi-Domain Context

The initial contribution of this thesis is introducing the concept of multi-domain traceability which was discussed in Section 1.1. Traceability is usually required throughout the project lifecycle, which in many cases extends across multiple engineering domains. Therefore, a traceability solution needs to operate across the project's different domains, effectively deals with various artefacts in different domains and relationships between them.

Chapter 3 investigated the existing work in the field of software traceability, provided in Chapter 2, and analysed the state of current traceability

literature in the context of multi-domain traceability. Accordingly, those challenges specifically existed because of the multi-domain nature of traceability were outlined and discussed. The investigation led to definition of a set of requirements for a multi-domain traceability solution which address challenges and characteristics of a general traceability solution, in addition to the identified challenges for multi-domain traceability.

On the other hand, as stated in the research hypothesis, this thesis proposed a modelling approach to develop a solution. Accordingly, the set of requirements also addresses issues related to model-based approaches in general (e.g. dealing with different metamodels) and such approaches in the context of traceability (e.g. dealing with non-model artefacts). The specified requirements are summarised in a GSN diagram, depicted in Figure 3.4

8.1.2 Model-Based Multi-Domain Traceability Solution

The main technical contribution of this thesis, presented in Chapter 4, is a model-based approach to develop a traceability solution which effectively operates across multiple domains. The proposed approach introduced a collection of tasks and structures which intend to address the challenges and requirements defined before. In the following the novel (the most important) parts of the proposed approach are briefly introduced in turn.

Usage Scenarios and Traceability Goals

Traceability is requirements-driven [Gotel et al., 2012a]. In terms of traceability, *usage scenarios* determine project-specific questions or traceability-enabled activities which are answered or supported by traceability information and, hence, specify requirements. Therefore, a traceability solution has to be designed and developed with respect to the specific requirements for traceability in a given project.

Accordingly, the proposed approach in this thesis is scenario-driven. Usage scenarios are the input of the introduced process which are then refined into lower-level (more detailed) *traceability goals*. In general, goals represent the ways in which a scenario supported.

Project-Specific TIM

A project-specific TIM, the core element of a traceability solution, is defined specifically for a project based on its characteristics and traceability goals. In the proposed approach, a TIM is defined through three steps: 1. determine traceability goals, 2. identify related project concepts, and 3. define the TIM.

First, traceability goals are determined based on the usage scenarios. Then, with the GQM, a goal-oriented approach, existing traceability-related

concepts and relationships between them are identified. These concept and relationships form the foundation of a suitable TIM. Finally, based on the previous tasks, a TIM is formally defined which specifies the entity types to be recorded and the relationship types between them. A TIM is developed as a domain-specific modelling language (DSML), the metamodel for the traceability model.

Inter-Domain Traceability Metamodels/Models

The relationships between domains are essential to accumulate the traceability information in order to support multi-domain traceability scenarios. Though, such relationships are usually defined informally or incompletely or not provided at all.

In the approach presented in this thesis, the relationships between each pairs of domains (inter-domain trace links) are captured in a special traceability model, a partialTM, which only involves two domains. For each partialTM, a traceability information model (partialTIMs) is defined which formally and explicitly defines the inter-domain trace link types between involved domains and the type of trace link ends for each type.

Mapping Model: Locating Related Models

Throughout our research, we observed that, in multi-domain traceability, most of the required traceability-related information is already available in other models and, so, could be extracted from them (reused) and put in a traceability model. Therefore, a traceability model is inherently a redundant model which provides a pervasive and coherent view of the other models from traceability point of view. This was also highlighted in the context of the example project (Section 1.1.1). However, in existing approaches which support project-specific traceability, TMs are created independently from other models. None of them has acknowledged to reuse existing models and specify this connection between a TM and other models.

Doing so, a *mapping model* is created which formally specifies how each concept (traceable element or trace link type) defined in a TIM relates to model elements in the other models. Generally, the model is similar to a Correspondence Model (CM) or Weaving Model (WM) in model composition [Bézivin et al., 2006]. The model is thereafter used to collect information from other models in order to automatically generate a project-wide TM.

A Project-Wide Traceability Model

Having defined the project-specific TIM and the mapping model,

traceability information can be collected and recorded. As [Mäder and Cleland-Huang, 2010] state, using a diverse set of traceability models is one of the main problems in working with traceability information. Accordingly, we suggested to create *a project-wide traceability model*. A single TM provides a coherent view of the traceability information and unifies the way in which the traceability information can be used. For example, it lets us define a Traceability Analysis Language to describe traceability analyses regardless of the diversity of the underlying models. This way, the model can be considered as a model built on top of the other models. A TM is generated automatically by a particular model transformation, called dynamic model transformation.

Dynamic Model Transformation

The TM is created and populated automatically by a domain-specific model transformation, so called *dynamic model transformation* which takes the mapping model and the other source models as input and produces the final TM as output. The transformation extracts traceability-related information from other models, including domain-specific models and partialTMs, based on the mapping model (illustrated in Figure 4.13).

The transformation is *dynamic* as it does not contain the low-level (executable) transformation rules, which explicitly define which model elements are transformed to which model elements. Rules are determined dynamically and executed at run-time. The transformation can be considered to be a higher-order transformation (HOT). However, in comparison to HOT which generates executable transformations rather than executing them, the dynamic transformation also performs the transformation and generates the target model –as a transformation engine.

Traceability Analysis Language

Generic query languages (e.g. SQL) and model management languages can be used to express traceability analyses, but these require knowledge of the underlying structures in which the traceability information is stored. Considering the context of this work, general-purpose model management languages (e.g. EOL) require user to explicitly specify how to retrieve particular elements from a models and generate an arbitrary output model. Whilst, traceability users are just interested in expressing a trace query or constraint regardless of how it is implemented and generates an output model.

Accordingly, a task-specific analysis language which enables users to explain traceability queries or constraints at the traceability abstraction level, so called Traceability Analysis Language (TAL) was defined. The TAL hides

the complexity of the underlying information and how it is stored and represented. Additionally, it is a TIM-specific language as it lets users express analyses by using the traceability domain terminology and project-specific terms (defined in the TIM).

The TAL is a simple textual language, similar to SQL, and allows users to describe queries (finding specific traceable elements) and constraints (check if the traceability model satisfies specific conditions). Each TAL script is executed on the project-wide TM and the result is presented in a simple model called Result Model.

8.1.3 Case Study and Evaluation

Initially, applying the proposed approach in a large-scale safety-critical project has explored the extent to which the proposed approach is beneficial and is practical (Chapter 6). The case study defined different types of usage scenarios and demonstrated the variety of cases that the approach could support.

Additionally, in Chapter 7 the approach introduced in this thesis has been evaluated with respect to the defined requirements for a multi-domain traceability solution and considering the peer-reviewed papers. In Section 7.1, a detailed evaluation of the approach was provided and illustrated the extent to which it satisfies the requirements, which is summarised in two complementary GSN diagrams (Figure 7.1 and Figure 7.2). In Section 7.3, the validity of the thesis proposal was considered with respect to the papers representing the outcomes of this research.

8.2 Future Work

The identified limitations and shortcomings of the approach (Section 7.4) and the discussion of using it in practice (Section 7.5) motivates several directions for future work following this work. On the other hand, throughout our research, we noticed some potential extensions which would enhance the proposed approach. In the following, these areas are described and, in case, initial work is outlined.

8.2.1 Non-Model Artefacts

As discussed in Section 7.4.1, non-model artefacts contains substantial information and are extensively used, even in a MDE environment. In this context, a comprehensive approach to traceability requires to consider these artefacts as they are essential in providing required information. It should clearly specifies in which way it deals with them. It could specify how models are traced to non-model artefacts and vice versa. However, in the approach presented in this thesis, non-model artefacts are considered as sources of

information and, therefore, are required to be represented in models to be able to extract required information from them to create a TM. Accordingly, in the context of this work, supporting non-model artefacts is about providing effective and practical mechanisms to represent non-model artefacts in models, in addition to the general challenge in MDE for tracing models to non-model artefacts. This way, required information can be extracted from these model similarly to other models.

Model transformation techniques can be applied to transform non-model artefacts, particularly when they are structured, into models. There are also different tools and techniques specifically developed to transform such artefacts into models (i.e. [Francis et al., 2013] to add spreadsheets into MDE context). However, there are situations in which non-model artefacts are not structured or it is not feasible to transform them into models. In such cases, depending the required information from given non-model artefact (i.e. its granularity and semantics), heuristic approaches, which are preferably repeatable, could be applied. An engineer could define a simple model which only represents traceability-related information provided in a given non-model artefact depending on the granularity of the information. For example, the model might have an element representing an analysis document as a whole and then she can use an existing approaches to trace models to text (i.e. offset/length-based solutions) to refer to specific parts in a document. Although this is a simple example, it shows potentials to identify recurring patterns to deal with non-model artefacts and provide low-level practices accordingly.

8.2.2 Traceability Maintenance

One of the main activities in providing traceability is maintenance. We identified that change is an important concern and more detailed and low-level practices are required to fully support traceability maintenance. The discussion in Section 7.5.3 identified possible approach to manage different change scenarios and demonstrated that how and in which ways existing model migration and co-evolution techniques could be helpful to effectively cope with them.

We also found out that it might be fruitful to adopt the state-based approach to traceability maintenance provided in [Drivalos-Matragkas et al., 2010] within the context of traceability metamodelling language introduced in [Drivalos et al., 2009]. Particularly, due to the potential to use the TML to define a project-specific TIM in our approach.

8.2.3 Semi-automated Mapping Model

As mentioned in Section 4.3.1, a mapping models is similar to a Correspondence Model (CM) in model composition [Bézivin et al., 2006]. A CM cap-

tures links between different models and is created with a match operation which is implemented with different procedures depending on the language used for composition. In AMW, a user interface and pluggable match algorithms are provided to create a CM. EML uses comparison rules (in ECL) to define a match operation, and in GGT, the expression of composition is created by a user interface.

Although mapping models and CMs are created for different purposes, existing approaches to create a CM suggest potential extensions to the proposed approach by adapting these techniques in this context, which would improve efficiency of defining a mapping model. Currently, this thesis has not prescribed a particular way to create a mapping model and focused on their usage in a solution. Therefore, in practice, a mapping model can be created differently depending on the tooling support (the modelling framework and model management platform). For example, in the prototype, a mapping model is created manually within a simple user interface (provided with EWL). However, considering the model composition with EML, comparison rules (ECL) could be used in creating a mapping model. ECL rules are defined to identify potential relationships between models which are then finalised by engineers and recorded in a mapping model.

8.2.4 Improved Traceability Analysis Language

The introduced traceability analysis language can be improved and enhanced with more complex constructs and functionality to allow users explain complicated queries. We also found out that ready-to-use templates or patterns for various usage scenarios would help users to work with traceability information in a better way. This is because templates and patterns are usually defined more efficiently, allow users to spend less time to describe a query, and also decrease errors.

Query optimisation is also a potential improvement. For example, as traceability models are usually large and contains many model elements, execution time of even simple queries on such models would take considerable amount of time and optimising queries would help to reduce this time. Current research in the domain of query languages, e.g. SQL, would explore promising directions and approaches.

Appendices

A

Categorisation Parameters

This appendix briefly describes the categorisation parameters introduced in Section 3.1, Figure 3.1.

A.1 TIM

This group of criteria indicates the type of the TIM explicitly defined or used in the context of an approach, if there is any. A TIM could be

- **General-purpose.** A general model to be used in any solution
- **Case-specific.** A model which is defined for a particular scenario or case and could be further classified as
  - **Static.** It is used directly, with no changes, in all similar scenarios or cases.
  - **Customisable.** It can be customised with respect to the context of the project at hand.

A.2 Activities

The purpose of the criteria in this group is to find out which traceability activities is covered and how. There are four activities and an approach is examined in the context of each activity with respect to the existing approaches to do an activity or its sub-activities.

1. Planning and Managing; it is determined which of the following sub-activities is covered in an approach:

- **Identifying Requirements.** mechanisms or techniques to identify traceability requirements
- **Defining a TIM.** mechanisms or techniques to define a TIM based on the identified requirements
- **Defining Traceability Process.** mechanisms or techniques to specify how and when activities are performed
- **Assessment (Feedback).** mechanisms or techniques to maintain a solution relevant regarding the evolving needs

2. Creation. It focuses on three aspects in this activity:
   a) Trace Acquisition. It demonstrates the approach used to create traces:
      i. Trace Capture. traces are created as activities are performed
         - **Event-driven (by-activity).** Traces are captures based on the events (i.e. specific development activities).
         - **Transformation (by-product).** Traces are created as artefacts (mainly models) are transformed to other ones.
      ii. Trace Recovery. It demonstrates which technique or method is used to identify traces from existing artefacts: **IR-based**, **rule-based**, or **misc/hybrid**.
   b) Storage; it shows how traces are stored
      - **Model;** traces are stored and represented as models
      - **Repository;** traces are stored in repositories (e.g. XML files)
   c) Automation. A cross-cutting parameter which could be used for all types of techniques. The proposed technique could be **automatic**, **semi-automatic** or **manual**.

3. Maintenance. It is demonstrated that which technique is used to maintain traces and how much it is automated.
   a) Mode. The maintenance of traces is performed in a **reactive** or in a **proactive** manner.
   b) Automation. A cross-cutting parameter which could be used for all types of techniques. The proposed technique could be **automatic**, **semi-automatic** or **manual**.

4. Usage. It discusses the two essential (sub-)activities to use traces and current techniques to support them:
   a) Trace Visualisation. Traces are visualised with one of the following techniques: **Matrices**, **Graphs**, or **Hyperlinks**.

b) Retrieving Traces. Traces have to be retrieved in order to support traceability scenarios. An approach might support
   - **Predefined Analyses.** Common and basic traceability scenarios
   - Retrieving traces with tool-specific **APIs**, **general** or **specific** query languages or model management languages.

A.3 Tooling

These criteria examine the type of the provided or specified tool in an approach and its features. An approach can be used with **general** available (requirements) traceability tools or it might require **specific** tools. Specific tools can be provided as a **complete toolkit**, which support other activities too, or a **partial** tool dedicated to the approach. Also, an specific tool might support **interoperability** and provide required mechanisms to interact with other tools.

A.4 Artefact

In terms of the artefacts which are supported, an approach could be

- **General.** It could deal with different types of artefacts in different formats.
- **Type-specific.** It is applied on specific types of artefacts (e.g. requirements, architecture, class diagram).
- **Format-specific.** It works with artefacts in specific formats and notations (e.g. text, model, XMI).

B

Summary of Existing Traceability Approaches

Table B.1 and Table B.2 provide a summary of available traceability approaches, introduced and discussed in chapter 2, based on the proposed categorisation in Section 3.1. Table B.1 covers Artefact, TIM, and Tooling parameters and Table B.2 focuses on supported Activities in traceability approaches.

Basically, in terms of the highest-level groups of parameters, the gray colour shows if a group is applicable to an approach; it is directly or explicitly considered or covered by an approach. Accordingly, white colour indicates that an approach does not cover or consider that group and the approach can not be examined based on that particular group. Then, a check mark (✓) in each cell shows if the correspondent approach covers or supports the related parameter.

Table B.1: Summary of Existing Traceability Approaches - Part 1
(Artefacts, TIM, and Tooling)

Approach	**Artefact**				**TIM**			**Tooling**			
	Type		Format			Case-Specific			specific		
	General	Specific	General	Specific	General	Static	Customisable	General	Partial	Complete	Interoperable
[Lago et al., 2009]							✓				
[Mäder et al., 2009a]	✓			✓			✓				
[Matragkas, 2011; Drivalos-Matragkas et al., 2010]	✓			✓			✓			✓	
[Ramesh and Jarke, 2001]	✓		✓				✓			✓	
[Spence and Probasco, 2000]		✓		✓		✓		✓			
[Letelier, 2002]		✓		✓			✓	✓			
[Jouault, 2005]	✓			✓	✓				✓		✓
[Amar et al., 2008]	✓			✓	✓				✓		✓
[Falleri et al., 2006]	✓			✓	✓				✓		✓
[Walderhaug et al., 2006]	✓			✓			✓				
[Kassab and Ormandjieva, 2006; Kassab et al., 2009]		✓		✓		✓					
[Anquetil et al., 2010]		✓		✓		✓				✓	✓
[Olsen and Oldevik, 2007]	✓			✓		✓				✓	✓
[Seibel et al., 2010, 2012]	✓			✓			✓			✓	
[Grammel and Vigot, 2009]	✓			✓		✓					
[Vanhooff et al., 2007]	✓			✓			✓		✓		
[Asuncion et al., 2010]	✓		✓							✓	✓

Approach	Artefact				TIM			Tooling			
	Type		Format			Case-Specific			specific		
	General	Specific	General	Specific	General	Static	Customisable	General	Partial	Complete	Interoperable
[Grechanik et al., 2007]		✓		✓						✓	✓
[Kagdi et al., 2007]		✓		✓					✓		
[Marcus and Maletic, 2003]		✓		✓							
[Mäder et al., 2008b]		✓		✓				✓			
[Spanoudakis et al., 2004]		✓		✓		✓					✓
[Gorp and Janssens, 2005]	✓			✓							✓
[Oldevik et al., 2005; Oldevik and Neple, 2006]		✓		✓		✓					
[Anderson et al., 2000]	✓			✓		✓				✓	✓
[Anderson et al., 2002]	✓			✓		✓				✓	✓
[Lucia et al., 2006b, 2007]	✓			✓					✓		
[Zou et al., 2010]	✓			✓		✓				✓	✓
[Cleland-Huang et al., 2005a; Cleland-Huang, 2005]		✓		✓					✓		
[Hayes et al., 2003]	✓			✓							
[Diaz et al., 2013]		✓		✓							
[Gethers et al., 2011]	✓			✓							
[Settimi et al., 2004]		✓		✓							
[McMillan et al., 2009]		✓		✓							
[Jirapanthong and Zisman, 2005]		✓		✓		✓			✓		
[Cysneiros and do Prado Leite, 2004]		✓		✓							

Approach	Artefact				TIM			Tooling			
	Type		Format			Case-Specific			specific		
	General	Specific	General	Specific	General	Static	Customisable	General	Partial	Complete	Interoperable
[Zimmermann et al., 2005]		✓		✓		✓			✓		
[Cleland-Huang et al., 2002b, 2003]	✓			✓		✓			✓		
[Pinheiro and Goguen, 1996]	✓			✓			✓			✓	
[Ying et al., 2004]		✓		✓							
[Pohl, 1996a]		✓		✓		✓			✓		
[Egyed and Grünbacher, 2005]		✓		✓					✓		
[Zhang et al., 2006]		✓		✓							
[Sherba et al., 2003]	✓		✓			✓			✓		✓
[Wenzel et al., 2007]	✓			✓					✓		✓
[Maletic et al., 2003, 2005]	✓			✓							
[Engels et al., 2002]		✓		✓							
[Mens et al., 2005]		✓		✓							
[Murta et al., 2006]		✓		✓					✓		
[Schwarz et al., 2008]		✓		✓		✓					
[Maletic and Collard, 2009]	✓			✓	✓						
[Mäder and Cleland-Huang, 2010]		✓		✓			✓		✓		✓
[Asuncion et al., 2007; Asuncion, 2008; Asuncion and Taylor, 2012]	✓		✓				✓			✓	✓
[Song et al., 2011]		✓		✓							
[Tekinerdogan et al., 2007a,b]	✓			✓		✓			✓		
[Krishna and Gregoriades, 2011]	✓			✓		✓					

Approach	**Artefact**				**TIM**			**Tooling**			
	Type		Format			Case-Specific			specific		
	General	Specific	General	Specific	General	Static	Customisable	General	Partial	Complete	Interoperable
[Mirakhorli and Cleland-Huang, 2011, 2012]		✓	✓				✓			✓	
[Katta and Stlhane, 2012]						✓					
[Albinet, 2008; Peraldi-Frati and Albinet, 2010]		✓		✓		✓				✓	✓
[Mason et al., 2003; Mason, 2005]	✓			✓		✓				✓	✓
[Schwarz et al., 2010]	✓			✓			✓	✓			

Table B.2: Summary of Existing Traceability Approaches - Part 2 (Activities)

Approach	Planning				Trace Creation									Maintenance				Usage						
					Auto.		Acquisition					Storage		Auto.		Mode		Visualisation			Retrieving			
							Capture		Recovery													Query		
	Identity Req.	Define TIM	Define Process	Assessment	Semi-automatic	Automatic	Transform	Event-based	IR	Rule	Misc	Model	Repository	Semi-automatic	Automatic	Reactive	Proactive	Matrix	Graph	Link	Predefined	API	General	Specific
[Lago et al., 2009]	✓	✓																						
[Mäder et al., 2009a]	✓	✓									✓		✓	✓			✓						✓	
[Matragkas, 2011; Drivalos-Matragkas et al., 2010]		✓			✓		✓		✓			✓		✓		✓							✓	
[Ramesh and Jarke, 2001]		✓			✓			✓		✓														
[Spence and Probasco, 2000]	✓	✓																						
[Letelier, 2002]		✓				✓	✓					✓												
[Jouault, 2005]						✓	✓					✓												
[Amar et al., 2008]						✓	✓					✓												
[Falleri et al., 2006]						✓	✓					✓												
[Walderhaug et al., 2006]		✓			✓							✓												
[Kassab and Ormandjieva, 2006; Kassab et al., 2009]		✓																			✓		✓	
[Anquetil et al., 2010]		✓			✓						✓		✓					✓	✓		✓			
[Olsen and Oldevik, 2007]					✓		✓					✓						✓					✓	
[Seibel et al., 2010, 2012]		✓			✓			✓		✓		✓		✓		✓		✓					✓	

Approach	Planning				Trace Creation									Maintenance				Usage						
					Auto.		Acquisition					Storage		Auto.		Mode		Visualisation			Retrieving			
							Capture		Recovery													Query		
	Identity Req.	Define TIM	Define Process	Assessment	Semi-automatic	Automatic	Transform	Event-based	IR	Rule	Misc	Model	Repository	Semi-automatic	Automatic	Reactive	Proactive	Matrix	Graph	Link	Predefined	API	General	Specific
[Grammel and Vigot, 2009]		✓				✓	✓					✓												
[Vanhooff et al., 2007]						✓	✓					✓												
[Asuncion et al., 2010]						✓		✓			✓		✓						✓					✓
[Grechanik et al., 2007]						✓					✓		✓						✓					
[Kagdi et al., 2007]						✓			✓				✓											
[Marcus and Maletic, 2003]						✓			✓				✓											
[Mäder et al., 2008b]															✓		✓							
[Spanoudakis et al., 2004]						✓				✓			✓		✓	✓								
[Gorp and Janssens, 2005]						✓	✓					✓												
[Oldevik et al., 2005; Oldevik and Neple, 2006]						✓	✓					✓												
[Anderson et al., 2000]						✓		✓					✓											
[Anderson et al., 2002]		✓			✓					✓				✓		✓				✓				
[Lucia et al., 2006b, 2007]					✓				✓					✓		✓								
[Zou et al., 2010]					✓				✓					✓		✓								
[Cleland-Huang et al., 2005a; Cleland-Huang, 2005]					✓				✓					✓		✓					✓			
[Hayes et al., 2003]					✓				✓															

Approach	Planning				Trace Creation										Maintenance			Usage							
					Acquisition							Storage			Auto.	Mode		Visualisation			Retrieving				
					Auto.		Capture			Recovery											Query				
	Identify Req.	Define TIM	Define Process	Assessment	Semi-automatic	Automatic	Transform	Event-based	IR	Rule	Misc	Model	Repository		Semi-automatic	Automatic	Reactive	Proactive	Matrix	Graph	Link	Predefined	API	General	Specific
[Diaz et al., 2013]					✓				✓						✓		✓								
[Gethers et al., 2011]					✓				✓						✓		✓								
[Settimi et al., 2004]					✓				✓																
[McMillan et al., 2009]				✓			✓																		
[Jirapanthong and Zisman, 2005]						✓				✓															
[Cysneiros and do Prado Leite, 2004]					✓					✓		✓													
[Zimmermann et al., 2005]						✓				✓			✓						✓			✓		✓	
[Cleland-Huang et al., 2002b, 2003]						✓		✓								✓		✓							
[Pinheiro and Goguen, 1996]		✓				✓		✓		✓			✓		✓		✓		✓		✓	✓		✓	
[Ying et al., 2004]						✓				✓			✓												
[Pohl, 1996a]					✓			✓					✓												
[Egyed and Grünbacher, 2005]					✓						✓								✓	✓		✓			
[Zhang et al., 2006]						✓					✓														
[Sherba et al., 2003]		✓			✓						✓		✓												✓
[Wenzel et al., 2007]					✓			✓												✓					
[Maletic et al., 2003, 2005]					✓						✓		✓		✓			✓						✓	

Approach	Planning				Trace Creation									Maintenance				Usage						
					Auto.		Acquisition					Storage		Auto.		Mode		Visualisation			Retrieving			
							Capture		Recovery													Query		
	Identity Req.	Define TIM	Define Process	Assessment	Semi-automatic	Automatic	Transform	Event-based	IR	Rule	Misc	Model	Repository	Semi-automatic	Automatic	Reactive	Proactive	Matrix	Graph	Link	Predefined	API	General	Specific
[Engels et al., 2002]															✓	✓								
[Mens et al., 2005]															✓	✓								
[Murta et al., 2006]															✓		✓	✓		✓				
[Schwarz et al., 2008]																			✓				✓	
[Maletic and Collard, 2009]																								✓
[Mäder and Cleland-Huang, 2010]	✓	✓																						✓
[Asuncion et al., 2007; Asuncion, 2008; Asuncion and Taylor, 2012]	✓	✓				✓		✓					✓	✓		✓		✓	✓	✓		✓		
[Song et al., 2011]						✓		✓																
[Tekinerdogan et al., 2007a,b]					✓					✓			✓										✓	
[Krishna and Gregoriades, 2011]																								
[Mirakhorli and Cleland-Huang, 2011, 2012]	✓	✓									✓		✓	✓		✓		✓	✓					
[Katta and Stlhane, 2012]																								
[Albinet, 2008; Peraldi-Frati and Albinet, 2010]											✓	✓												

Approach	Planning				Trace Creation										Maintenance			Usage								
	Identity Req.	Define TIM	Define Process	Assessment	Acquisition							Storage			Auto.	Mode		Visualisation			Retrieving				
					Auto.	Capture			Recovery												Query				
						Transform	Event-based	IR	Rule	Misc	Model	Repository	Semi-automatic	Automatic	Reactive	Proactive	Matrix	Graph	Link	Predefined	API	General	Specific
[Mason et al., 2003; Mason, 2005]													✓		✓										
[Schwarz et al., 2010]		✓			✓	✓		✓				✓	✓		✓			✓		✓					✓

C

GSN Diagrams

In this appendix, a larger view of the GSN diagrams which were introduced in the thesis is provided. Figure C.1 shows the identified requirements of a multi-domain traceability solution and Figure C.2 and C.3 demonstrate the overall argument on how the requirements and research objectives have been fulfilled.

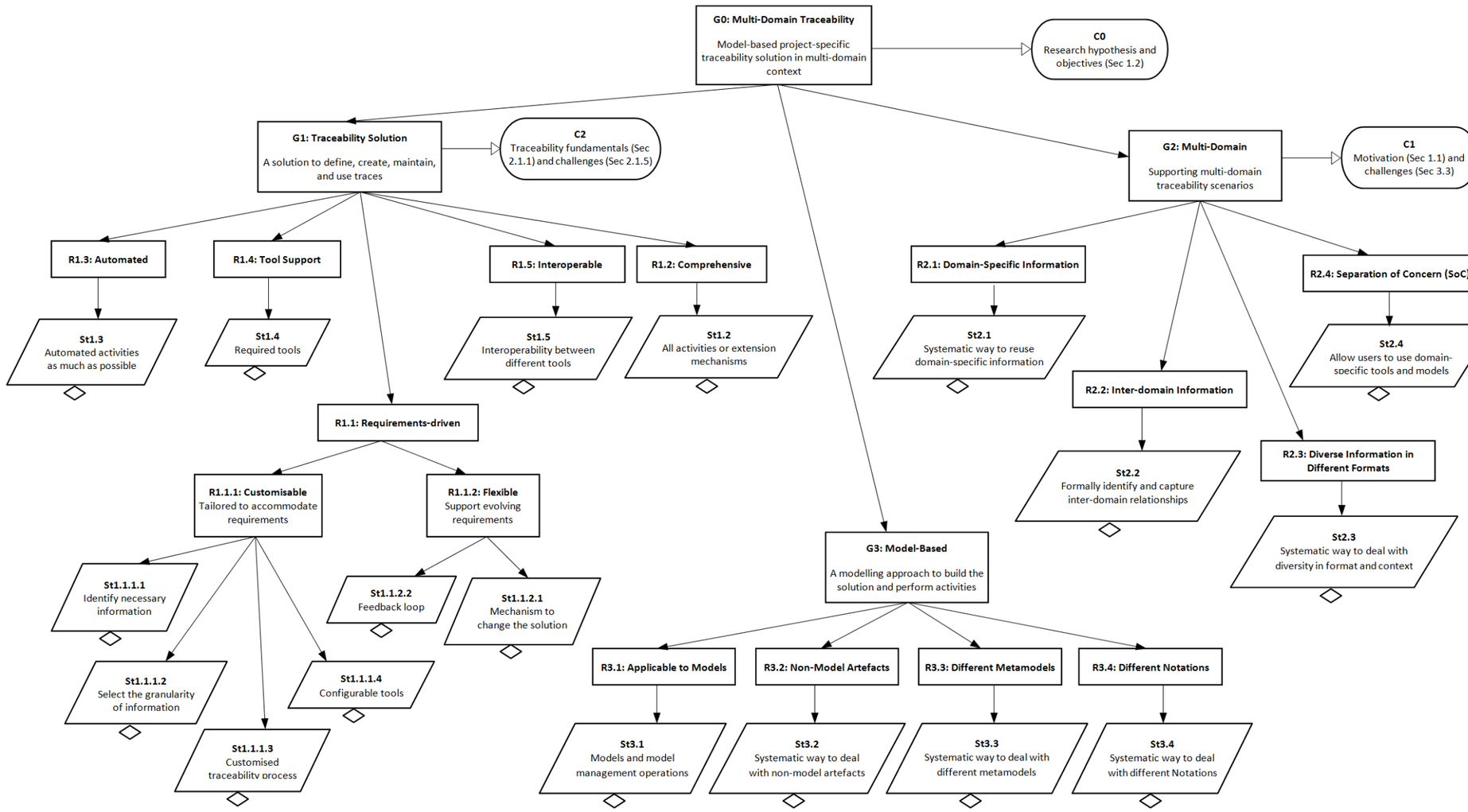


Figure C.1: GSN diagram representing requirements of a multi-domain traceability solution (overall argument)

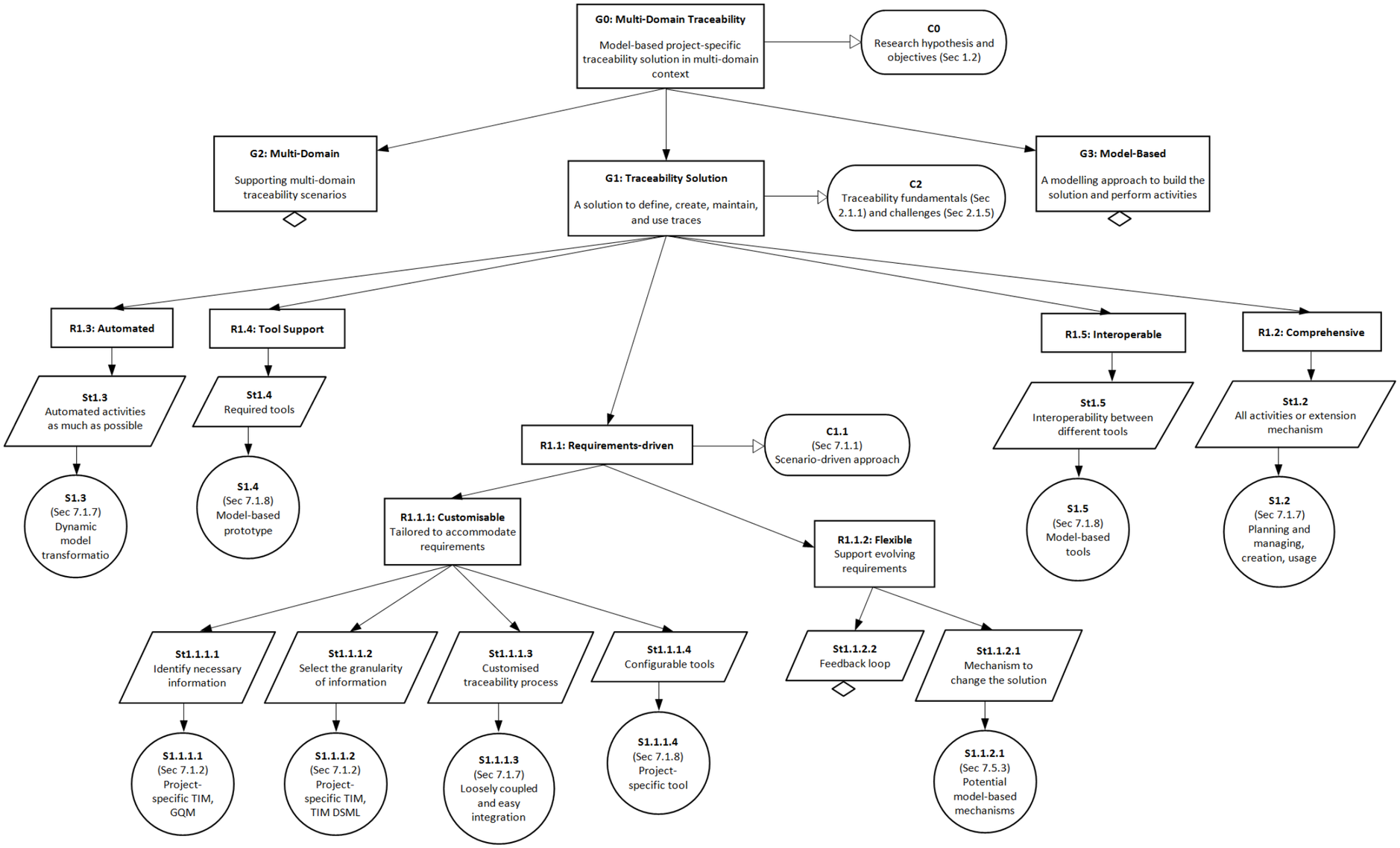


Figure C.2: GSN diagram summarising the requirements-based evaluation for goal G1

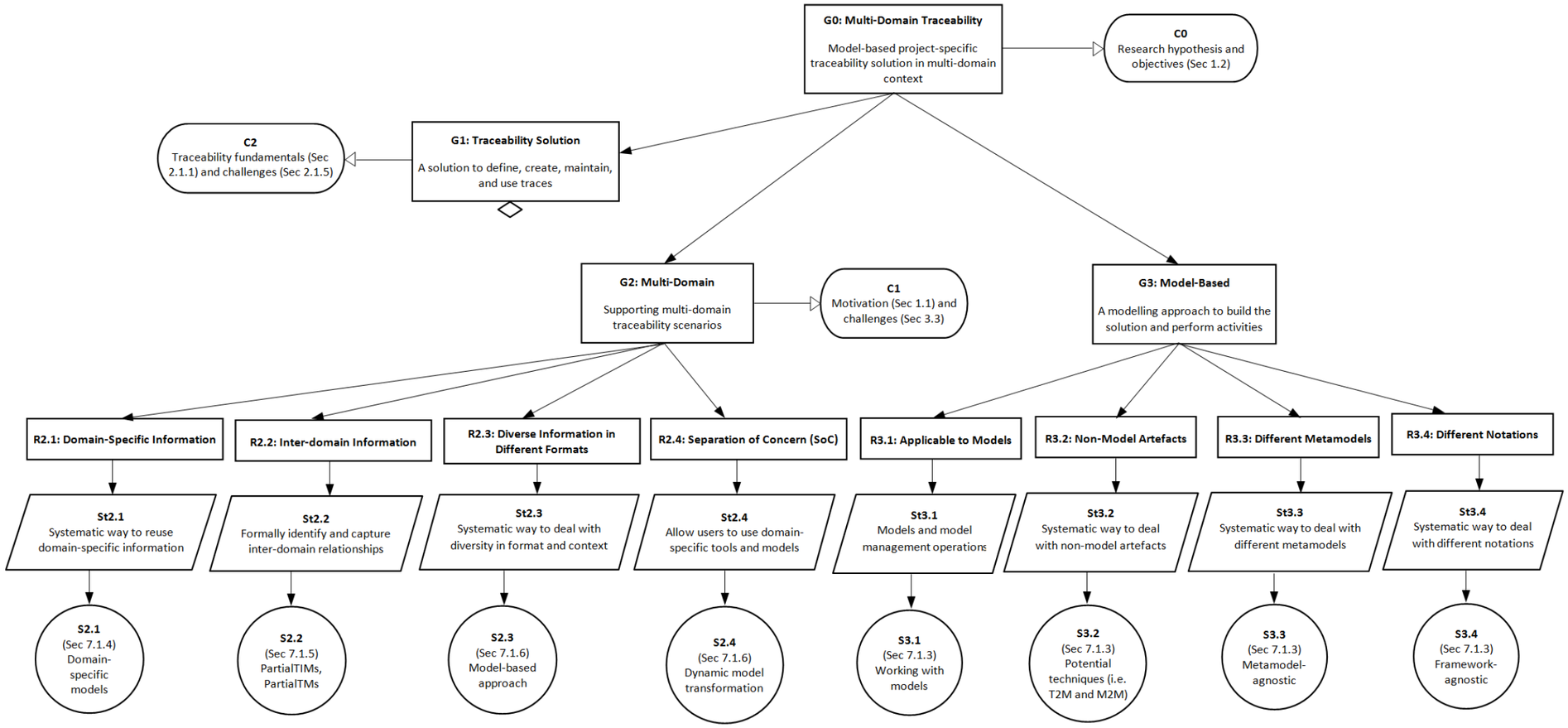


Figure C.3: GSN diagram summarising the requirements-based evaluation for goals G2 and G3

D

TAL Xtext Grammar

This appendix shows the *general* part of the TAL grammar. This part consists of the grammar rules for defining a query or a constraint, describing a condition, and explaining an expression in TAL.

```
QueryModel:
  (queries += Query | checks += Constraint)*
;

Query:
  (comment = SL_COMMENT)?
  'Query' qName = ID ('(' params += Parameter (';' params
    += Parameter)* ')')? '{'
    query = QueryFunction
  '}'
;

Constraint:
  (comment = SL_COMMENT)?
  'Constraint' cName = ID ('(' params += Parameter (';'
    params += Parameter)* ')')? '{'
    check = ValidationFunction
  '}'
;

UsedQuery:
  qName = ID ('as' alias = ID)?
;

QueryFunction:
  Find | Count
;

Find:
  'find' element = TraceableElement ('where' cond =
    Condition)?
;
```

```
Count:
  'count' element = TraceableElement ('where' cond =
     Condition)?
;

ValidationFunction:
  Exist | ExistOne | ForAll
;

Exist:
  'exist' element = Element ('where' cond = Condition)?
;

ExistOne:
  'existOne' element = Element ('where' cond = Condition)?
;

ForAll:
  'forAll' element = TraceableElement ('where' restriction
     = Condition)? ':'
    cond = Condition
;

Condition:
   mainExpr = Expression (additionalExprs += ExtendedExpr)*
;

Expression:
  (not ?= 'not')? (beg ?= '(') (stmt = SimpleExpr | func =
     ValidationFunction) (end ?= ')')
;

SimpleExpr:
  left = Operand op = Operator rightOp = Operand
  | left  = Operand op = Operator rightVal = ConstantValue
  | left = Operand
  | left = Operand 'in' input = ID ('(' paramVal = ID')')?
  | left = Operand 'in' '(' inQuery = Find ')'
  | source = TraceableElement traceType = TraceLink target
     = TraceableElement;

ExtendedExpr:
  conj = AndOr exp = Expression
;

Parameter:
  pName = ID ':' pType = TraceableElement
;
```

```
ConstantValue:
    STRING | INT | 'null' | 'true' | 'false'
;

Operator:
   '=' | '<' | '>' | '<>' | '<=' | '>='
;

AndOr:
   'and' | 'or'
;

BuiltInFunction:
   'size()' | 'isDefined()' | ('startsWith(' Input ')')
;

Input:
   ConstantValue (',' ConstantValue)*
;

Element:
   TraceableElement | TraceLink
;
```

Listing D.1: Xtext grammar of the general part of the TAL grammar

E

Dynamic Model Transformation

Listing E.1 provides the EOL program for the dynamic model transformation used to generate a TM.

```
1  var emfTool = new Native("org.eclipse.epsilon.emc.emf.tools.
       EmfTool");
2  var myTool = new Native("org.eclipse.epsilon.examples.tools.
       MyJavaTool");
3
4  var ecoreUtil = emfTool.ecoreUtil;
5
6  //Input: Mapping Model
7  var mappingModel = inputMappingModel!MappingModel.all().first
       ();
8
9  var TIM  = mappingModel.TIM;
10 var mappingEntries = mappingModel.mappings;
11 var equivalents = mappingModel.equivalences;
12 var sourceMetaModels = mappingModel.sourceMetaModels;
13
14 var sourceModelElements = new Map;
15 for (mm in sourceMetaModels) {
16  //Locate the source model for the specified metamodel and
       put its elements in the map
17   sourceModelElements.put(mm.name, loadSourceModelElements(mm
       ));
18 }
19
20
21 //Output: Traceability Model
22 var traceabilityModel = ecoreUtil.create(TIM.theMetaModel);
23 traceabilityModel.name = mappingModel.name;
24 emfTool.createModel(traceabilityModel, "
       outputTraceabilityModel");
25
```

```
26 //Utility Model: Equivalent Model (a generic traceability
       model)
27 var equivalentModel = new outputEquivalentModel!
       TraceabilityModel;
28 equivalentModel.name = mappingModel.name;
29 var eqLinkType = new outputEquivalentModel!LinkType;
30 eqLinkType.name = "Equivalent";
31 equivalentModel.linkTypes.add(eqLinkType);
32 var sameLinkType = new outputEquivalentModel!LinkType;
33 sameLinkType.name = "same";
34 equivalentModel.linkTypes.add(sameLinkType);
35
36 //Iterate over mapping entries
37 var entry : inputMappingModel!MappingEntry;
38 for (entry in mappingEntries) {
39  //Assume that endPoint1 always refers to TIM
40  var endPoint1 : inputMappingModel!ModelElementRef = entry.
       endPoint1;
41  var endPoint2 : inputMappingModel!ModelElementRef = entry.
       endPoint2;
42  var definitions = entry.definitions;
43  var conditions = entry.conditions;
44
45
46  //Select the elements belong to the desired source model
47  var inModelElements = sourceModelElements.get(endPoint2.
       owner.name);
48
49  //Target reference in the TM used to add the elements to the
        model
50  var targetEReference = traceabilityModel.eClass().
       getEAllReferences().selectOne(sf | sf.getEType().name =
       endPoint1.theElement.name);
51
52  var inputElements;
53
54  if (endPoint1.type = inputMappingModel!ModelElementType#
       Clazz and endPoint2.type = inputMappingModel!
       ModelElementType#Clazz) {
55   inputElements = inModelElements.select(el | isTypeOf(el,
        endPoint2.theElement));
56
57   var outElements : Sequence = new Sequence;
58   for (element : Any in inputElements) {
59    var outElement = outputTraceabilityModel.createInstance(
        endPoint1.theElement.name);
60
61    //Initialise the new element based on definitions
62    initialiseElement(definitions, outElement, null, element,
        null);
63
64    //Add the new element into the output TM
65    traceabilityModel.eGet(targetEReference).add(outElement);
66
```

```
67      //Add an equivalenve link in the utility model
68      addToEquivalentModel(element, outElement, eqLinkType);
69
70      //Check for equivalents in source models
71      var equal = equivalents.selectOne(eq | eq.endPoint1.owner
            = endPoint2.owner and eq.endPoint1.theElement =
            endPoint2.theElement);
72      if (equal.isDefined()) {
73       var links = sourceModelElements.select(el | isTypeOf(el,
             equal.corrTraceLink.theElement));
74
75       if (links.isDefined() and not links.isEmpty()) {
76        //Retrieve the StructuralFeature from the eClass!!!
77        var tempLink = links.at(0);
78
79        var sfName1 = equal.definitions.selectOne(def | def.name
               = "ep1").stmtStr;
80        var sf1 = tempLink.eClass().getEAllStructuralFeatures().
              selectOne(sf | sf.name = sfName1);
81        var sfName2 = equal.definitions.selectOne(def | def.name
               = "ep2").stmtStr;
82        var sf2 = tempLink.eClass().getEAllStructuralFeatures().
              selectOne(sf | sf.name = sfName2);
83
84        var link = links.selectOne(l | l.eGet(sf1) = element);
85        if (link.isDefined()) {
86         addToEquivalentModel(link.eGet(sf2), outElement,
               sameLinkType);
87        }
88       }
89      }
90     }
91    } else if (endPoint1.type = inputMappingModel!
          ModelElementType#Clazz and endPoint2.type =
          inputMappingModel!ModelElementType#Reference) {
92     for (modEl in inModelElements) {
93      var sf = modEl.eClass().getEAllStructuralFeatures().
            selectOne(sf | sf.name = endPoint2.theElement.name);
94      if (sf.isDefined()) {
95       inputElements = modEl.eGet(sf);
96       if (inputElements.isDefined()) {
97        for (element in inputElements) {
98         var outElement = outputTraceabilityModel.createInstance
               (endPoint1.theElement.name);
99         //Initialise the new element based on definitionss
100        initialiseElement(definitions, outElement, null, modEl,
               element);
101
102        //Add the new element into the output TM
103        traceabilityModel.eGet(targetEReference).add(outElement
              );
104       }
105      }
106     }
```

```
107     }
108    }
109   }
110
111   operation initialiseElement(definitions : Any, el1Src : Any,
          el1Tar : Any, el2Src : Any, el2Tar : Any) {
112    if (definitions.isUndefined())
113     return;
114
115    var el1 = el1Src;
116    var el2 = el2Src;
117    for (def in definitions) {
118     var parts = def.stmtStr.split(' = ');
119
120     //the TIM side: a reference to the attribute
121     var left: Any = parts.at(0);
122
123     //the other side: resolved value for the attribute
124     var right : Any = parts.at(1);
125
126     //Extract the name of the attribute of the element in TM
127     var attr1 = left.split('\\.').at(1); //.split('\\]').at(0);
128
129     //Resolve the value
130     var rightParts = right.split('\\+');
131     var value;
132     for (part in rightParts){
133      var arr = part.split('\\.');
134      for (a in arr) {
135       a = a.trim();
136       //Manage keywords
137       if (a.trim() = "ep2")
138        continue;
139       if (a = "_source") {
140        el2 = el2Src;
141       } else if (a = "_target") {
142        el2 = el2Tar;
143       } else {
144        var attr2 = a;
145        var srcEsf = el2.eClass().getEAllStructuralFeatures().
              selectOne(sf | sf.name = attr2);
146        if (srcEsf.isDefined()) {
147         if (srcEsf.isTypeOf(EAttribute)) {
148          value = value + el2.eGet(srcEsf);
149         } else if (srcEsf.isTypeOf(EReference)) {
150          value =  getEquivalent(el2.eGet(srcEsf));
151         }
152        } else {
153         value = value + (a.replace("\"", ""));
154        }
155       }
156      }
157     }
158
```

```
159    if (value.isUndefined()) {
160     value = getEquivalent(el2);
161    }
162
163    var tarEsf = el1.eClass().getEAllStructuralFeatures().
           selectOne(sf | sf.name = attr1);
164    if (tarEsf.isTypeOf(EAttribute)) {
165     el1.eSet(tarEsf, ecoreUtil.createFromString(tarEsf.
            getEAttributeType(), value));
166    } else if (tarEsf.isTypeOf(EReference)) {
167     el1.eSet(tarEsf, value);
168    }
169   }
170  }
171
172  operation addToEquivalentModel(source : Any, target : Any,
       linkType: Any) {
173   var link = new outputEquivalentModel!Link;
174   link.name = source.name + " -- " + target.name;
175   link.type = linkType;
176   link.linkEnd1 = source;
177   link.linkEnd2 = target;
178   equivalentModel.links.add(link);
179  }
180
181  operation Any getEquivalent(element : Any) {
182   var eqEl = equivalentModel.links.selectOne(l | l.linkEnd1 =
         element);
183   if (eqEl.isDefined())
184    return eqEl.linkEnd2;
185   return null;
186  }
187
188  operation Boolean isTypeOf(element : Any, type : Any) {
189   return element.eClass().name = type.name;
190  }
191
192  operation Any loadSourceModelElements(metamodel : Any) {
193   var srcModel = System.user.prompt("Locate the model for '" +
          metamodel.name + "'");
194   var srcEMFModel;
195   if (srcModel.isDefined()) {
196    srcEMFModel = myTool.createEmfModel(metamodel.name+"_model"
          , srcModel, metamodel.theMetaModel.eContainer().
          getNsURI(), true, false);
197
198    if (srcEMFModel.isDefined())
199     return srcEMfModel.allContents();
200   }
201   return null;
202  }
```

Listing E.1: Dynamic model transformation EOL program

F

RVSM Case Study Supplement

This appendix provides the complementary information for the RVSM case study including the GQM EMF model, EVL constraints for the TIM, partialTIMs, and the complete mapping model.

F.1 GQM Model

Figure F.1 shows the EMF model of the GQM model defined in the case study.

F.2 EVL Constraints

In this section, the EVL constraints which are defined for the TIM in the RVSM case study are provided. Listing F.1 shows these constraints.

The first constraint describes that a safety integrity requirement can be directly allocated to a system element if and only if its associated hazard is classified as SafetyCritical. The second one specifies that actions required by a derived safety requirement have to trace to mitigations identified for the hazard associated with the safety requirement. The last ones have been introduced in Section 6.5.1.

```
1 context SafetyIntegrityRequirement {
2   guard : DirAllocatedTo.all().exists(d | d.traceLinkEnd1 =
        self)
3   constraint SafetyCriticalHazardDirAllocatedToSysElement{
4   check : not DefinedFor.all.exists(
5       df | df.traceLinkEnd1 = self
6       and
7       df.traceLinkEnd2.type = HazardClassification#
            NonSafetyCritical);
```

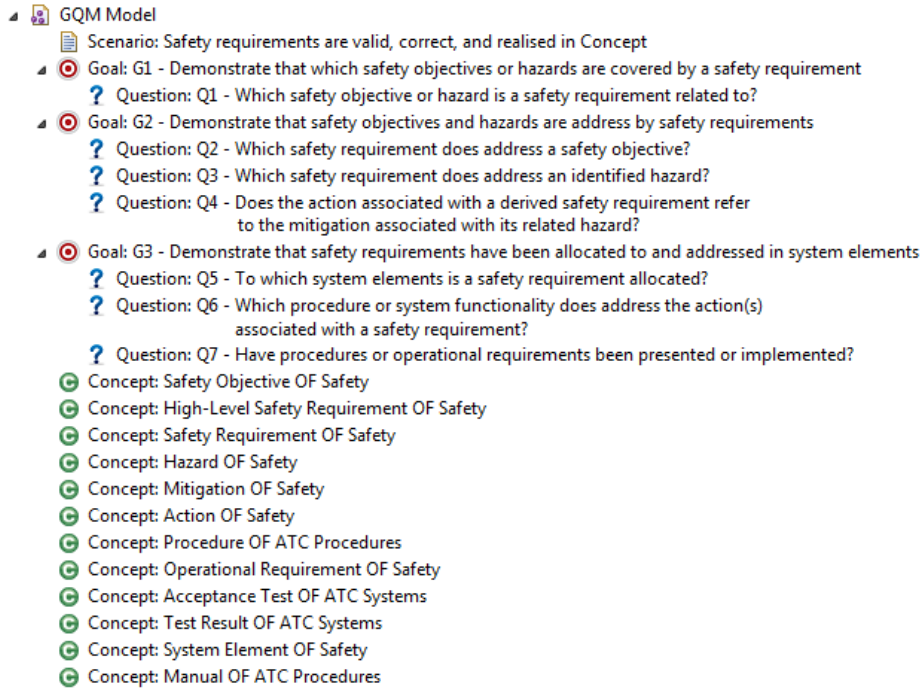


Figure F.1: The GQM EMF model for the example traceability scenario in RVSM

```
8
9     message : "Safety Integrity Requirement `" + self + "' has
          been defined based on a NonSafetyCritical hazard and
          could not be directly allocated to system elements";
10    }
11  }
12
13  context Action {
14    guard : RequiredBy.all().exists(rb | rb.traceLinkEnd1 =
          self and rb.traceLinkEnd2.derived)
15    constraint ActionReferToMitigation{
16    check :
17       var sr = RequiredBy.all().selectOne(rb | rb.
             traceLinkEnd1 = self).traceLinkEnd2;
18       var sir = DerivedFrom.all()selectOne(df | df.
             traceLinkEnd1 = sr).traceLinkEnd2;
19       var hazard = DefinedFor.all().selectOne(df | df.
             traceLinkEnd1 = sir).traceLinkdEnd2;
20       var mitigations = MitigatedBy.all().select(mb | mb.
             traceLinkEnd1 = hazard);
21       return mitigations.forAll(m | ReferTo.all.exists(rt |
             rt.traceLinkEnd1 = self and rt.traceLinkEnd2 = m));
22
```

```
23    message : "Action `" + self + "' has to refer to related
          mitigations";
24    }
25  }
26
27  context SafetyRequirement {
28    guard : self.derived
29    constraint DerivedForNonSafetyCriticalHazard {
30    check :
31       var safetyIntegrityReq = DerivedFrom.all().selectOne( df
              | df.traceLinkEnd1 = self).traceLinkEnd2;
32       return not DefinedFor.all.exists(df | df.traceLinkEnd1 =
              safetyIntegrityReq and df.traceLinkEnd2.type =
             HazardClassification#SafetyCritical);
33
34    message : "Safety Requirement `" + self + "' traces to a
          SafetyCritical hazard";
35    }
36  }
37
38  context SafetyRequirement {
39    constraint NoDanglingSafetyRequirement{
40    check : DerivedFrom.all.exists(df | df.traceLinkEnd1 = self
41                       and
42                        df.traceLinkEnd2.isDefined())
43       or
44       DecomposedTo.all.exists(dt | dt.traceLinkEnd2 = self
45                       and
46                       dt.traceLinkEnd1.isDefined())
47
48    message : "All Safety Requirement have to be traced to a
          safety objective or a hazard";
49    }
50  }
```

Listing F.1: EVL constraints for the TIM in RVSM

F.3 partialTIMs

In this section, the partialTIMs defined to specify required inter-domain relationships are provided.

F.4 Mapping Model

The mapping model for the RVSM project is shown in two parts. Figure F.6 shows the mapping between TraceableElements and other elements. Figure F.7 shows how TraceLinks are mapped to other models, in addition to three equivalence relationships identified between domain-specific models (defined in Section 6.5.2).

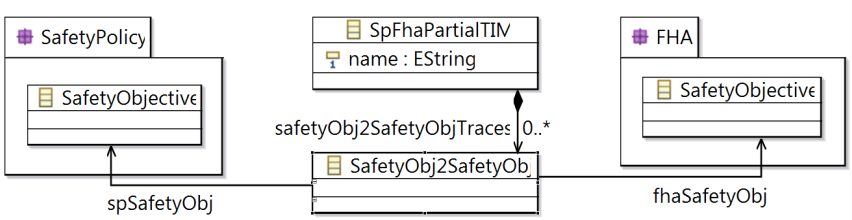


Figure F.2: The partialTIM between Safety Policy and FHA

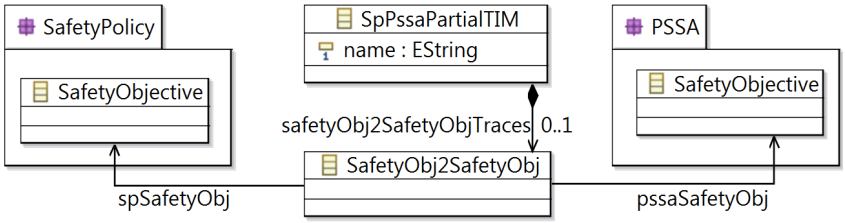


Figure F.3: The partialTIM between Safety Policy and PSSA

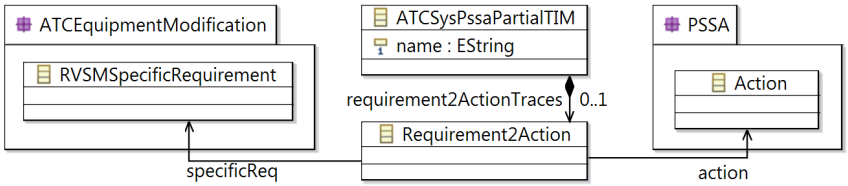


Figure F.4: The partialTIM between PSSA and ATC supporting system development

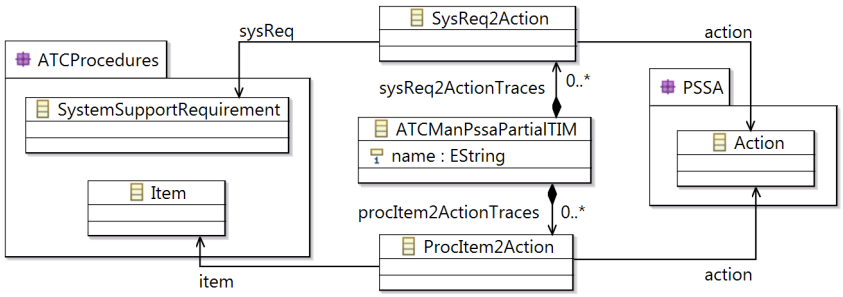


Figure F.5: The partialTIM between PSSA and ATC Manual

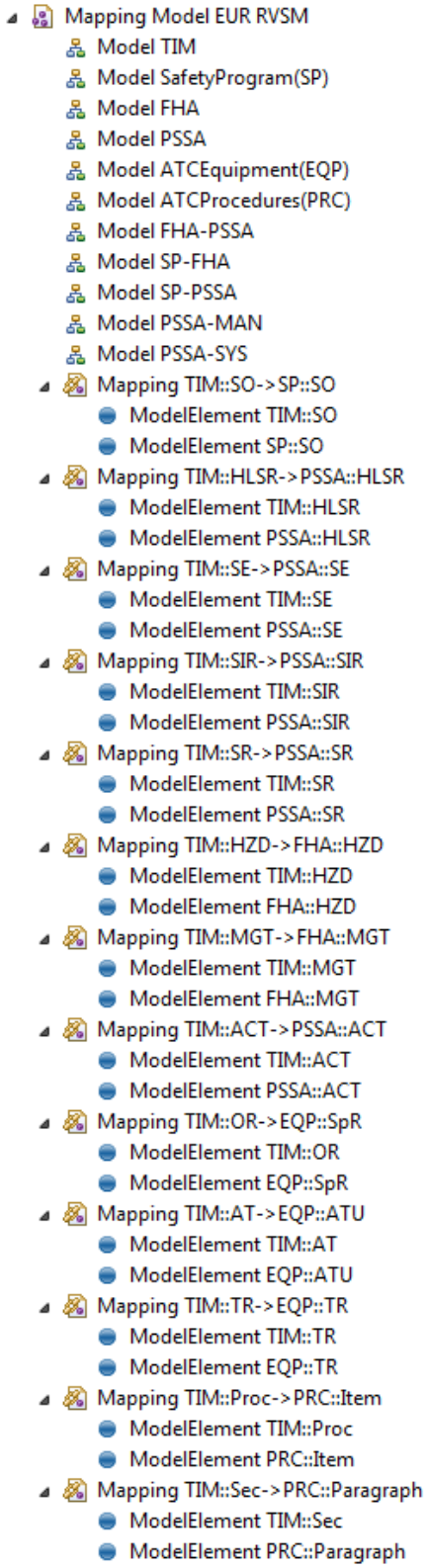


Figure F.6: The mapping model - part 1

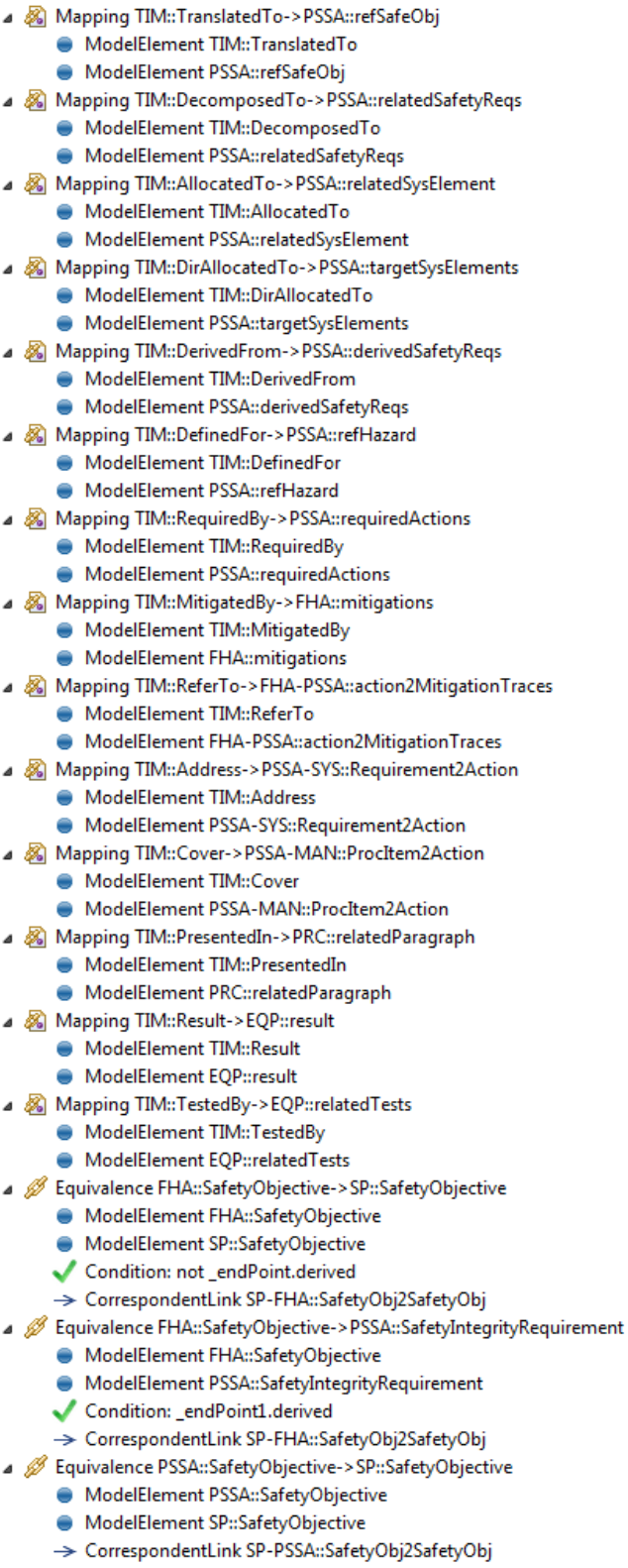


Figure F.7: The mapping model - part 2

References

[Abadi et al., 2008] A. Abadi, M. Nisenson, and Y. Simionovici. A traceability technique for specifications. In *Proc. of the 16th IEEE International Conference on Program Comprehension*, ICPC '08, pages 103–112. IEEE Computer Society, 2008.

[Aizenbud-Reshef et al., 2005] N. Aizenbud-Reshef, R. F. Paige, J. Rubin, Y. Shaham-Gafni, and D. S. Kolovos. Operational semantics for traceability. In *Proc. of the ECMDA Traceability Workshop*, ECMDA-TW '05, pages 8–14. Sintef, 2005.

[Aizenbud-Reshef et al., 2006] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni. Model traceability. *IBM Systems Journal*, 45(3):515–526, 2006.

[Ajila and Kaba, 2004] S. A. Ajila and A. B. Kaba. Using traceability mechanisms to support software product line evolution. In *Proc. of the IEEE International Conference on Information Reuse and Integration*, IRI '04, pages 157–162. IEEE Computer Society, 2004.

[Albinet, 2008] A. Albinet. The MeMVaTEx methodology: From requirements to models in automotive application design. In *Proc. of the 4th European Congress on Embedded Real-Time Software*, ERTS '08, 2008.

[Alexander et al., 2005] I. Alexander, S. Robertson, and N. Maiden. What influences the requirements process in industry? a report on industrial practice. In *Proc. of the 13th IEEE International Requirements Engineering Conference*, RE '05, pages 411–415. IEEE Computer Society, 2005.

[Alexander, 2002] I. Alexander. Towards automatic traceability in industrial practice. In *Proc. of the 1st International Workshop on Traceability*, pages 26–31, 2002.

[Alexander, 2005] I. Alexander. A taxonomy of stakeholders: human roles in system development. *International Journal of Technology and Human Interaction*, 1(1):23–59, 2005.

[Alexander and Maiden, 2004] I. Alexander and N. Maiden. *Scenarios, stories, use cases: through the systems development life-cycle.* J. Wiley and sons, Chichester, 2004.

[Amar et al., 2008] B. Amar, H. Leblanc, and B. Coulette. A traceability engine dedicated to model transformation for software engineering. In *Proc. of the ECMDA Traceability Workshop*, ECMDA-TW '08, 2008.

[Ambler and Jeffries, 2002] S. W. Ambler and R. Jeffries. *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process.* Wiley, 2002.

[AMPLE, 2007] AMPLE. Project AMPLE: Aspect-Oriented, Model-Driven Product Line Engineering. `http://ample.holos.pt`, 2007.

[Anderson et al., 2000] K. M. Anderson, R. N. Taylor, E. J. Whitehead, and Jr. Chimera: Hypermedia for heterogeneous software development environments. *ACM Transactions on Information Systems*, 18:211–245, 2000.

[Anderson et al., 2002] K. M. Anderson, S. A. Sherba, and W. V. Lepthien. Towards large-scale information integration. In *Proc. of the 22nd International Conference on Software Engineering*, ICSE '02, pages 524–534. ACM, 2002.

[Anquetil et al., 2010] N. Anquetil, U. Kulesza, R. Mitschke, A. Moreir, J.-C. Royer, A. Rummler, and A. Sousa. A model-driven traceability framework for software product lines. *Software and Systems Modeling*, 9(4):427–451, 2010.

[Antoniol et al., 2001] G. Antoniol, G. Canfora, G. Casazza, and A. D. Lucia. Maintaining traceability links during object-oriented software evolution. *Software: Practice and Experience*, 31:331–355, 2001.

[Antoniol et al., 2002] G. Antoniol, G. Canfora, G. Casazza, D. L. Andrea, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.

[Appleton et al., 2007] B. Appleton, R. Cowham, and S. Berczuk. Lean traceability: A smattering of strategies and solutions. `http://meu-tcc.googlecode.com/svn/trunk/Artigos/Lean%20Traceability%20a%20smattering%20of%20strategies%20and%20solutions`, 2007.

[Appleton, 2005] B. Appleton. Traceability and trustability. `http://bradapp.blogspot.co.uk/2005/03/traceability-and-trust-ability.html`, 2005.

[Arkley and Riddle, 2005] P. Arkley and S. Riddle. Overcoming the traceability benefit problem. In *Proc. the 13th IEEE International Requirements Engineering Conference*, RE '05, pages 385–389. IEEE Computer Society, 2005.

[Asuncion, 2008] H. U. Asuncion. Towards practical software traceability. In *Proc. of the 30th International Conference on Software Engineering - Companion*, ICSE Companion '08, pages 1023–1026. ACM, 2008.

[Asuncion and Taylor, 2012] H. U. Asuncion and R. N. Taylor. Automated techniques for capturing custom traceability links across heterogeneous artifacts. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 129–146. Springer, 2012.

[Asuncion et al., 2007] H. U. Asuncion, F. François, and R. N. Taylor. An end-to-end industrial software traceability tool. In *Proc. of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC-FSE '07, pages 115–124. ACM, 2007.

[Asuncion et al., 2010] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor. Software traceability with topic modeling. In *Proc. of the 32nd International Conference on Software Engineering - Volume 1*, ICSE '10, pages 95–104. ACM, 2010.

[ATESST2 Consortium, 2010] ATESST2 Consortium. Advancing Traffic Efficiency and Safety through Software Technology phase 2 (ATESST2). `http://www.atesst.org/scripts/home/publigen/content/templates/show.asp?P=125&L=EN&ITEMID=8`, 2010.

[Australian Transport Safety Burea, 2005] Australian Transport Safety Burea. Aviation safety investigation report - in-flight upset event 240 km northwest of perth, wa, boeing company 777-200, 9m-mrg. `http://www.atsb.gov.au/media/24550/aair200503722_001.pdf`, 2005.

[Baeza-Yates and Ribeiro-Neto, 1999] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press/Addison-Wesley, 1999.

[Barbero et al., 2007] M. Barbero, M. Didonet, D. Fabro, and J. Bézivin. Traceability and provenance issues in global model management. In *Proc. of the ECMDA Traceability Workshop*, ECMDA-TW '07, 2007.

[Basili and Caldiera, 1994] V. R. Basili and G. Caldiera. Goal question metric paradigm. *Encyclopaedia of Software Engineering*, 1:528–532, 1994.

[Bass et al., 1998] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[Bayer and Widen, 2002] J. Bayer and T. Widen. Introducing traceability to product lines. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, PFE '01, pages 409–416. Springer-Verlag, 2002.

[Beck and Andres, 2004] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2nd edition, 2004.

[Bernstein and Melnik, 2007] P. A. Bernstein and S. Melnik. Model management 2.0: Manipulating richer mappings. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 1–12. ACM, 2007.

[Bézivin and Gerbe, 2001] J. Bézivin and O. Gerbe. Towards a precise definition of the OMG/MDA framework. In *Proc. of the 16th IEEE/ACM International Conference on Automated Software Engineering*, ASE '01, pages 273–280, 2001.

[Bézivin et al., 2004] J. Bézivin, F. Jouault, and P. Valduriez. On the need for megamodels. In *Proc. of the OOPSLA and GPCE Workshop on Best Practices for Model Driven Software Development*, OOPSLA '04, 2004.

[Bézivin et al., 2006] J. Bézivin, S. Bouzitouna, M. D. D. Fabro, M.-P. Gervais, F. Jouault, D. Kolovos, I. Kurtev, and R. F. Paige. A canonical scheme for model composition. In *Proc. of the 2nd European Conference on Model Driven Architecture-Foundations and Applications*, ECMDA-FA '06, pages 346–360, 2006.

[BigLever Software Inc, 2012] BigLever Software Inc. BigLever Software Gears. `http://www.biglever.com`, 2012.

[Bouzitouna et al., 2005] S. Bouzitouna, M.-P. Gervais, , and X. Blanc. Model reuse in MDA. In H. R. Arabnia and H. Reza, editors, *Proc. of the Software Engineering Research and Practice*, pages 354–360. CSREA Press, 2005.

[Burge and Brown, 2008] J. E. Burge and D. C. Brown. Software engineering using {RATionale}. *Journal of Systems and Software*, 81(3):395–413, 2008.

[Capobianco et al., 2013] G. Capobianco, A. D. Lucia, R. Oliveto, A. Panichella, and S. Panichella. Improving ir-based traceability recovery via noun-based indexing of software artifacts. *Journal of Software: Evolution and Process*, 25(7):743–762, 2013.

[Center of Excellence for Software Traceability (CoEST), 2014] Center of Excellence for Software Traceability (CoEST). Tracelab. `http://www.coest.org/index.php/about-tracelab`, 2014.

[Chan and Paige, 2005] Z. E. Chan and R. F. Paige. Designing a domain-specific contract language: A metamodelling approach. In *Proc. of the 1st European Conference on Model Driven Architecture: Foundations and Applications*, ECMDA-FA '05, pages 175–189. Springer-Verlag, 2005.

[Charalambous, 2007] R. Charalambous. Towards agile engineering of high integrity software. Master's thesis, Department of Computer Science, University of York, 2007.

[Chrissis et al., 2006] M. B. Chrissis, M. Konrad, and S. Shrum. *CMMI: Guidelines for Process Integration and Product Improvement (SEI Series in Software Engineering)*. Addison-Wesley Longman, Amsterdam, 2nd edition, 2006.

[Clark et al., 2004] T. Clark, A. Evans, P. Sammut, and J. Willans. Transformation language design: A metamodelling foundation. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *Graph Transformations*, volume 3256 of *Lecture Notes in Computer Science*, pages 13–21. Springer Berlin Heidelberg, 2004.

[Cleland-Huang and Schmelzer, 2003] J. Cleland-Huang and D. Schmelzer. Dynamically tracing non-functional requirements through design pattern invariants. In *Proc. of the International Workshop on Traceability in Emerging Forms of Software Engineering in Conjunction with ASE*, TEFSE '03, 2003.

[Cleland-Huang et al., 2005a] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhanskaya, and S. Christina. Goal-centric traceability for managing non-functional requirements. In *Proc. of the International Conference on Software Engineering*, ICSE '05, pages 362 – 371, 2005a.

[Cleland-Huang, 2005] J. Cleland-Huang. Toward improved traceability of non-functional requirements. In *Proc. of the International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '05, pages 14–19. IEEE Computer Society, 2005.

[Cleland-Huang, 2006] J. Cleland-Huang. Just enough requirements traceability. In *Proc. of the 30th Annual International Computer Software and Applications Conference - Volume 01*, COMPSAC '06, pages 41–42. IEEE Computer Society, 2006.

[Cleland-Huang, 2012] J. Cleland-Huang. Traceability in agile projects. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 265–275. Springer London, 2012.

[Cleland-Huang et al., 2002a] J. Cleland-Huang, C. K. Chang, and Y. Ge. Supporting event based traceability through high-level recognition of change events. In *Proc. of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*, COMPSAC '02, pages 595–602. IEEE Computer Society, 2002a.

[Cleland-Huang et al., 2002b] J. Cleland-Huang, C. K. Chang, G. Sethi, K. Javvaji, H. Hu, and J. Xia. Automating speculative queries through event-based requirements traceability. In *Proc. of the 10th Anniversary IEEE Joint International Requirements Engineering Conference*, RE '02, pages 289–298. IEEE Computer Society, 2002b.

[Cleland-Huang et al., 2003] J. Cleland-Huang, C. K. Chang, and M. Christensen. Event-based traceability for managing evolutionary change. *IEEE Transactions on Software Engineering*, 29(9):796–810, 2003.

[Cleland-Huang et al., 2005b] J. Cleland-Huang, R. Settimi, C. Duan, and X. Zou. Utilizing supporting evidence to improve dynamic requirements traceability. In *Proc. of the 13th IEEE International Requirements Engineering Conference*, RE '05, pages 135–144. IEEE Computer Society, 2005b.

[Cleland-Huang et al., 2007] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. Automated classification of non-functional requirements. *Requirements Engineering*, 12(2):103–120, 2007.

[Cleland-Huang et al., 2012] J. Cleland-Huang, M. Heimdahl, J. H. Hayes, R. Lutz, and P. Mäder. Trace queries for safety requirements in high assurance systems. In *Proc. of the 18th Working Conference on Requirements Engineering: Foundation for Software Quality*, REFSQ '12, pages 179–193, 2012.

[Costa and da Silva, 2007] M. Costa and A. R. da Silva. RT-MDD framework a practical approach. In *Proc. of the ECMDA Traceability Workshop*, ECMDA-TW '07, 2007.

[Cottenier et al., 2006] T. Cottenier, A. Berg, and T. Elrad. Modeling aspect-oriented compositions. In J.-M. Bruel, editor, *Satellite Events at the MoDELS 2005 Conference*, volume 3844 of *Lecture Notes in Computer Science*, pages 100–109. Springer Berlin Heidelberg, 2006.

[Cysneiros and do Prado Leite, 2004] L. M. Cysneiros and J. C. S. do Prado Leite. Nonfunctional requirements: from elicitation to conceptual models. *IEEE Transactions on Software Engineering*, 30(5):328–350, 2004.

[Cysneiros, 2007] L. M. Cysneiros. Evaluating the effectiveness of using catalogues to elicit non-functional requirements. In *Workshop em Engenharia de Requisitos*, WER '07, pages 107–115, 2007.

[Czarnecki and Helsen, 2006] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645, 2006.

[Czarnecki, 2002] K. Czarnecki. *Domain Engineering.* John Wiley & Sons, Inc., 2002.

[Dahlstedt and Persson, 2005] A. Dahlstedt and A. Persson. Requirements interdependencies: State of the art and future challenges. In A. Aurum and C. Wohlin, editors, *Engineering and Managing Software Requirements*, pages 95–116. Springer-Verlag, 2005.

[Dashofy et al., 2007] E. Dashofy, H. Asuncion, S. Hendrickson, G. Suryanarayana, J. Georgas, and R. Taylor. Archstudio 4: An architecture-based meta-modeling environment. In *Proc. of the 29th International Conference on Software Engineering - Companion*, ICSE Companion '07, pages 67–68, 2007.

[Dashofy et al., 2001] E. M. Dashofy, A. V. der Hoek, and R. N. Taylor. A highly-extensible, XML-based architecture description language. In *Proc. of the Working IEEE/IFIP Conference on Software Architecture*, WICSA '01, pages 103–112. IEEE Computer Society, 2001.

[de Pádua Albuquerque Oliveira et al., 2007] A. de Pádua Albuquerque Oliveira, J. C. S. do Prado Leite, L. M. Cysneiros, and C. Cappelli. Eliciting multi-agent systems intentionality: from language extended lexicon to i* models. In *Proc. of the 14th International Conference of the Chilean Computer Science Society*, SCCC '07, pages 40–49. IEEE Computer Society, 2007.

[Deerwester et al., 1990] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of The American Society for Information Science*, 41(6):391–407, 1990.

[DeRemer and Kron, 1975] F. DeRemer and H. Kron. Programming-in-the large versus programming-in-the-small. *SIGPLAN Not.*, 10(6):114–121, 1975.

[Despotou and Kelly, 2008] G. Despotou and T. Kelly. Investigating the use of argument modularity to optimise through-life system safety assurance. In *Proc. of the 3rd IET International Conference on System Safety*, pages 1–6, 2008.

[Diaz et al., 2013] D. Diaz, G. Bavota, A. Marcus, R. Oliveto, S. Takahashi, and A. D. Lucia. Using code ownership to improve ir-based traceability link recovery. In *Proc. of the 21st IEEE International Conference on Program Comprehension*, ICPC '13, pages 123–132. IEEE Computer Society, 2013.

[Dick, 2005] J. Dick. Rich traceability. In *Proc. of the International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '05, pages 62–66. IEEE Computer Society, 2005.

[Dömges and Pohl, 1998] R. Dömges and K. Pohl. Adapting traceability environments to project-specific needs. *Commun. ACM*, 41(12):54–62, 1998.

[Drivalos et al., 2009] N. Drivalos, D. S. Kolovos, R. F. Paige, and K. J. Fernandes. Engineering a dsl for software traceability. In D. Gašević, R. Lämmel, and E. Wyk, editors, *Software Language Engineering*, pages 151–167. Springer-Verlag, 2009.

[Drivalos-Matragkas et al., 2010] N. Drivalos-Matragkas, D. S. Kolovos, R. F. Paige, and K. J. Fernandes. A state-based approach to traceability maintenance. In *Proc. of the ECMFA Traceability Workshop*, ECMFA-TW '10, pages 23–30. ACM, 2010.

[Duan and Cleland-Huang, 2006] C. Duan and J. Cleland-Huang. Visualization and analysis in automated trace retrieval. In *Proc. of the 1st International Workshop on Requirements Engineering Visualization*, REV '06. IEEE Computer Society, 2006.

[Dutoit et al., 2006] A. H. Dutoit, R. McCall, I. Mistrik, and B. Paech. *Rationale Management in Software Engineering.* Springer-Verlag New York, Inc., 2006.

[Ebert et al., 2002] J. Ebert, B. Kullbach, V. Riediger, and A. Winter. Gupro: Generic understanding of programs – an overview. In *Electronic Notes in Theoretical Computer Science*, 2002.

[Ebner and Kaindl, 2002] G. Ebner and H. Kaindl. Tracing all around in reengineering. *IEEE Software*, 19(3):70–77, 2002.

[Eclipse Foundation, 2013] Eclipse Foundation. Eclipse Modelling Framework Project. `http://www.eclipse.org/modeling/emf`, 2013.

[Egyed and Grünbacher, 2005] A. Egyed and P. Grünbacher. Supporting software understanding with automated requirements traceability. *International Journal of Software Engineering and Knowledge Engineering*, 15(5):783–810, 2005.

[Egyed et al., 2005] A. Egyed, S. Biffl, M. Heindl, and P. Grünbacher. Determining the cost-quality trade-off for automated software traceability. In *Proc. of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ASE '05, pages 360–363. IEEE Computer Society, 2005.

[Egyed et al., 2007] A. Egyed, P. Grunbacher, M. Heindl, and S. Biffl. Value-based requirements traceability: Lessons learned. In *Proc. of the 15th IEEE International Requirements Engineering Conference*, RE '07, pages 240–257. IEEE Computer Society, 2007.

[Ellson et al., 2003] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull. Graphviz and dynagraph – static and dynamic graph drawing tools. In *Graph Drawing Software*, pages 127–148. Springer-Verlag, 2003.

[Engels et al., 2002] G. Engels, R. Heckel, J. M. Kuster, and L. Groenewegen. Consistency-preserving model evolution through transformations. In J.-M. Jezequel, H. Hussmann, and S. Cook, editors, *«UML» 2002 – The Unified Modelling Language*, volume 2460 of *Lecture Notes in Computer Science*, pages 212–227. Springer Berlin Heidelberg, 2002.

[Espinoza and Garbajosa, 2008a] A. Espinoza and J. Garbajosa. A proposal for defining a set of basic items for project-specific traceability methodologies. In *Proc. of the 32nd Annual IEEE Software Engineering Workshop*, SEW '08, pages 175–184. IEEE Computer Society, 2008a.

[Espinoza and Garbajosa, 2008b] A. Espinoza and J. Garbajosa. Tackling traceability challenges through modelling principles in methodologies underpinned by metamodels. In *Proc. of the CEE-SET WiP*, CEE-SET WiP '08, pages 41–54. IEEE Computer Society, 2008b.

[Espinoza and Garbajosa, 2011] A. Espinoza and J. Garbajosa. A study to support agile methods more effectively through traceability. *Innovations in Systems and Software Engineering*, 7(1):53–69, 2011.

[Espinoza et al., 2006] A. Espinoza, P. Alarcon, and J. Garbajosa. Analyzing and systematizing current traceability schemas. In *Proc. of the 30th Software Engineering Workshop*, 2006.

[EUROCONTROL, 1999a] EUROCONTROL. European Reduced Vertical Separation Minimum (RVSM) Programme. `http://www.eurocontrol.int/eur-rma`, 1999a.

[EUROCONTROL, 1999b] EUROCONTROL. Reduced Vertical Separation Minimum (RVSM) Master Plan. `www.seguridadaerea.gob.es/media/Migracion/PDF/89756/2032.pdf`, 1999b.

[EUROCONTROL, 2000] EUROCONTROL. AFI Reduced Vertical Separation Minimum (RVSM) Safety Policy (1^{st} Edition), 2000.

[EUROCONTROL, 2001a] EUROCONTROL. ATC Manual for a Reduced Vertical Separation Minimum (RVSM) in Europe (2^{nd} Edition). `www.code-team.com/mavien/RVSM_Manual.pdf`, 2001a.

[EUROCONTROL, 2001b] EUROCONTROL. The EUR RVSM Functional Hazard Assessment, 2001b.

[EUROCONTROL, 2001c] EUROCONTROL. The EUR RVSM Pre-Implementation Safety Case. `dependability.cs.virginia.edu/research/safetycases/EUR_RVSM.pdf`, 2001c.

[Fabro et al., 2005] M. D. D. Fabro, J. Bézivin, F. Jouault, E. Breton, and G. Gueltas. AMW: A generic model weaver. In *Proc. of the 1éré Journéees sur l'Ingénierie Dirigée par les Modéles*, IDM '05, 2005.

[Falleri et al., 2006] J.-R. Falleri, M. Huchard, and C. Nebut. Towards a traceability framework for model transformations in kermeta. In *Proc. of the ECMDA Traceability Workshop*, ECMDA-TW '06, 2006.

[Federation of EA Professional Organizations, 2013] Federation of EA Professional Organizations. Common perspectives on enterprise architecture. *Architecture and Governance Magazine*, 4, 2013.

[Finkelstein and Dowell, 1996] A. Finkelstein and J. Dowell. A comedy of errors: the London Ambulance Service case study. In *Proc. the 8th IEEE International Workshop on Software Specification and Design*, IWSSD '96. IEEE Computer Society, 1996.

[Francis et al., 2013] M. Francis, D. Kolovos, N. Matragkas, and R. Paige. Adding spreadsheets to the MDE toolbox. In *Proc. of the 16th ACM/IEEE International Conference on Model Driven Engineering Languages & Systems*, MoDELS '13, 2013.

[Galvao and Goknil, 2007] I. Galvao and A. Goknil. Survey of traceability approaches in model-driven engineering. In *Proc. of the 11th IEEE International Enterprise Distributed Object Computing Conference*, EDOC '07, pages 313–324. IEEE Computer Society, 2007.

[Garcí-Domíguez et al., 2011] A. Garcí-Domíguez, D. S. Kolovos, L. M. Rose, R. F. Paige, and I. Medina-Bulo. Eunit: A unit testing framework for model management tasks. In J. Whittle, T. Clark, and T. Kuhne, editors, *Model Driven Engineering Languages and Systems*, volume 6981 of *Lecture Notes in Computer Science*, pages 395–409. Springer Berlin Heidelberg, 2011.

[Ge et al., 2010] X. Ge, R. F. Paige, and J. A. McDermid. An iterative approach for development of safety-critical software and safety arguments. In *Proc. of Agile Conference*, Agile '10, pages 35–43, 2010.

[Gethers et al., 2011] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. Lucia. On integrating orthogonal information retrieval methods to improve traceability recovery. In *Proc. of the 27th IEEE International Conference on Software Maintenance*, ICSM '11, pages 133–142, 2011.

[Gitzel and Korthaus, 2004] R. Gitzel and A. Korthaus. The role of metamodeling in model-driven development. In *Proc. of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics*, SCI '04, 2004.

[Gorp and Janssens, 2005] P. V. Gorp and D. Janssens. Cavit: a consistency maintenance framework based on visual model transformation

and transformation contracts. In *Transformation Techniques in Software Engineering, number 05161 in Dagstuhl Seminar Proc. of the Internationales Begegnungs- und Forschungszentrum fu Informatik (IBFI), Schloss Dagstuhl*, 2005.

[Gotel and Finkelstein, 1994] O. C. Z. Gotel and C. W. Finkelstein. An analysis of the requirements traceability problem. In *Proc. of the 1st IEEE International Requirements Engineering Conference*, RE '94, pages 94 –101. IEEE Computer Society, 1994.

[Gotel and Mäder, 2012] O. Gotel and P. Mäder. Acquiring tool support for traceability. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 43–68. Springer London, 2012.

[Gotel et al., 2012a] O. Gotel, J. Cleland-Huang, J. Hayes, A. Zisman, A. Egyed, P. Grunbacher, A. Dekhtyar, G. Antoniol, and J. Maletic. The grand challenge of traceability (v1.0). In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 343–409. Springer London, 2012a.

[Gotel et al., 2012b] O. Gotel, J. Cleland-Huang, J. Hayes, A. Zisman, A. Egyed, P. Grunbacher, A. Dekhtyar, G. Antoniol, J. Maletic, and P. Mäder. Traceability fundamentals. In O. Gotel, J. Cleland-Huang, and A. Zisman, editors, *Software and Systems Traceability*, pages 3–22. Springer London, 2012b.

[Gotel et al., 2012c] O. Gotel, J. Cleland-Huang, and A. Zisman, editors. *Software and Systems Traceability*. Springer London, 2012c.

[Grammel and Vigot, 2009] B. Grammel and K. Vigot. Foundations for a generic traceability framework in model-driven software engineering. In *Proc. of the ECMDA Traceability Workshop*, ECMDA-TW '09, 2009.

[Grechanik et al., 2007] M. Grechanik, K. S. McKinley, and D. E. Perry. Recovering and using use-case-diagram-to-source-code traceability links. In *Proc. of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC-FSE '07, pages 95–104. ACM, 2007.

[Gruschko et al., 2007] B. Gruschko, D. S. Kolovos, and R. F. Paige. Towards synchronizing models with evolving metamodels. In *Proc. of the International Workshop on Model-Driven Software Evolution held with the ECSMR*, MDSE '07, 2007.

[Guerra et al., 2010] E. Guerra, J. de Lara, D. S. Kolovos, R. F. Paige, and O. M. dos Santos. transml: A family of languages to model model transformations. In *Proc. of the 13th International Conference on Model Driven*

Engineering Languages and Systems: Part I, MODELS '10, pages 106–120. Springer-Verlag, 2010.

[Haarslev et al., 2004] V. Haarslev, R. Möler, and M. Wessel. Querying the semantic web with Racer + nRQL. In *Proc. of the KI International Workshop on Applications of Description Logics*, ADL '04, 2004.

[Habli and Kelly, 2006] I. Habli and T. Kelly. Process and product certification arguments: Getting the balance right. *SIGBED Rev.*, 3(4):1–8, 2006.

[Hawkins and Kelly, 2009] R. Hawkins and T. Kelly. Software safety assurance - what is sufficient? In *Proc. of the IET System Safety Conference*. IEEE Computer Society, 2009.

[Hayes et al., 2004] J. H. Hayes, A. Dekhtyar, S. Sundaram, and S. Howard. Helping analysts trace requirements: An objective look. In *Proc. of the 12th IEEE International Requirements Engineering Conference*, RE '04, pages 249–259. IEEE Computer Society, 2004.

[Hayes and Dekhtyar, 2005] J. H. Hayes and A. Dekhtyar. Humans in the traceability loop: Can't live with 'em, can't live without 'em. In *Proc. of the International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '05, pages 20–23. IEEE Computer Society, 2005.

[Hayes et al., 2003] J. H. Hayes, A. Dekhtyar, and J. Osborne. Improving requirements tracing via information retrieval. In *Proc. of the 11th IEEE International Requirements Engineering Conference*, RE '03, pages 138–. IEEE Computer Society, 2003.

[Hayes et al., 2006] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Transactions on Software Engineering*, 32(1):4–19, 2006.

[Heidenreich et al., 2009] F. Heidenreich, J. Johannes, S. Karol, M. Seifert, and C. Wende. Derivation and refinement of textual syntax for models. In R. F. Paige, A. Hartman, and A. Rensink, editors, *Model Driven Architecture - Foundations and Applications*, volume 5562 of *Lecture Notes in Computer Science*, pages 114–129. Springer Berlin Heidelberg, 2009.

[Heim et al., 2008] P. Heim, S. Lohmann, K. Lauenroth, and J. Ziegler. Graph-based visualization of requirements relationships. In *Proc. of the 3rd International Workshop on Requirements Engineering Visualization*, REV '08, pages 51–55. IEEE Computer Society, 2008.

[Heindl and Biffl, 2005] M. Heindl and S. Biffl. A case study on value-based requirements tracing. In *Proc. of the 10th European Software Engineering*

Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ESEC/FSE '13, pages 60–69. ACM, 2005.

[Henderson, 2003] P. B. Henderson. The role of modeling in software engineering education. In *Proc. of the 33rd ASEE/IEEE Frontiers in Education Conference*, 2003.

[Hoffmann et al., 2004] M. Hoffmann, N. Kuhn, and M. Weber. Requirements for requirements management tools. In *Proc. of the IEEE International Requirements Engineering Conference*, RE '04, pages 301–308. IEEE Computer Society, 2004.

[IBM, 2014a] IBM. Rational DOORS. `http://www-03.ibm.com/software/products/en/ratidoor`, 2014a.

[IBM, 2014b] IBM. IBM Rationa RequisitePro. `http://www-03.ibm.com/software/products/en/reqpro`, 2014b.

[ICAO, 2002] ICAO. ICAO Document 9574 (2^{nd} Edition) - Manual on Implementation of a 300M (1000 FT) Vertical Separation Minimum between FL290 and FL410 inclusive. `http://www.skybrary.aero/bookshelf/content/bookDetails.php?bookId=1311`, 2002.

[IEC, 2005] IEC. IEC/TR 61508 - Functional safety of electrical/electronic/programmable electronic safety-related systems. `http://www.iec.ch/functionalsafety/standards/page3.htm`, 2005.

[IEC, 2008] IEC. IEC 60880. Nuclear power plants instrumentation and control systems important to safety software aspects for computer-based systems performing category A functions. `http://webstore.iec.ch/preview/info_iec60880%7Bed2.0%7Db.pdf`, 2008.

[IEC, 2012] IEC. `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=60555`, 2012.

[IEEE, 1984] IEEE. *IEEE Guide to Software Requirements Specification*. The Institute of Electrical and Electronics Engineers: New York, 1984.

[IEEE, 1990] IEEE. *IEEE Standard Glossary of Software Engineering Terminology*. The Institute of Electrical and Electronics Engineers: New York, 1990.

[INCOSE, 2010] INCOSE. Tools database working group (tdwg). international council on systems engineering requirements (incose) management tools survey. `http://www.incose.org/ProductsPubs/products/rmsurvey.aspx`, 2010.

[Information Technology ISO/IEC, 2002] Information Technology ISO/IEC. Z formal specification notation - syntax, type system and semantics, 2002.

[Information Technology ISO/IEC, 2007] Information Technology ISO/IEC. Acceleo Pro Traceability. `http://www.obeo.fr/pages/obeo-traceability/en`, 2007.

[Ingram and Riddle, 2012] C. Ingram and S. Riddle. Cost-benefits of traceability. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 23–42. Springer, 2012.

[Integrate, 2014] Integrate. Traceline for doors. `http://www.integrate.biz/traceline/`, 2014.

[ISO, 2011] ISO. Road vehicles – Functional safety. `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=43464`, 2011.

[ISO/IEC, 2007] ISO/IEC. ISO/IEC 24744:2007 Software Engineering - Metamodel for Development Methodologies, 2007.

[ISO/IEC and ITU-T, 1998] ISO/IEC and ITU-T. The Reference Model of Open Distributed Processing: ITU-T Rec. X.901-X.904 — ISO/IEC 10746. `http://www.rm-odp.net/`, 1998.

[ISO/IEC and ITU-T, 2009] ISO/IEC and ITU-T. Information technology - Open distributed processing - Use of UML for ODP system specifications: ITU-T Rec. X.906 — ISO/IEC 19793. `http://www.lcc.uma.es/~av/download/UML4ODP_IS_V2.pdf`, 2009.

[Jarke and Pohl, 1992] M. Jarke and K. Pohl. Information systems quality and quality informations systems. In *Proc. the IFIP WG8.2 Working Conference on The Impact of Computer Supported Technologies in Information Systems Development*, pages 345–375. North-Holland Publishing Co., 1992.

[Jirapanthong and Zisman, 2005] W. Jirapanthong and A. Zisman. Supporting product line development through traceability. In *Proc. of the 12th Asia-Pacific Software Engineering Conference*, APSEC '05. IEEE Computer Society, 2005.

[Jirapanthong, 2007] W. Jirapanthong. Techniques and approaches for developing software product line. In H. R. Arabnia and H. Reza, editors, *Proc. of the International Conference on Software Engineering Research & Practice*, SERP '07, pages 276–281. CSREA Press, 2007.

[Jouault, 2005] F. Jouault. Loosely coupled traceability for ATL. In *Proc. of the ECMDA Traceability Workshop*, ECMDA-TW '05, 2005.

[Jouault et al., 2006] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, and P. Valduriez. ATL: A QVT-like transformation language. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*, OOPSLA '06, pages 719–720. ACM, 2006.

[Kagdi et al., 2007] H. Kagdi, J. I. Maletic, and B. Sharif. Mining software repositories for traceability links. In *Proc. of the 15th IEEE International Conference on Program Comprehension*, ICPC '07, pages 145–154. IEEE Computer Society, 2007.

[Kannenberg and Saiedian, 2009] A. Kannenberg and H. Saiedian. Why software requirements traceability remains a challenge. *The Journal of Defense Software Engineering*, 22:14–19, 2009.

[Kassab and Ormandjieva, 2006] M. Kassab and O. Ormandjieva. Towards an aspect oriented software development model with tractability mechanism. In *Proc. of the Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design*, AOSD '06, 2006.

[Kassab et al., 2009] M. Kassab, O. Ormandjieva, and M. Daneva. A metamodel for tracing non-functional requirements. In *Proc. of the WRI World Congress on Computer Science and Information Engineering - Volume 07*, CSIE '09, pages 687–694. IEEE Computer Society, 2009.

[Katta and Stlhane, 2012] V. Katta and T. Stlhane. A conceptual model of traceability for safety systems. Technical report, Laboratory of Algorithmics, Complexity and Logic, 2012.

[Kazman et al., 2000] R. Kazman, M. H. Klein, and P. C. Clements. Atam: Method for architecture evaluation. Technical report, Software Engineering Institute (SEI), 2000.

[Kelly and Weaver, 2004] T. Kelly and R. Weaver. The goal structuring notation - a safety argument notation. *Elements*, 2004.

[Kelly, 1998] T. P. Kelly. *Arguing Safety – A Systematic Approach to Managing Safety Cases*. PhD thesis, Department of Computer Science, University of York, 1998.

[Kirova et al., 2008] V. Kirova, N. Kirby, D. Kothari, and G. Childress. Effective requirements traceability: Models, tools, and practices. *Bell Labs Technical Journal*, 12:143–157, 2008.

[Kleppe et al., 2003] A. G. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[Kolovos et al., 2010] D. Kolovos, L. Rose, and R. Paige. *The Epsilon Book*. Eclipse, 2010.

[Kolovos, 2007] D. S. Kolovos. Editing emf models with exeed (extended emf editor). Technical report, Department of Computer Science, University of York, 2007.

[Kolovos, 2008] D. S. Kolovos. *An Extensible Platform for Specification of Integrated Languages for Model Management*. PhD thesis, University of York, 2008.

[Kolovos, 2009] D. S. Kolovos. Establishing correspondences between models with the Epsilon Comparison Language. In R. F. Paige, A. Hartman, and A. Rensink, editors, *Model Driven Architecture - Foundations and Applications*, volume 5562 of *Lecture Notes in Computer Science*, pages 146–157. Springer Berlin Heidelberg, 2009.

[Kolovos and Paige, 2013] D. S. Kolovos and R. F. Paige. Extensible Platform for Specification of Integrated Languages for mOdel maNagement (Epsilon). `http://www.eclipse.org/gmt/epsilon`, 2013.

[Kolovos et al., 2006a] D. S. Kolovos, R. F. Paige, and F. A. C. Polack. Eclipse development tools for Epsilon. In *Eclipse Summit Europe, Eclipse Modeling Symposium*, 2006a.

[Kolovos et al., 2006b] D. S. Kolovos, R. F. Paige, and F. A. C. Polack. The Epsilon Object Language (EOL). In *Proc. of the 2nd European conference on Model Driven Architecture: foundations and Applications*, ECMDA-FA '06, pages 128–142. Springer-Verlag, 2006b.

[Kolovos et al., 2006c] D. S. Kolovos, R. F. Paige, and F. A. Polack. Merging models with the Epsilon Merging Language (EML). In O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, editors, *Model Driven Engineering Languages and Systems*, volume 4199 of *Lecture Notes in Computer Science*, pages 215–229. Springer Berlin Heidelberg, 2006c.

[Kolovos et al., 2007] D. S. Kolovos, R. F. Paige, F. A. Polack, and L. M. Rose. Update transformations in the small with the Epsilon Wizard Language. *Journal of Object Technology*, 6(9):53–69, 2007.

[Kolovos et al., 2008] D. S. Kolovos, R. F. Paige, and F. A. Polack. The Epsilon Transformation Language. In *Proc. of the 1st International Conference on Theory and Practice of Model Transformations*, ICMT '08, pages 46–60. Springer-Verlag, 2008.

[Kolovos et al., 2009] D. S. Kolovos, R. F. Paige, and F. A. Polack. On the evolution of OCL for capturing structural constraints in modelling languages.

In J.-R. Abrial and U. Glässer, editors, *Rigorous Methods for Software Construction and Analysis*, volume 5115 of *Lecture Notes in Computer Science*, pages 204–218. Springer Berlin Heidelberg, 2009.

[Krishna and Gregoriades, 2011] A. Krishna and A. Gregoriades. Extending UML with non-functional requirements modelling. In J. Pokorny, V. Repa, K. Richta, W. Wojtkowski, H. Linger, C. Barry, and M. Lang, editors, *Information Systems Development*, pages 357–372. Springer New York, 2011.

[Kurtev, 2004] I. Kurtev. *Adaptability of Model Transformations*. PhD thesis, University of Twente, 2004.

[Kurtev et al., 2007] I. Kurtev, M. Dee, A. Göknil, and K. van den Berg. Traceability-based change management in operational mappings. In *Proc. of the ECMDA Traceability Workshop*, pages 57–67. SINTEF, 2007.

[Lago et al., 2009] P. Lago, H. Muccini, and H. van Vliet. A scoped approach to traceability management. *Journal of Systems and Software*, 82(1):168–182, 2009.

[Lee et al., 2003] C. Lee, L. Guadagno, and X. Jia. An agile approach to capturing requirements and traceability. In *Proc. of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering*, pages 17–23, 2003.

[Letelier, 2002] P. Letelier. A framework for requirements traceability in UML-based projects. In *Proc. of the Intlernational Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '02, pages 32–41. IEEE Computer Society, 2002.

[Li and Maalej, 2012] Y. Li and W. Maalej. Which traceability visualization is suitable in this context? a comparative study. In *Proc. of the 18th International Conference on Requirements Engineering: Foundation for Software Quality*, REFSQ '12, pages 194–210. Springer-Verlag, 2012.

[Limon and Garbajosa, 2005] A. E. Limon and Garbajosa. The need for a unifying traceability scheme. In *Proc. of the ECMDA Traceability Workshop*, ECMDA-TW '05, 2005.

[Lin et al., 2006] J. Lin, C. C. Lin, J. Cleland-Huang, R. Settimi, J. Amaya, G. Bedford, B. Berenbach, O. B. Khadra, C. Duan, and X. Zou. Poirot: A distributed tool supporting enterprise-wide automated traceability. In *Proc. of the 14th IEEE International Requirements Engineering Conference*, RE '06, pages 356–357. IEEE Computer Society, 2006.

[Lindvall and Sandahl, 1996] M. Lindvall and K. Sandahl. Practical implications of traceability. *Software: Practice and Experience*, 26(10):1161–1180, 1996.

[Lucia et al., 2004] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora. Enhancing an artefact management system with traceability recovery features. In *Proc. of the 20th IEEE International Conference on Software Maintenance*, ICSM '04, pages 306–315. IEEE Computer Society, 2004.

[Lucia et al., 2006a] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora. Can information retrieval techniques effectively support traceability link recovery? In *Proc. of the 14th IEEE International Conference on Program Comprehension*, ICPC '06, pages 307–316. IEEE Computer Society, 2006a.

[Lucia et al., 2006b] A. D. Lucia, R. Oliveto, and P. Sgueglia. Incremental approach and user feedbacks: a silver bullet for traceability recovery. In *Proc. of the 22nd IEEE International Conference on Software Maintenance*, ICSM '06, pages 299–309. IEEE Computer Society, 2006b.

[Lucia et al., 2008] A. D. Lucia, R. Oliveto, and G. Tortora. IR-based traceability recovery processes: An empirical comparison of "One-Shot" and incremental processes. In *Proc. of the 23rd IEEE/ACM International Conference on Automated Software Engineering*, ASE '08, pages 39–48. IEEE Computer Society, 2008.

[Lucia et al., 2009] A. D. Lucia, R. Oliveto, and G. Tortora. Assessing ir-based traceability recovery tools through controlled experiments. *Journal of Empirical Software Engineering*, 14(1):57–92, 2009.

[Lucia et al., 2011] A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella. Improving ir-based traceability recovery using smoothing filters. In *Proc. of the 19th IEEE International Conference on Program Comprehension*, ICPC '11, pages 21–30. IEEE Computer Society, 2011.

[Lucia et al., 2012] A. Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk. Information retrieval methods for automated traceability recovery. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 71–98. Springer London, 2012.

[Lucia et al., 2007] 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 Methodology*, 16(4), 2007.

[Ludewig, 2003] J. Ludewig. Models in software engineering – an introduction. *Software and Systems Modeling*, 2(1):5–14, 2003.

[Lutz, 2000] R. R. Lutz. Software engineering for safety: A roadmap. In *Proc. of the Conference on The Future of Software Engineering*, ICSE '00, pages 213–226. ACM, 2000.

[Mäder et al., 2007] P. Mäder, I. Philippow, and M. Riebisch. Customizing traceability links for the unified process. In *Proc. of the Quality of Software Architectures 3rd International Conference on Software Architectures, Components, and Applications*, QoSA '07, pages 53–71. Springer-Verlag, 2007.

[Mäder et al., 2008a] P. Mäder, O. Gotel, and I. Philippow. Enabling automated traceability maintenance by recognizing development activities applied to models. In *Proc. of the 23rd IEEE/ACM International Conference on Automated Software Engineering*, ASE '08, pages 49–58. IEEE Computer Society, 2008a.

[Mäder and Cleland-Huang, 2010] P. Mäder and J. Cleland-Huang. A visual traceability modeling language. In *Proc. of the 13th International Conference on Model Driven Engineering Languages and Systems*, MODELS '10, pages 226–240, 2010.

[Mäder et al., 2008b] P. Mäder, O. Gotel, and I. Philippow. Rule-based maintenance of post-requirements traceability relations. In *Proc. of the 16th IEEE International Requirements Engineering Conference*, RE '08, pages 23–32. IEEE Computer Society, 2008b.

[Mäder et al., 2009a] P. Mäder, O. Gotel, and I. Philippow. Getting back to basics: Promoting the use of a traceability information model in practice. In *Proc. of the International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '09, pages 21–25. IEEE Computer Society, 2009a.

[Mäder et al., 2009b] P. Mäder, O. Gotel, and I. Philippow. Motivation matters in the traceability trenches. In *Proc. of the 17th IEEE International Requirements Engineering Conference*, RE '09, pages 143–148. IEEE Computer Society, 2009b.

[Maletic and Collard, 2009] J. I. Maletic and M. L. Collard. TQL: A query language to support traceability. In *Proc. of the International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '09, pages 16–20. IEEE Computer Society, 2009.

[Maletic et al., 2003] J. I. Maletic, E. V. Munson, A. Marcus, and T. N. Nguyen. Using a hypertext model for traceability link conformance analysis. In *Proc. of the International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '03, pages 47–54. IEEE Computer Society, 2003.

[Maletic et al., 2005] J. I. Maletic, M. L. Collard, and B. Simoes. An XML based approach to support the evolution of model-to-model traceability links. In *Proc. of the International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '05, pages 67–72. IEEE Computer Society, 2005.

[Marasco, 2006] J. Marasco. Software development productivity and project success rates: Are we attacking the right problem? `http://www.cs.st-andrews.ac.uk/~ifs/Talks/IEEInaugural.pdf`, 2006.

[Marcus and Maletic, 2003] A. Marcus and J. I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proc. of the 25th International Conference on Software Engineering*, pages 125–135, 2003.

[Marcus et al., 2005] A. Marcus, X. Xie, and D. Poshyvanyk. When and how to visualize traceability links? In *Proc. of the International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '05, pages 56–61. IEEE Computer Society, 2005.

[Mason, 2005] P. Mason. On traceability for safety critical systems engineering. In *Proc. of the 12th Asia-Pacific Software Engineering Conference*, APSEC' 05, pages 272–282. IEEE Computer Society, 2005.

[Mason et al., 2003] P. Mason, A. Saeed, P. Arkely, and S. Riddle. Metamodelling approach to traceability for avionics: A framework for managing the engineering of computer based aerospace systems. In *Proc. of the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, ECBS '03, pages 233–246. IEEE Computer Society, 2003.

[Matragkas, 2011] N. Matragkas. *Establishing and Maintaining Semantically Rich Traceability: A Metamodelling Approach.* PhD thesis, University of York, 2011.

[Matula, 2003] M. Matula. Netbeans metadata repository. `netbeans-uml-extender-plugin.googlecode.com/files/MDR-whitepaper.pdf`, 2003.

[McMillan et al., 2009] C. McMillan, D. Poshyvanyk, and M. Revelle. Combining textual and structural analysis of software artifacts for traceability link recovery. In *Proc. of the International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '09, pages 41–48. IEEE Computer Society, 2009.

[Mens and Demeyer, 2008] T. Mens and S. Demeyer. *Software Evolution.* Springer-Verlag, 2008.

[Mens et al., 2005] T. Mens, R. V. D. Straeten, and J. Simmonds. A framework for managing consistency of evolving UML models. In H. Yang, editor, *Software Evolution with UML and XML*, pages 1–31. Idea Group Publishing, 2005.

[Merten et al., 2011] T. Merten, D. Juppner, and A. Delater. Improved representation of traceability links in requirements engineering knowledge using Sunburst and Netmap visualizations. In *Proc. of the 4th International Workshop on Managing Requirements Knowledge*, MARK '11, pages 17–21, 2011.

[Mirakhorli and Cleland-Huang, 2011] M. Mirakhorli and J. Cleland-Huang. Using tactic traceability information models to reduce the risk of architectural degradation during system maintenance. In *Proc. of the 27th IEEE International Conference on Software Maintenance*, ICSM '11, pages 123–132. IEEE Computer Society, 2011.

[Mirakhorli and Cleland-Huang, 2012] M. Mirakhorli and J. Cleland-Huang. Tracing non-functional requirements. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 299–320. Springer London, 2012.

[Munson and Nguyen, 2005] E. V. Munson and T. N. Nguyen. Concordance, conformance, versions, and traceability. In *Proc. of the International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '05, pages 62–66. IEEE Computer Society, 2005.

[Murta et al., 2006] L. G. P. Murta, A. van der Hoek, and C. M. L. Werner. ArchTrace: Policy-based support for managing evolving architecture-to-implementation traceability links. In *Proc. of the 21st IEEE/ACM International Conference on Automated Software Engineering*, ASE '06, pages 135–144. IEEE Computer Society, 2006.

[Nentwich et al., 2002] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelsteiin. Xlinkit: A consistency checking and smart link generation service. *ACM Trans. Internet Technol.*, 2(2):151–185, 2002.

[Nuseibeh et al., 1994] B. Nuseibeh, J. Kramer, and A. Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering*, 20(10):760–773, 1994.

[Nuseibeh and Easterbrook, 2000] B. Nuseibeh and S. Easterbrook. Requirements engineering: A roadmap. In *Proc. of the International Conference on Software Engineering*, ICSE '00, pages 35–46. ACM, 2000.

[Object Management Group, 2003] Object Management Group. OMG. Model Driven Architecture (MDA) Guide 1.0.1. http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf, 2003.

[Object Management Group, 2005] Object Management Group. A proposal for an MDA foundation model, ormsc/05-04-01. Technical report, Object Management Group, 2005. URL http://www.omg.org/cgi-bin/doc?ormsc/05-04-01.

[Object Management Group, 2007] Object Management Group. OMG. MOF 2.0/XMI Mapping, v2.1.1 . http://www.omg.org/spec/XMI/2.1.1/PDF/index.htm, 2007.

[Object Management Group, 2008] Object Management Group. MOF Model to Text Transformation Language, v1.0. http://www.omg.org/spec/MOFM2T/1.0, 2008.

[Object Management Group, 2010a] Object Management Group. OMG. Systems Modeling Language 1.2. http://www.omg.org/spec/SysML/1.2/PDF, 2010a.

[Object Management Group, 2010b] Object Management Group. OMG. Unified Modeling Language 2.3 Infrastructure. http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/, 2010b.

[Object Management Group, 2011a] Object Management Group. The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems. http://www.omgmarte.org/, 2011a.

[Object Management Group, 2011b] Object Management Group. OMG. Meta Object Facility 2.4.1. http://www.omg.org/spec/MOF/2.4.1/PDF/, 2011b.

[Object Management Group, 2011c] Object Management Group. MOF Query/View/Transformation Specification, v1.1. http://www.omg.org/spec/QVT/1.1, 2011c.

[Object Management Group, 2012] Object Management Group. OMG. Object Constraint Language 2.3.1. http://www.omg.org/spec/OCL/2.3.1/PDF/, 2012.

[Oldevik and Neple, 2006] J. Oldevik and T. Neple. Traceability in model-to-text transformation. In *Proc. of the ECMDA Traceability Workshop*, ECMDA-TW '06, pages 64–69, 2006.

[Oldevik et al., 2005] J. Oldevik, T. Neple, R. Grønmo, J. Aagedal, and A.-J. Berre. Toward standardised model to text transformations. In *Proc. of the 1st European Conference on Model Driven Architecture: Foundations and Applications*, ECMDA-FA '05, pages 239–253. Springer-Verlag, 2005.

[Oliveto et al., 2010] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. D. Lucia. On the equivalence of information retrieval methods for automated traceability link recovery. In *Proc. of the 18th IEEE International Conference on Program Comprehension*, ICPC '10, pages 68–71. IEEE Computer Society, 2010.

[Olsen and Oldevik, 2007] G. K. Olsen and J. Oldevik. Scenarios of traceability in model to text transformations. In *Proc. of the 3rd European Conference on Model Driven Architecture-Foundations and Applications*, ECMDA-FA' 07, pages 144–156. Springer-Verlag, 2007.

[Osterbye and Wiil, 1996] K. Osterbye and U. K. Wiil. The flag taxonomy of open hypermedia systems. In *Proc. of the 7th ACM Conference on Hypertext*, pages 129–139. ACM, 1996.

[Ozkaya, 2006] I. Ozkaya. Representing requirement relationships. In *Proc. of the 1st International Workshop on Requirements Engineering Visualization*, REV '06, page 3, 2006.

[Paige et al., 2008] R. F. Paige, G. K. Olsen, D. S. Kolovos, S. Zschaler, and C. Power. Building model-driven engineering traceability classifications. In *Proc. of the ECMDA Traceability Workshop*, ECMDA-TW '08, pages 49–58. Sintef, 2008.

[Paige et al., 2007] R. F. Paige, P. J. Brooke, and J. S. Ostroff. Metamodel-based model conformance and multi-view consistency checking. *ACM Transactions on Software Engineering and Methodolology*, 16, 2007.

[Paige et al., 2011] R. F. Paige, A. Galloway, R. Charalambous, X. Ge, and P. J. Brooke. High-integrity agile processes for the development of safety critical software. *International Journal of Critical Computer-Based Systems*, 2:181–216, 2011.

[Palmer, 1999] J. D. Palmer. Traceability. In R. H. Thayer and M. Dorfman, editors, *Software Requirements Engineering, 2nd*, pages 412–422. IEEE Computer Society Press, 1999.

[Panesar-Walawege et al., 2010] R. K. Panesar-Walawege, M. Sabetzadeh, L. Briand, and T. Coq. Characterizing the chain of evidence for software safety cases: A conceptual model based on the IEC 61508 standard. In *Proc. of the 3rd International Conference on Software Testing, Verification and Validation*, ICST '10, pages 335–344, 2010.

[Parr, 2007] T. Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Programmers. Pragmatic Bookshelf, 1st edition, 2007.

[Peraldi-Frati and Albinet, 2010] M.-A. Peraldi-Frati and A. Albinet. Requirement traceability in safety critical systems. In *Proc. of the 1st Workshop on Critical Automotive applications: Robustness & Safety*, CARS '10, pages 11–14. ACM, 2010.

[Pilato, 2004] M. Pilato. *Version Control With Subversion*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004.

[Pilgrim et al., 2008] J. Pilgrim, B. Vanhooff, I. Schulz-Gerlach, and Y. Berbers. Constructing and visualizing transformation chains. In I. Schieferdecker and A. Hartman, editors, *Model Driven Architecture Foundations and Applications*, volume 5095 of *Lecture Notes in Computer Science*, pages 17–32. Springer Berlin Heidelberg, 2008.

[Pinheiro, 2003] F. A. C. Pinheiro. Requirements traceability. In S. do Prado, J. Leite, and J. Doorn, editors, *Perspectives on Software Requirements*, Lecture Notes in Business Information Processing, pages 93–103. Springer, Berlin, 2003.

[Pinheiro and Goguen, 1996] F. A. C. Pinheiro and J. A. Goguen. An object-oriented tool for tracing requirements. *IEEE Software*, 13(2):52–64, 1996.

[Pohl, 1996a] K. Pohl. PRO-ART: Enabling requirements pre-traceability. In *Proc. of the 2nd IEEE International Requirements Engineering Conference*, RE '96, pages 76–84. IEEE Computer Society, 1996a.

[Pohl, 1996b] K. Pohl. *Process-Centered Requirements Engineering*. John Wiley & Sons, Inc., 1996b.

[Porter, 1980] M. Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.

[Pottinger and Bernstein, 2003] R. Pottinger and P. A. Bernstein. Merging models based on given correspondences. In *Proc. of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pages 862–873, 2003.

[Potts and Bruns, 1988] C. Potts and G. Bruns. Recording the reasons for design decisions. In *Proc. the 10th International Conference on Software Engineering*, ICSE '88, pages 418–427, 1988.

[pure-systems GmbH, 2014] pure-systems GmbH. pure::variants. `http://www.pure-systems.com/Variant_Management.49.0.html`, 2014.

[QuEST, 1998] QuEST. *TL 9000 Quality Management System*. Quest, 1998.

[Ramamoorthy et al., 1988] C. V. Ramamoorthy, V. Garg, and A. Prakash. Support for reusability in Genesis. *IEEE Transactions on Software Engineering*, 14:1145–1154, 1988.

[Ramesh and Edwards, 1993] B. Ramesh and M. Edwards. Issues in the development of a requirements traceability model. In *Proc. the IEEE International Symposium on Requirements Engineering*, pages 256 –259, 1993.

[Ramesh and Jarke, 2001] B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, 27:58–93, 2001.

[Ramsin and Paige, 2008] R. Ramsin and R. F. Paige. Process-centered review of object oriented software development methodologies. *ACM Computer Survey*, 40(1):3:1–3:89, 2008.

[Ratanotayanon et al., 2009] S. Ratanotayanon, S. E. Sim, and D. J. Raycraft. Cross-artifact traceability using lightweight links. In *Proc. of the International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '09, pages 57–64. IEEE Computer Society, 2009.

[Robinsons, 2014] Robinsons. Doctrace - requirements traceability tool. `http://www.robinsons.co.uk/doctrace.html`, 2014.

[Rose, 2011] L. M. Rose. *Structures and Processes for Managing Model-Metamodel Co-evolution*. PhD thesis, University of York, 2011.

[Rose et al., 2008] L. M. Rose, R. F. Paige, D. S. Kolovos, and F. A. Polack. The Epsilon Generation Language. In *Proc. of the 4th European conference on Model Driven Architecture: Foundations and Applications*, ECMDA-FA '08, pages 1–16. Springer-Verlag, 2008.

[Rose et al., 2009] L. M. Rose, D. S. Kolovos, R. F. Paige, and F. A. Polack. Enhanced automation for managing model and metamodel inconsistency. In *Proc. of the 26th IEEE/ACM International Conference on Automated Software Engineering*, volume 0 of *ASE '11*, pages 545–549. IEEE Computer Society, 2009.

[RTCA, 2005] RTCA. *DO-254: Design Assurance Guidance for Airborne Electronic Hardware*, 2005.

[RTCA and EUROCAE, 1992] RTCA and EUROCAE. *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*. Radio Technical Commission for Aeronautics (RTCA), 1992.

[Rummler et al., 2007] A. Rummler, B. Grammel, and C. Pohl. Improving traceability in model driven development of business applications. In *Proc. of the 3rd Traceability Workshop of European Conference on Model Driven Architecture: Foundations and Applications*, ECMDA-FA '07, 2007.

[Sabetzadeh and Easterbrook, 2005] M. Sabetzadeh and S. Easterbrook. Traceability in viewpoint merging: a model management perspective. In *Proc.*

the International Workshop on Traceability in Emerging Forms of Software Engineering, TEFSE '05, pages 44–49. IEEE Computer Society, 2005.

[Salay et al., 2008] R. Salay, J. Mylopoulos, and S. Easterbrook. Managing models through macromodeling. In *Proc. of the 23rd IEEE/ACM International Conference on Automated Software Engineering*, ASE '08, pages 447–450, 2008.

[Salay et al., 2009] R. Salay, J. Mylopoulos, and S. Easterbrook. Using macromodels to manage collections of related models. In *Proc. of the 21st International Conference on Advanced Information Systems Engineering*, CAiSE '09, pages 141–155. Springer-Verlag, 2009.

[Salazar-Zárate et al., 2003] G. Salazar-Zárate, P. Botella, and A. Dahanayake. Introducing non-functional requirements in UML. In L. Favre, editor, *UML and the Unified Process*, pages 116–128. IGI Global, 2003.

[Santiago et al., 2012] I. Santiago, A. Jiménez, J. M. Vara, V. D. Castro, V. A. Bollati, and E. Marcos. Model-driven engineering as a new landscape for traceability management: A systematic literature review. *Information Software and Technology*, 54(12):1340–1356, 2012.

[Schmid et al., 2006] K. Schmid, K. Krennrich, and M. Eisenbarth. Requirements management for product lines: extending professional tools. In *Proc. of the 10th International Software Product Line Conference*, SPLC '06, pages 10–122, 2006.

[Schmidt, 2006] D. C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, 2006.

[Schwaber and Beedle, 2001] K. Schwaber and M. Beedle. *Agile Software Development with Scrum*. Prentice Hall PTR, 1st edition, 2001.

[Schwarz et al., 2008] H. Schwarz, J. Ebert, V. Riediger, and A. Winter. Towards querying of traceability information in the context of software evolution. In *Proc. of 10th Workshop on Software Reengineering*, pages 144–148, 2008.

[Schwarz et al., 2010] H. Schwarz, J. Ebert, and A. Winter. Graph-based traceability: A comprehensive approach. *Software and Systems Modeling*, 9 (4):473–492, September 2010.

[Seibel et al., 2010] A. Seibel, S. Neumann, and H. Giese. Dynamic hierarchical mega models: comprehensive traceability and its efficient maintenance. *Software and Systems Modeling*, 9(4):493–528, 2010.

[Seibel et al., 2012] A. Seibel, R. Hebig, and H. Giese. Traceability in model-driven engineering: Efficient and scalable traceability maintenance. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 215–240. Springer London, 2012.

[Seidewitz, 2003] E. Seidewitz. What models mean. *IEEE Software*, 20(5):26–32, 2003.

[Settimi et al., 2004] R. Settimi, J. Cleland-Huang, O. B. Khadra, J. Mody, W. Lukasik, and C. DePalma. Supporting software evolution through dynamically retrieving traces to UML artifacts. In *Proc. of the 7th International Workshop on Principles of Software Evolution*, IWPSE '04, pages 49–54. IEEE Computer Society, 2004.

[Sharif and Maletic, 2007] B. Sharif and J. I. Maletic. Using fine-grained differencing to evolve traceability links. In *Proc. of the International Symposium on Grand Challenges in Traceability*, GCT '07, pages 76–81. ACM, 2007.

[Sherba et al., 2003] S. A. Sherba, K. M. Anderson, and M. Faisal. A framework for mapping traceability relationships. In *Proc. of the International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '03, pages 32–39. IEEE Computer Society, 2003.

[Sommerville, 2007] I. Sommerville. *Software engineering*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 8th edition, 2007.

[Song et al., 2011] S. Song, Y. Kim, S. Park, and S. Park. A non-functional requirements traceability management method based on architectural patterns. In R. Lee, editor, *Computers,Networks, Systems, and Industrial Engineering 2011*, volume 365 of *Studies in Computational Intelligence*, pages 25–35. Springer Berlin Heidelberg, 2011.

[Spanoudakis and Zisman, 2004] G. Spanoudakis and A. Zisman. Software Traceability: A Roadmap. In *Handbook of Software Engineering and Knowledge Engineering*, pages 395–428. World Scientific Publishing, 2004.

[Spanoudakis et al., 2003] G. Spanoudakis, A. S. d'Avila Garcez, and A. Zisman. Revising rules to capture requirements traceability relations: A machine learning approach. In *Proc. of the 5th International Conference on Software Engineering & Knowledge Engineering*, SEKE '03, pages 570–577, 2003.

[Spanoudakis et al., 2004] G. Spanoudakis, A. Zisman, E. Perez-Minana, and P. Krause. Rule-based generation of requirements traceability relations. *Journal of Systems and Software*, 72(2):105–127, 2004.

[Sparx Systems Pty Ltd., 2014] Sparx Systems Pty Ltd. Sparx Enterprise Architect^{TM}. `http://www.sparxsystems.com.au/`, 2014.

[Spence and Probasco, 2000] I. Spence and L. Probasco. Traceability strategies for managing requirements with use cases. Technical report, IBM, 2000.

[Starfield et al., 1990] M. Starfield, K. Smith, and A. Bleloch. *How to model it: Problem Solving for the Computer Age.* McGraw-Hill Inc., New York, 1990.

[Steinberg et al., 2009] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework.* Addison-Wesley, Boston, MA, 2nd edition, 2009.

[Tang et al., 2007] A. Tang, Y. Jin, and J. Han. A rationale-based architecture model for design traceability and reasoning. *Journal of Systems and Software*, 80(6):918–934, 2007.

[Tang et al., 2010] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. A. Babar. A comparative study of architecture knowledge management tools. *Journal of Systems and Software*, 83(3):352–370, 2010.

[Taromirad et al., 2013] M. Taromirad, N. Matragkas, and R. F. Paige. Towards a multi-domain model-driven traceability approach. In *Proc. of the 7th International Workshop on Multi-Paradigm Modelling*, MPM '13, 2013.

[Tekinerdogan et al., 2007a] B. Tekinerdogan, C. Hofmann, and M. Aksit. Modeling traceability of concerns for synchronizing architectural views. *Journal of Object Technology*, 6(7):7–25, 2007a.

[Tekinerdogan et al., 2007b] B. Tekinerdogan, C. Hofmann, M. Aksit, and J. Bakker. Metamodel for tracing concerns across the life cycle. In A. Moreira and J. Grundy, editors, *Early Aspects: Current Challenges and Future Directions*, volume 4765 of *Lecture Notes in Computer Science*, pages 175–194. Springer Berlin Heidelberg, 2007b.

[The Eclipse Foundation, 2013a] The Eclipse Foundation. Xtend. `http://www.eclipse.org/xtend/`, 2013a.

[The Eclipse Foundation, 2013b] The Eclipse Foundation. Xtext. `http://www.eclipse.org/Xtext/`, 2013b.

[The Open Group, 2011] The Open Group. TOGAF Version 9.1. `http://pubs.opengroup.org/architecture/togaf9-doc/arch/`, 2011.

[The UK Ministry of Defence, 2012] The UK Ministry of Defence. MOD Architecture Framework. `https://www.gov.uk/mod-architecture-framework`, 2012.

[Thorn, 2013] S. Thorn. Redefining traceability in enterprise architecture and implementing the concept with TOGAF 9.1 and/or

ArchiMate 2.0. `http://sergethorn.blogspot.co.uk/2013/05/redefining-traceability-in-enterprise.htmll`, 2013.

[Tisi et al., 2009] M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin. On the use of higher-order model transformations. In R. F. Paige, A. Hartman, and A. Rensink, editors, *Model Driven Architecture - Foundations and Applications*, volume 5562 of *Lecture Notes in Computer Science*, pages 18–33. Springer Berlin Heidelberg, 2009.

[U.S. Department of Defense (DoD), 1997] U.S. Department of Defense (DoD). Command, control, computers, communication, intelligence, surveillance, and reconnaissance (C4ISR) framework. `http://www.afcea.org/education/courses/archfwk2.pdf`, 1997.

[van den Berg et al., 2006] K. van den Berg, J. M. Conejero, and J. Hernández. Analysis of crosscutting across software development phases based on traceability. In *Proc. the International Workshop on Early Aspects at ICSE*, EA '06, pages 43–50. ACM, 2006.

[Vanhooff et al., 2007] B. Vanhooff, D. Ayed, S. Baelen, W. Joosen, and Y. Berbers. Uniti: A unified transformation infrastructure. In G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, editors, *Model Driven Engineering Languages and Systems*, volume 4735 of *Lecture Notes in Computer Science*, pages 31–45. Springer Berlin Heidelberg, 2007.

[Varro and Balogh, 2007] D. Varro and A. Balogh. The model transformation language of the VIATRA2 framework. *Science of Computer Programming*, 68(3):214 – 234, 2007. Special Issue on Model Transformation.

[Vliet, 2008] H. V. Vliet. Software architecture knowledge management. In *Proc. of the 19th Australian Conference on Software Engineering*, ASWEC '08, pages 24–31, March 2008.

[Von Knethen and Paech, 2002] A. Von Knethen and B. Paech. A survey on tracing approaches in practice and research. Technical report, Fraunhofer IESE, 2002.

[Walderhaug et al., 2006] S. Walderhaug, U. Johansen, E. Stav, and J. Aagedal. Towards a generic solution for traceability in MDD. In *Proc. of the ECMDA Traceability Workshop*, ECMDA-TW '06, 2006.

[Watkins and Neal, 1994] R. Watkins and M. Neal. Why and how of requirements tracing. *IEEE Software*, 11(4):104 –106, 1994.

[Wenzel et al., 2007] S. Wenzel, H. Hutter, and U. Kelter. Tracing model elements. In *Proc. of the IEEE International Conference on Software Maintenance*, ICSM '07, pages 104–113. IEEE Computer Society, 2007.

[Wieringa, 1995] R. J. Wieringa. An introduction to requirements traceability. Technical Report IR-389, Faculty of Mathematics and Computer Science, Vrije Universiteit, 1995.

[Winkler, 2008] S. Winkler. On usability in requirements trace visualizations. In *Proc. of the 3rd International Workshop on Requirements Engineering Visualization*, REV '08, pages 56–60. IEEE Computer Society, 2008.

[Winkler and Pilgrim, 2010] S. Winkler and J. Pilgrim. A survey of traceability in requirements engineering and model-driven development. *Software and Systems Modeling (SoSyM)*, 9:529–565, 2010.

[World Wide Web Consortium (W3C), 1999] World Wide Web Consortium (W3C). XSL Transformations (XSLT) Version 1.0. `http://www.w3.org/TR/xslt`, 1999.

[Yie et al., 2009] A. Yie, R. Casallas, D. Deridder, and D. Wagelaar. A practical approach to multi-modeling views composition. *ECEASST*, 21, 2009.

[Ying et al., 2004] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering*, 30(9):574–586, 2004.

[Zhang et al., 2006] Y. Zhang, R. Witte, J. Rilling, and V. Haarslev. An ontology-based approach for traceability recovery. In *Proc. of the 3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering*, ATEM '06, pages 36–43, 2006.

[Zimmermann et al., 2005] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, June 2005.

[Zisman, 2012] A. Zisman. Using rules for traceability creation. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 147–170. Springer, 2012.

[Zou et al., 2006] X. Zou, R. Settimi, and J. Cleland-Huang. Phrasing in dynamic requirements trace retrieval. In *Proc. of the 30th Annual International Computer Software and Applications Conference*, volume 1 of *COMPSAC '06*, pages 265–272, 2006.

[Zou et al., 2010] X. Zou, R. Settimi, and J. Cleland-Huang. Improving automated requirements trace retrieval: A study of term-based enhancement methods. *Empirical Software Engineering*, 15(2):119–146, 2010.