Traceability Modeling for the Engineering of Heterogeneous Systems

By

Nasser Mustafa

A Thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

PhD

in

Electrical and Computer Engineering

Carleton University

Ottawa, Ontario

Abstract

Capturing traceability information among artifacts helps ensure product quality and assists tracking functional and non-functional requirements, and performing system validation and impact analysis. Although literature provides many techniques for modeling traceability, existing solutions are either tailored to specific domains or not complete enough (e.g., lack support to specify traceability link semantics). This research examines the current traceability solutions and identifies the drawbacks that hinder capturing some traceability information of heterogeneous artifacts. In this context, heterogeneous artifacts refer to artifacts that come from widely different modeling notations (e.g., UML, Simulink, natural language text, source code). In this thesis, our contribution comprises a traceability framework that can accommodate the traceability of system engineering artifacts which come from different domains of expertise. The framework includes the followings: First, a set of requirements for a traceability model that are necessary to build a generic traceability model. Second, a generic traceability model that is not domain specific and which, therefore, provides a solution for modeling traceability links among heterogeneous models, that is, models for which traceability links need to be established between artifacts in widely different modeling languages (e.g., UML, block diagrams, informal documents). We argue that the proposed requirements are sufficient to build a traceability model oblivious of the heterogeneity of the models whose artifacts need to be traced. We also argue that our traceability model is extensible in the sense that it can adapt to new modeling languages, new ways of characterizing traceability information for instance, without requiring changes to the model itself; Third, a trace links taxonomy that encompasses semantically well-defined trace links that can be utilized along with the traceability model. The design of our framework is validated through a set of validation methods. Also, it is supported by our findings from a survey and a systematic literature review.

Acknowledgement

I would like to thank my supervisor Dr. Yvan Labiche the head of the he Software Quality Engineering Laboratory (SQUALL) at Carleton University for his unlimited support and advice during my PhD study. Also, I would like to extend my thanks to Dr. Daniel Amyot from the University of Ottawa for his valuable comments and the revision of my thesis.

I would like to thank NSERC, CRIAQ, CAE, CMC Electronics, and Mannarino Systems & Software for their financial support.

Last but not least, many thanks for my lovely family; my wife Hanan, my daughter Nathalie, and my son Mohammed for their support and encouragement.

Dedication

To the souls of my father and mother who gave me their courage and support.

Table of Contents

Abstract ..... **i**

Table of Contents ..... **iv**

List of Figures ..... **viii**

1 Introduction **1**

1.1 Thesis Problem: Tracing Heterogeneous Artifacts 2
1.2 Thesis Objective 5
1.3 Thesis Contributions 6
1.4 Research Methodology 7
1.5 Thesis Organization 9

2 Traceability Concepts and Practices **11**

2.1 Background 11
2.2 Traceability concepts 12
2.3 Summary 15

3 Systematic Literature Review **17**

3.1 Planning the Review 17
3.2 Inclusion and Exclusion 19
3.3 Conducting the review 21
3.4 Refining the Search Article 22
3.5 The Review Findings 23
3.5.1 Traceability Concepts and definitions 23
3.5.2 Modeling Traceability 24
3.5.3 Traceability Reviews and Surveys 27
3.6 Summary 28

4 Trace Links Classification **30**

4.1 Requirements Engineering Classifications 30
4.2 Systems Engineering Classifications 35
4.3 Summary 36

5 Traceability Modeling in the Literature **38**

5.1 Domain Specific Traceability Modeling 38
5.2 Generic Traceability Modeling 41
5.3 Analysis of Existing Research on Modeling Traceability 49
5.4 Summary 51

6 Traceability Survey **52**

6.1 Survey Planning 52
6.2 Survey Objectives 53
6.3 Targeted Audience and Population 53
6.4 Survey Questionnaire Design 54
6.4.1 Demographic Questions 55
6.4.2 Traceability Questions 56
6.5 Publishing the Survey 57
6.6 Survey Analysis 57
6.6.1 Analysis of Demographic Feedback 58

6.6.2 Analysis of Artifacts and Trace Links Feedback 59
6.6.3 Analysis of Traceability Tools Feedback 60
6.7 Conclusion 63

7 Generic Traceability Model Design 65

7.1 Context 65
7.2 Requirements for a Generic Traceability Model 66
7.3 Traceability Model Description 68
7.4 Validation of Traceability Model by Construction 74
7.5 Tool Support and Integration 75

8 A New Encompassing Traceability Link Taxonomy 78

8.1 The Need for a Trace Link Taxonomy 78
8.2 Simple, Motivating Example 80
8.3 The Drawbacks of Existing Classifications 83
8.4 The Taxonomy Requirements 84
8.5 Employing the RDF in Building the Taxonomy 85
8.6 Taxonomy Design and implementation 87
8.6.1 Taxonomy Design 87
8.6.2 Design Decisions 89
8.6.3 Taxonomy Implementation 90
8.7 Taxonomy Validation 92
8.7.1 Taxonomy Compliance Criteria 92
8.8 Summary 97

9 Model Validation 98

9.1 Validation Criteria 98
9.2 Validation Cases for Existing Traceability Models 99
9.3 Analysis of the Validation Results 101

10 Validation by Example—An Avionics Case Study 104

10.1 Objectives 104
10.2 Data Collection 104
10.3 Case Study Description 105
10.4 Problem Identification 106
10.5 Slat/Flap Requirements Specification 108
10.6 Artifacts Identification 110
10.7 Identifying Trace Links 113
10.8 Traceability Model Validation 115
10.9 Conclusion 119

11 Threats to Validity 120

11.1 Strategies for Mitigating Validity Threats 120
11.2 Mitigating Threats to the Validity of the Systematic Literature Review 123
11.3 Mitigating the Threats to the Validity of the Survey 124
11.4 Mitigating the Threats to the Traceability Model Requirements 126
11.5 Mitigating the Threats to the Traceability Model Design 127
11.5.1 Internal threats to validity 127
11.5.2 External threats to validity 129
11.6 Mitigating Threats to the Validity of the Trace Links Taxonomy 129
11.7 Mitigating Threats to the Validity of the Case Study 129
11.8 Remaining Threats 130
11.8.1 Remaining Threats to Survey 130
11.8.2 Remaining Threats to our Traceability Model 131

11.8.3 Remaining Threats to our Taxonomy 133
11.8.4 Remaining Threats to the Literature Review 133
11.8.5 Remaining Threats of the Case Study 134
11.9 Summary 134
12 Conclusion and Future Work 136
13 References 169

List of Tables

Table 1: Extended definitions to horizontal and vertical trace links in Systems Engineering 14
Table 2: Guidelines for a systematic literature review 18
Table 3: Number and percentage of traceability articles in major digital libraries 21
Table 4: Number of retrieved traceability articles 21
Table 5: Number of articles distributed by each traceability topic 22
Table 6: Classifications for RE trace links 31
Table 7: RE Process-related and Product-related classification 33
Table 8: Model Driven Engineering trace links classifications 34
Table 9: Trace links types in Systems Engineering 36
Table 10: Results of analyzing existing traceability models 50
Table 11: Questions about artifacts and trace links 56
Table 12: Questions about traceability tools 57
Table 13: Traceability Model design rationale 75
Table 14: Trace links classifications in Requirements Engineering, Model Driven Engineering, and Systems Engineering 82
Table 15: Trace links between instances of the i* an the UML-Class models 96
Table 16: Validation of existing traceability models 102
Table 17: Aircraft lever positions 106
Table 18. Identifying requirements and artifacts from the Slat/Flap case Study 113
Table 19: Validation cases for validating the traceability model 114
Table 20: Identified threats to validity 122

List of Figures

Figure 1: Cases of semantic coupling traceability relations, source [104] 14
Figure 2: Transformation chain traceability model, source [81] 39
Figure 3: Traceability Example: *i** (excerpt) model (left), UML (excerpt) class diagram (right), traceability links (grayed dashed lines), source [51] 40
Figure 4: SPL traceability metamodel, source [89] 43
Figure 5: Traceability Metamodel for Data Warehouses, source [87] 44
Figure 6: TIM traceability model for critical systems, source [51] 45
Figure 7: TML Traceability model, source [48] 47
Figure 8: Traceability model for component -package example, source [99] 48
Figure 9: Traceability Questionnaire 54
Figure 10: Feedback from demographic questions 58
Figure 11: Feedback of trace links and artifacts questions 59
Figure 12: Features of traceability tools 61
Figure 13: Generic traceability model 69
Figure 14: The relation between traceability model and MOF architectures 73
Figure 15: Proposed traceability framework, source [89] 76
Figure 16: RDF Code for describing trace links 84
Figure 17: Trace links examples: leaf and type 89
Figure 18: The dependency relation in the taxonomy 90
Figure 19: The trace links taxonomy (excerpt) 91
Figure 20: Traceability Example: *i** (excerpt) model (left), UML (excerpt) class diagram (right), traceability links (grayed dashed lines [51] 95
Figure 21: Wing chord and relative air flow, source [154] 108
Figure 22: State Machine Diagram 110
Figure 23: Activity diagram of SFCC and ADIRU 111
Figure 24: Component diagram for slat flap units 112
Figure 25: Signal Flow diagram 112
Figure 26: Object traceability diagram for applying characterization to a trace link 115
Figure 27: Object traceability diagram for tracing a source artifact to a target artifact 115
Figure 28: Object traceability model for tracing a source artifact to more than one target artifacts 116
Figure 29: Object traceability diagram for artifacts characterization 117
Figure 30: Object traceability diagram for version control information 117
Figure 31: Object traceability diagram for bidirectional traceability between two artifacts 118
Figure 32: Traceability object diagram for applying a constraint on trace link 118
Figure 33: Object traceability diagram for tracing artifacts from different domains 119

1 Introduction

Traceability in its simplest form is the ability to describe and follow the life of software artifacts [1]. The need for traceability started originally in Requirements Engineering and later permeated Model Driven Engineering and Systems Engineering. Requirements Engineering refers to the application development phase where requirements are gathered and processed to produce requirement specifications [2]. The term Model Driven Engineering refers to a systematic software development approach in which abstract models of software systems are created and transformed into concrete applications [3]. Systems Engineering is an interdisciplinary approach and a means to enable the realization of systems composed of software and hardware components [4].

Traceability gained more interest at the NATO Software Engineering conference in 1968 [5]. One of the conference papers emphasized the need for an effective methodology to ensure that any developed system adheres to its requirements. As a result of applying an effective methodology, some projects were praised because they included traces from requirements to design. Later, in the 1980s and 1990s, traceability gained more attention in academia and industry.

According to a comprehensive study conducted by Winkler and Pilgrim [1], traceability can be used for many purposes. First, it can be used for development process assessment: the logs of events that occur during the development process are contained within traces, these traces can be used to revise and evaluate the development process. Second, traceability information can be used to discuss different design alternatives; trace links allow tracking functional and non-functional requirements and evaluate alternatives with respect to those requirements. Third, traceability can help perform system validation; the trace links can be analyzed to discover whether all requirements have been fulfilled [6], [7]; Traceability information can be used by

project managers to track requirements in order to find out which requirements are released and which requirements are not yet implemented [8]. Fourth, traceability information can be utilized in project management to perform impact analysis [7]. The trace links can identify dependencies among entities during all phases of system development: for example, the traceability information can be used to assess the effect of changing a requirement on other requirements. Fifth, traceability provides accountability: it can be used to relate requirements or design components to the stakeholders who originate them [9]. For instance, traceability can provide information about who originated a design element, when it was originated, and who validated it.

1.1 Thesis Problem: Tracing Heterogeneous Artifacts

In Systems Engineering, many systems encompass widely different domains of expertise. As an example, consider the engineering of a full flight simulator, which artificially re-creates an aircraft flight and the environment in which it flies and is used for pilot training. A full flight simulator typically includes software (e.g., simulating specific hardware components, simulating missions), visuals and audio (e.g., audio rendering of sound inside the flight deck, video rendering of typical airports), mechanical systems (e.g., to provide accurate force feedback to the pilot, to provide motion for the entire flight deck simulator), communication systems (e.g., air traffic). In this example, several heterogeneous models need to be related to one another during system engineering. These models are heterogeneous primarily because they tend to be specific to the many disciplines that are involved in the design of the system. For instance, one would have a simulation model for mission simulation that can be used to create specific scenarios for training purposes; one model would be a Simulink hydraulic actuation system; one model would be a simulation model of a hardware component; one (graphical) model would be used to represent takeoff and landing characteristics (e.g., visuals, air traffic data) of typical airports; one model would record the requirements for the whole

system and the requirements for its many (software and hardware) parts; one model would record data about verification and validation activities; one model would record faults and failures. These models are also heterogeneous because they are not typically specified using a common notation and semantics, i.e., a common metamodel. Indeed, some software elements can be specified with the Unified Modeling Language (UML) [10], some system level characteristics can be modeled with the System Modeling Language (SysML) [11], some hardware elements can be modeled with Simulink models, and some information can even be provided in a plain language (e.g., requirements). Note that in this context, we use the term model in the widest sense of the word, and the notion of model includes (but is not restricted to) diagrams, plain language texts, equations, and source code.

Therefore, industries such as the aviation are facing challenges in tracing multiple artifacts that constitute a complex solution such as a full flight simulator. These challenges arise from the heterogeneity of system artifacts, and the complexity of the system. Additionally, not all modeling techniques used in Systems Engineering to describe different aspects of a system are typically instances of the OMG's Meta-Object Facility (MOF), a standard for representing modeling languages [12]. Finally, traceability management is a fluid activity in the sense that not all traceability requirements are necessarily known upfront when traceability links are first recorded. For instance, one may not know precisely from the outset the granularity of the artifacts that need to be traced to one another; one may discover down the road that artifacts from new models need to be traced. Therefore, traceability of heterogeneous artifacts implies different meanings that can be summarized as follows:

- The traceable artifacts are modeled by different modeling tools and can have different formats and semantics. For instance, in MDD during model transformation, the generated artifacts and trace links can have different types than the ones in Requirements Engineering. In this context, the traceable artifacts cannot be transformed

into other formats. For instance, a class diagram modeled in Papyrus can be transformed into a Java or XML code; however, if we model the same diagram with the Microsoft Visio tool, Visio will not have the capability of transforming a class diagram into a Java code. Therefore, artifact heterogeneity can be tool dependent.

- The traceable artifacts are not using common notation. As a result, artifacts of non-transformable format are produced.
- The traceable artifacts can exist at different phases within the same model or across different models.
- Although traceable artifacts can exist in one model, there are situations in which trace artifacts are traced at different levels of granularity based on a user's need. This implies the need to accommodate various, and user-defined, artifacts characterization.
- As a result of artifacts characterization, there should be various types of trace links to link these artifacts which implies the need for new trace links that can accommodate the traceability of these artifacts.
- A trace link can have different interpretations by different stockholders, which implies the need for a precise semantic for every type.

In conclusion, the capability of a modeling tool to transform or produce different formats of a given model governs the definition of artifacts heterogeneity. However, we argue that even if a certain modeling tool transform a model into another format, say XML, then we need traceability at the XML level between tagged data. It is not realistic to trace artifacts at the XML level since it is created to be machine readable and not human readable. Also, the value of artifacts traceability at the XML level is not clear. Therefore, traceability of heterogeneous artifacts at the model level is required. We also note that the type of systems that is of interest to our industry partner is often referred to as cyber-physical systems. The need for multi-domains, multi-disciplines, multi-levels (of abstraction), in other words heterogeneous models,

has been documented and showed to be an important issue to circumvent for the design and construction of cyber-physical systems [13].

We summarize the thesis problem by listing several questions, which are divided into two groups: (a) a group that investigates the previous research on traceability (questions 1-4), and (b) questions that build upon our investigation and identifies our contributions (questions 5-7).

Question 1: What are the available solutions (i.e., models and tools), in the literature, to model and capture traceability among artifacts?

Question 2: How generic (i.e., not domain specific) are these solutions, and to what extent can they handle heterogeneous artifacts?

Question 3: What are the drawbacks of the existing solutions, if any, that hinder capturing traceability information of heterogeneous artifacts?

Question 4: To what extent can these solutions help a user specify traceability information (e.g., the type of a trace link between any two given artifacts, the type of artifacts, and the constraints applied to trace links and artifacts) with precision?

Question 5: Can we improve upon the current solutions in the specific, heterogeneous context discussed previously?

Question 6: Can our solution be integrated with other development tools?

Question 7: What is the scalability of our solution? Can it relate artifacts of complex systems?

1.2 Thesis Objective

Our thesis objective is to build a traceability framework that encompasses a traceability model and trace links taxonomy which can accommodate the capturing of traceability information of heterogeneous artifacts. The traceability model must be oblivious of the heterogeneity of the model's elements to be traced. It should not rely on the fact that artifacts are instances of a particular modeling language. In addition, it should accommodate other situations where new artifacts, possibly in new models, need to be traced. In other words, the devised traceability

model should not change when new artifacts, coming from newly created modeling notations need to be traced. Similarly, it should accommodate the situation where different, possibly new, ways of characterizing trace data are required since, for example, characterizing trace data is likely to be domain, organization, or even project dependent. One can view these two last constraints as having to identify a traceability model that can be extensible (to accommodate new models, new artifacts, different, possibly new ways of characterizing them) without having to change the model itself or its instantiation (i.e., existing traceability links). On the other hand, the trace links taxonomy should consider the possible trace link types that exist between heterogeneous artifacts whether these artifacts are coming from Requirements Engineering, Model-Driven Engineering, or Systems Engineering. The taxonomy shall combine all existing trace links types in a way to eliminate duplications of the current classifications. Moreover, it shall allow the users of the traceability model to identify the type of a trace link among given artifacts based on the usage, semantic, or functionality of the trace link.

1.3 Thesis Contributions

Our search for a solution in the literature to the problem specified earlier was not successful. The main reason is that each existing solution is tailored to a specific domain: e.g., some traceability modeling techniques can only trace artifacts from MOF-based models, some traceability modeling techniques can only trace during model transformation. As we conducted our search, we also noticed, putting aside the abovementioned issues, that each traceability modeling technique has its own advantages and drawbacks, that is, it is difficult to find a solution that offers the advantages of all the currently existing solutions at once. This is especially true about the different, complementary ways we have found to characterize traceability information. Regarding traceability links, our literature search shows that there is no unified way of identifying a type of trace links among artifacts in models.

Based on these findings, our contribution in this thesis is divided into the following six facets:

1. Conducting a systematic literature review in order to examine the current traceability models and identify their advantages and drawbacks, specifically reasons that prevent capturing traceability information of heterogeneous artifacts.
2. Conducting a traceability survey in order to get feedback from professionals about traceability practices in industry.
3. Proposing traceability requirements that are necessary to build a generic traceability model. We envision these requirements based on previous works on traceability modeling and their gaps, the feedback we get from the survey, our need for a traceability model that can capture traceability information among heterogeneous artifacts, and negotiate these requirements with our industrial partner software team at CAE (http://www.cae.com/) who is one of our stakeholders in this work.
4. Proposing a generic traceability model that can capture traceability information among heterogeneous artifacts while satisfying the requirements we devised.
5. Proposing a trace links taxonomy with well-defined semantics which can be employed by the traceability model.
6. Validating our traceability model and the taxonomy through different ways.

1.4 Research Methodology

We adopt certain steps that are based on the design and creation methodology in conducting our research [14]. The systematic literature review in chapter 3 is the base for answering the research questions about the importance of capturing traceability among heterogeneous artifacts particularly, critical systems artifacts. Although the literature review shows previous solutions with respect to traceability modeling, these solutions fall short to satisfy our research problem since most of them are domain-specific.

Based on the gaps that we found about the existing traceability models, we proposed a generic traceability model design that can complement the existing traceability solutions. Therefore, we adopt the following steps in our research methodology:

- Employ the systematic literature review in order to identify previous works on traceability modeling in Software Engineering and Systems Engineering. We believe this step is important since it ensures the coverage of up-to-date research on traceability. The details of this step are explained further in chapter 3.
- Examine the articles that focus on modeling traceability and summarize the features of the traceability models that they discuss. This step also involves debating each model in terms of the platform it supports, its extensibility, the type of artifacts and trace links it can capture, and its drawbacks in satisfying our research questions. The step is necessary in order to identify the capability of each model in capturing traceability information.
- Aside from traceability models, we examine articles that classify trace links in Software Engineering and Systems Engineering and summarize their classifications. The gathered information is utilized in building a trace links taxonomy that can be used during the instantiation of our traceability model.
- Utilize the feedback from the survey to provide support for our finding from the literature review.
- Test the validity of existing traceability models using our validation criteria. The importance of this step is to identify the benefits and drawbacks of each existing traceability model.
- Incorporate our findings from the literature review, the survey, and the requirements of our industrial partner to propose requirements for a generic traceability model.

- Design a generic traceability model by incorporating some features from previous traceability models and adding new features that satisfy the proposed requirements.
- Design a trace links taxonomy that provides semantics for trace links through a set of properties. The taxonomy is important during traceability model instantiation.
- Ensure the validity of our model by using two methods: a) validation by construction, b) validation using an industrial case study from the aviation domain. In the case of validation by construction, we justify the need for creating each class, association and attribute in our traceability model. This validation method is unique in our model since we cannot have a case study that covers all types of artifacts, trace links, and associations. Therefore, we found validation by construction is effective in this situation. The survey provides us with feedback about traceability practices in industry which provides another way for validation. Finally, the industrial case study represents a real-world problem that we can employ to find out whether our generic traceability model can satisfy all the requirements and hence solve the research problem.

1.5 Thesis Organization

The remainder of the thesis is structured as follows. Chapter 2 explains some traceability concepts and definitions that provide an initial understanding of our research. Chapter 3 provides a systematic literature review about traceability in Requirements Engineering, Model Driven Engineering, and Systems Engineering. It specifies the necessary steps for planning and conducting the review, constructing the search string, and presenting the review findings. Chapter 4 discusses the classifications of trace links in Requirements Engineering, Model Driven Engineering, and Systems Engineering based on our findings from the literature review.

Chapter 5 complements chapter 4 by investigating existing traceability (Meta) models. It provides a description of these (Meta) models, their usage, and their drawbacks. Chapter 6 discusses our method for conducting a survey on traceability and the result we obtained. Chapter 7 proposes a set of generic traceability model requirements, the design of a generic traceability model. We also justify and validate our design in this chapter. Chapter 8 extends chapter 4 and chapter 7 by constructing a trace links taxonomy that provides semantics for trace links. The taxonomy can be used to reference any trace link type during the instantiation of the traceability model. Chapter 9 discusses our design for a generic traceability model based on our findings from the systematic literature review and the survey. Chapter 10 provides a description for an aviation case study and our validation for the traceability model. Chapter 11 discusses the threat to the validity of our research. The research involves several software engineering methods and several elements of our solution that we need to validate and we need to show how we can mitigate their validity threats. Specifically, we need to mitigate threats to validity for our literature review, our traceability survey, our traceability requirements, our traceability model, our trace links taxonomy, and our case study. Finally, chapter 12 summarizes our work and draws conclusions and suggestions for future work.

2 Traceability Concepts and Practices

This chapter introduces the importance of traceability in Systems Engineering specifically in critical systems such as in the aviation and automobile industries. Moreover, it discusses the definitions and classifications of traceability in Requirements Engineering, Systems Engineering, and Model Driven Engineering domains. We conclude the chapter by showing the main challenges that hinder the application of traceability in different domains. Finally, we show how our solution addresses some of these challenges.

2.1 Background

Traceability has originated in Requirements Engineering and permeated Model Driven Engineering and Systems Engineering. Although we are concerned with traceability in general, we focus our research on traceability in Systems Engineering since the issue of tracing elements among the many artifacts that are created when engineering systems such as critical control systems for trains, automobiles, and airplanes is acute. In order to ensure proper functionality of these systems, rigorous measures must be applied to the software that control them. Applying traceability is one of these measures that must be applied in order to certify and qualify the systems software. Many agencies and standards such as the U.S Federal Aviation Administration (FAA), the European Space Agency (ESA), the Nuclear and Rail industries, the Food and Drug Administration (FDA), and the Capability Maturity Model Integration (CMMI) [15] require traceability practices as a quality assurance parameter to certify or qualify these systems [3], [4], [16]. For instance, the FAA requires DO-178C as the accepted means of certifying all new aviation software. This standard specifies that at each and every stage of development software developers must be able to demonstrate traceability of designs against requirements [16].

2.2 Traceability concepts

Traceability is defined by the IEEE [17] 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 definition applies to traceability in Requirements Engineering, Model Driven Engineering, and Systems Engineering as well.

The IEEE definition is extended to include other types and subtypes of relationships. Cleland-Huang and colleagues [18] describe trace link semantics and types. A trace link semantics refers to the purpose or meaning of the relationship between associated artifacts. A trace link type refers to the characterization of all trace links that have a similar structure (syntax) and/or purpose (semantics). The description of a trace link type encapsulates the definition of a trace link semantics since it is explained based on the link's semantic role, and may include other properties such as the rationale for creating a trace link. For instance, all trace links that relate two artifacts where one artifact is derived from another have the trace link type "*derived from*". *Derived* represents the meaning of the relation between such artifacts. Therefore, we might have similar or extended types of trace links among the Requirements Engineering, Model Driven Engineering, and Systems Engineering domains. Readers should note that we use a *trace link* and *relation* interchangeably, however, there is a difference between both terms since the latter refers to all trace links created between two sets of trace artifact types [18].

Gotel and colleagues [19] define requirement traceability as the ability to describe and follow the life of a requirement, in both forward and backward direction, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases. They extended this definition to define other types of traceability such as pre-requirement specification (pre-RS), which refers to "the aspects of a requirement life prior to its inclusion in the requirement specification", and post-requirement specification (post-RS), which refers

to “the aspects of a requirement’s life that result from its inclusion in the requirement specification”. For example, tracing a requirement from a baseline (reference point that specifies a change in requirement or design during product development life cycle) back to a requirement specification is considered (pre-RS) traceability.

Also, there are the notions of vertical and horizontal traceability [9], [20]–[22]. Horizontal traceability refers to tracing artifacts created in the same system lifecycle phase, or at the same level of abstraction. For instance, tracing two requirements based on the ‘derived from’ relationship is horizontal. Vertical traceability refers to tracing artifacts created in different phases or at different levels of abstraction, such as tracing a requirement in the requirement specification phase to a test case in the testing phase.

In Model Driven Engineering, the notion of traceability is typically restricted to model transformations where abstract models for requirements are transformed into more concrete models or Platform-Specific models. Aizenbud-Reshef [23] extended Gotel definition [19] of Requirements Engineering traceability to Model Driven Engineering traceability as “any relationship that exists between artifacts involved in the software engineering life cycle.” This definition encompasses the explicit links that are generated during model transformation or the implicit links that can be manually specified and computed based on history or existing information such as the information obtained from code dependency analysis [24].

Pinheiro [25] enhanced the definition of requirement traceability by including the trace of social aspects (e.g., people, policies, decisions) to technical aspects (e.g., specifications, diagrams, and code): requirement traceability is “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.”

Ramesh and Edwards [9] indicate that forward traceability implies referencing each requirement to a design element, and backward traceability means tracing back a design

Table 1: Extended definitions to horizontal and vertical trace links in Systems Engineering

Vertical/ Horizontal	Micro	Macro
Intra	Within system description and within system levels of decomposition	Within system description and across system levels of decomposition
Inter	Across system description and within system levels of decomposition.	Across system description and across system levels of decomposition.

element to a requirement, which implies bidirectional traceability. Bidirectional traceability is important to ensure that all design elements functionalities can be traced to a source requirement, and all resource requirements are completely addressed.

Costa and colleagues [26] identified two types of dependency relations between artifacts: *conceptual level* and *semantic coupling. Conceptual traceability* refers to the relation between artifacts based on the level of abstraction and relation between models. Traceability based on level produces the *vertical* and *horizontal* traceability, while traceability based on relations between models produces the *Intra traceability* (i.e., within the same model) and *Inter traceability* (i.e., between different models), see Table 1.

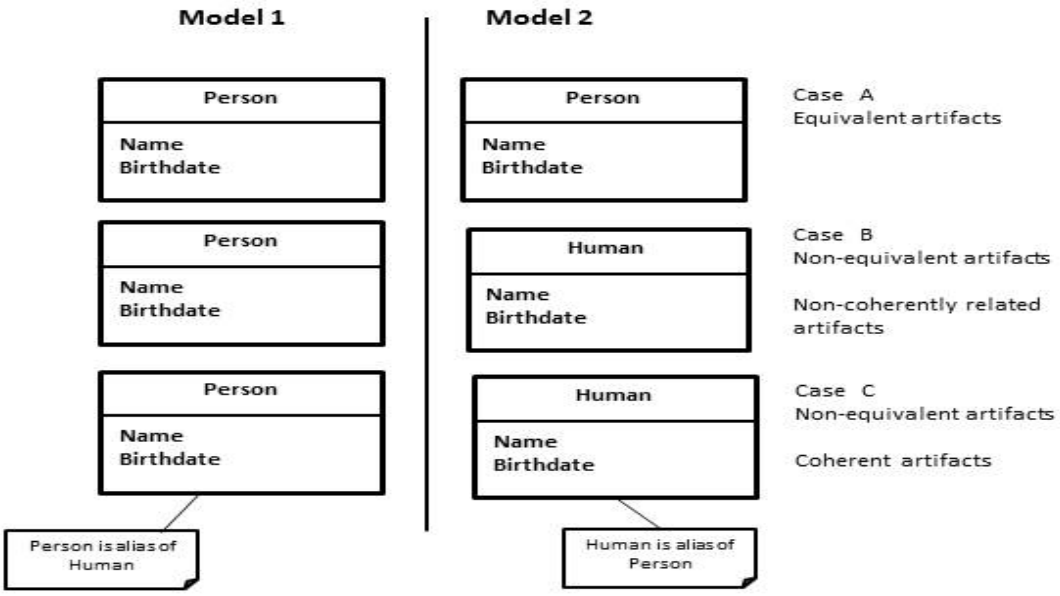

Figure 1: Cases of semantic coupling traceability relations, source [101]

On the other hand, *semantic coupling* traceability refers to the relation between artifacts based on their level of coupling. It involves the *equivalence relation* which is a relation between artifacts having the same representation. For instance, two classes with the same name having the same attributes even if they exist at different levels of abstraction or different models. As depicted by Figure 1, the authors identified three types of equivalence traceability relations: (a) equivalent traceability which relates artifacts that have the same name and same representation; (b) non-equivalent and non-coherent artifacts traceability which relates artifacts that have different names and the same representation; and (c) non-equivalent and coherent artifacts traceability which relates artifacts that have different names but their annotation is the same.

2.3 Summary

This chapter elucidates the importance of traceability in Systems Engineering and other domains. In addition, it provides the definitions of traceability in Requirements Engineering, Model Driven Engineering, and Systems Engineering.

Despite an agreement among practitioners (e.g., project managers, business analysts) and software process improvement models (e.g., CMMI) about the significance of traceability, there is a universal lack in solutions to traceability modeling and management amongst government and industrial projects [5]. A group of researchers from the traceability community collaborated with traceability practitioners and addressed the traceability challenges in a paper that draws a roadmap for the future of traceability [5]. The main challenges include configuration, trust, scalability, portability, and valued and ubiquity, see Gotel and colleagues for more insight [5]. These challenges open the door wide for researchers to tackle them and explore individual solutions for each challenge.

Our research contribution exists within the context of these challenges. In terms of trust we are proposing a traceability model that can capture systems requirements at different levels of granularity and trace these requirements to the related design, code, and test cases. Moreover,

our solution can accommodate the traceability of heterogeneous and homogeneous artifacts and among different domains. With respect to the configuration and portability, the proposed model shall not be dependent on any platform. It shall be oblivious of any platform. In addition, the model shall be easily integrated with other software applications that facilitate traceability. Regarding extensibility, the model shall be able to accommodate future traceability needs without changing the design of the traceability model.

In conclusion, our main contribution focusses on modeling a generic traceability solution that accommodate heterogeneous and, of course, homogeneous artifacts. We are not considering a specific flow of use of traceability information. In other words, we are not concerned with the traceability management process such as retrieving, monitoring, automatic retrieval of traces or exploitation of traceability information.

3 Systematic Literature Review

The need for research on traceability has increased as a result of mandating traceability in many software projects. The review covers the publications on traceability between the years 2000 to 2016. This range of publications should help discover most of the recent work on modeling traceability. In addition, there are several reviews and surveys published within these years that cover the traceability research prior to the year 2000: e.g., [1], [27]–[32]. The methodology that we used in this review is based on well-known guidelines in the field of Empirical Software Engineering research that were proposed by Kitchenham and colleagues [33], [34]. It covers three main parts (see Table 2): planning the review which is described in section 3.1, discussing inclusion and exclusion criteria in section 3.2, conducting the review which is covered in section 3.3, refining the search to fit our research questions as discussed in section 3.4, and reporting on the review which is done in section 3.5. Section 3.6 summarizes the chapter.

3.1 Planning the Review

The objective of conducting this review is to understand recent research trends on traceability. Although the review covers the work on traceability in general, our concern is to study the traceability among artifacts coming from different domains of expertise (i.e., heterogeneous artifacts). This concern becomes vital due to many reasons: First, as the complexity of systems under development grows, the complexity of its models grows. As a result, managing the requirements becomes a burden on engineers who are involved in the system development process. Any change to a system's requirement in one phase or model propagates to other models, which become unmanageable without applying traceability. Second, the lack of traceability in some systems such as safety critical systems can lead to catastrophic results as a result of not covering all test cases that ensure systems safety. For instance, the bodies that regulate aviation, train transportation, and power plant mandate traceability in their software

Table 2: Guidelines for a systematic literature review

Planning the review		Conducting the review	Reporting the review
▪ Identification of the need for a review ▪ Specifying the research question(s) ▪ Developing a review protocol ▪ Evaluating the review	▪	▪ Identification of research ▪ Selection of primary studies ▪ Study quality assessment ▪ Data extraction and monitoring ▪ Data synthesis	▪ Specifying dissemination mechanisms ▪ Formatting the main report ▪ Evaluating the report

systems and require zero tolerance to unexpected behaviors. Third, this review allows us to understand existing research in modeling traceability and pinpoints their shortcomings, and allows us to propose a solution for our domain problem which is modeling traceability among heterogeneous artifacts. The method used in our search focuses on answering the following research questions that we stated already in section 1.1:

Question 1: What are the available solutions (i.e., models and tools), in the literature, to model and capture traceability among artifacts?

Question 2: How generic (i.e., not domain specific) are these solutions, and to what extent can they handle heterogeneous artifacts?

Question 3 What are the drawbacks of the existing solutions, if any, that hinder capturing traceability information of heterogeneous artifacts?

Question 4: To what extent can these solutions help a user specify traceability information (e.g., the type of a trace link between any two given artifacts, the type of artifacts, and the constraints applied to trace links and artifacts) with precision?

Question 5: Can we improve upon the current solutions in the specific, heterogeneous context discussed previously?

Question 6: Can our solution be integrated with other development tools?

Question 7: What is the scalability of our solution? Can it relate artifacts of complex systems?

These questions are devised based on our need for modeling traceability in heterogeneous systems in order to achieve system verification and validation. Based on the research questions, we first identified key terms: traceability, modeling, heterogeneous. Since we are interested in different fields, we also added Model Driven Engineering, Requirements Engineering, and Systems Engineering. We, therefore, specified the following search string in order to extract the related publications:

"Traceability AND (Heterogeneous OR Modeling OR Models OR MDE OR Model Driven OR Trace Link OR Requirement Engineering OR Systems Engineering OR Software Engineering)".

We decided to conduct the review by searching the following five major electronic libraries that publish journal and conference papers in the field of software and Systems Engineering:

- IEEE Xplore,
- ACM Digital Library,
- Google Scholar,
- Science Direct,
- Springer Link.

Different search engines may require the user to format the search in slightly different ways and may have different capabilities. For instance, with IEEE Xplore, the ACM Digital Library and Science direct we were able to specify that the search took place in titles, abstracts, and keywords. In the case of Google Scholar and Springer Link, the user does not control in which part(s) of papers the search is performed. We nevertheless used the same search string but we removed the parenthesis since Google scholar and Springer Link do not allow the user to input a complex search string.

3.2 Inclusion and Exclusion

The word traceability and its synonyms or terms in our search string can be used in contexts other than Software and Systems Engineering. For instance, it can be used in chemistry for

tracing chemical compounds, and in agriculture for tracing agricultural products during shipments. Therefore, our systematic search returned, similarly to any other similar search, irrelevant documents. Systematic search protocols, therefore, include a phase of inclusion and exclusion of documents [35]. In order to include or exclude certain publications, we used the options provided by the searched libraries, when such options are available, to customize the search results.

We followed some steps in order to exclude the unrelated articles. Our inclusion and exclusion procedure is based on the followings steps:

- Search the traceability article published in peer reviewed journals or conferences between the years 2000 to 2016.
- Find the relevance of the article to the proposed research question. This is done by first reading the title, and abstract of the article, and possibly the entire paper if reading the title/abstract was not enough.
- Identify whether the article discusses traceability in the context of Software Engineering and Systems Engineering or other contexts such as chemistry or agriculture.
- Include the articles that are related to traceability in the context of Software Engineering and Systems Engineering and exclude other areas.
- Identify whether the article provides valuable information in terms of a modeling technique, trace links, or a traceability tool.
- Identify whether the articles help in supporting our solution.

On the other hand, articles that did not conform to our inclusion criteria were excluded.

3.3 Conducting the review

We used the Zotero plugin for Firefox [36] to store the information about each extracted article which includes the article's title, author(s), year of publication, abstract, conference or journal name, among other things. The search results of each digital library were then exported into a separate Excel file.

Table 3: Number and percentage of traceability articles in major digital libraries

Library	Number of articles	Percentage
ACM	187	13%
IEEE	076	5%
Science Direct	383	27%
Google Scholar	471	33%
Springer	304	22%
Publications Total	**1421**	**100%**

Table 3 shows the number and percentage of the publications that we found in each corresponding library. We applied a sequence of filtering steps to get the required articles. First, we merged all search results into one Excel file, the total number of articles was 1421. Second, we used the built-in-Excel functionalities to sort and remove the duplicate articles among all libraries based on the article's title. We found and removed 356 duplicates. Third, we read the title and abstract of the remaining 1068 papers to find the articles that discuss traceability in the context of Software Engineering or Systems Engineering. Using our inclusion/exclusion criteria, we found 327 articles were closely related to the scope of our study, we found out 738 articles were discussing traceability in other disciplines such as chemistry and agriculture (see Table 4), these articles were excluded and not analyzed in the review. The remaining 327 articles were found to be closely related to our problem domain as shown in Table 5.

Table 4: Number of retrieved traceability articles

Title	Number of articles
Duplicate Articles	356 (25%)
Closely related Articles	327 (23%)
Unrelated Articles	738 (52%)
Total	**1421 (100%)**

Table 5: Number of articles distributed by each traceability topic

Area of study	Number of articles	Percentage
Traceability concepts and types	198	59.5%
Modeling traceability and traceability tools	121	37.5%
Traceability reviews and surveys	8	3%
Total	**327**	**100%**

3.4 Refining the Search Article

The 327 related articles were further refined into three areas of study in order to answer the research questions. The articles were classified based on the following areas of interest:

- Articles that discuss general traceability concepts such as terminology, traceability, and testing, traceability rules, traceability schema specification, and traceability usage, for instance, [23], [37]–[48].
- Articles that focus on traceability modeling[30], [48]–[53] traceability tools [27], [54]–[59], or trace links [28], [48], [60]–[78].
- Articles that report on traceability surveys [1], [31], [45], or traceability systematic reviews in Requirements Engineering, Model Driven Engineering, or Systems Engineering [28], [29], [32], [39], [79].

We classified the retrieved articles based on their relevance to the problem domain. This was achieved in two steps: the first step is to remove unrelated articles by scanning the title of each retrieved article and exclude out of context articles. For instance, we found articles discussing traceability, tracing, tracking, or monitoring agricultural products such as fruits, rice, meat and tobacco, aquatic products such as frozen shrimp and fish. Also, we excluded other articles that discuss traceability of chemical compounds in medicine, supply chains, and products recall. The second step was to group the remaining articles into three areas based on the research questions by reading the abstract of each article which gives us a clear picture about the amount

of research directed to a certain area. A complete list of the papers is available online on the GitHub of our Software Quality Engineering Laboratory (SQUALL)[1].

3.5 The Review Findings

The refined areas that have been shown in Table 5 can provide us with a clearer picture about the directions of research on traceability in general. Moreover, it allows us to suggest a research proposal related specifically to modeling traceability of heterogeneous artifacts.

3.5.1 Traceability Concepts and definitions

These papers are cross-referenced by the reviews and surveys in section 3.5.3. They provide us with answers to **Question 3** and **Question 4**. The articles cover general traceability concepts in the fields of Requirements Engineering, Model Driven Engineering, and Systems Engineering. We gained a better understanding of the different types of trace links, how traceability information can be captured and represented, and the different modeling techniques. The following areas of traceability are covered:

- Traceability terminology and definitions.
- Traceability and testing.
- Traceability theory and practices.
- Traceability types.
- Traceability information representations.
- Traceability modeling languages.
- Traceability maintenance.
- Traceability links recovery.
- Traceability visualization.
- Traceability automation.

[1] https://github.com/yvanlabiche/TraceabilityHeterogeneousModels

3.5.2 Modeling Traceability

The reviewed papers on modeling traceability and traceability tools provide us with answers to **Question 1**, **Question 2,** and **Question 3**. Concerning **Question 1**, we found that literature discusses modeling traceability within two contexts: generic traceability models and domain-specific traceability models. A domain specific traceability model captures case specific traceability links with well-defined semantics [80]. On the other hand, a generic traceability model has the ability to capture the traceability relationships between any type of model artifacts [80].

Regarding **Question 2** and **Question 3**, the review findings show that the most common traceability models are domain specific [60], [65]–[72], [76]–[78], [81]–[83]. For instance, Aversano and colleagues [78] presented a domain specific model for capturing traceability links between a business process model which comprises a set of enterprise activities and software components that control the business process model. Tsuchiya and colleagues [60] proposed a traceability method to recover traceability links between requirements and the source code that represents these requirements in a software product. Antoniol and colleagues [84] proposed a recovery method for trace links between documentation and source code using an information retrieval model. The authors applied their method using two case studies related to C++ and Java code. The results of the two case studies are measured using the information retrieval metrics; *Precision* and *Recall.* Jia [68] defined a method for verifying traceability between a feature model and a software architecture. Falleri and colleagues [81] defined a traceability metamodel for recording traceability information during model transformation. These are only some examples that propose domain specific or problem specific traceability model. Although these articles can provide us with useful information for modeling traceability among heterogeneous artifacts, they give only partial solutions, therefore, they cannot satisfy our goal.

On the other hand, traceability of heterogeneous artifacts is discussed in some papers [38], [48], [50], [51], [73], [80], [82], [85], [86]. Some of these articles provide description for heterogeneous artifacts, some discuss modeling the traceability of heterogeneous artifacts, and others discuss the types of trace links between heterogeneous artifacts.

With respect to the definition of heterogeneous artifacts, Taromirad and Matragkas [51] state that heterogeneous artifacts are artifacts that come from multi-domains with different formats using different modeling tools. Boronat and colleagues [87] describe heterogeneous artifacts as artifacts that come from different sources or tools such as UML models by means of visual modeling environments, relational of a database management system through the Rational Rose tool, and XML schemas. Cleland-Huang and colleagues [21] describe heterogeneous artifacts as the artifacts that come from third party tools and often come across organizational boundaries. Therefore, we can conclude that heterogeneous artifacts are usually produced by different tools in different domains.

Regarding trace links and their semantics, most articles provide methods for identification of a subset of trace links based on the problem under study: i.e., trace link semantics is problem specific. For instance, Asuncion and colleagues [73] proposed an automated technique for capturing trace link semantics among heterogeneous artifacts. The idea behind their technique for capturing semantic trace links comes from e-Science in which data provenance are collected while data sets are being processed. This technique allows the capturing of trace links by analyzing user input events such as keyboard inputs. The artifacts are assumed to be sequentially or concurrently generated or edited. Polack [80] provides a technique for classification of the types of relationships that can exist among the heterogeneous models that are involved in the software development process. The main aim of their work is to detect the inconsistencies in the relationship among these models. Their work is beneficial because of the type of relationships that exist among heterogeneous artifacts, but it does not provide a modeling technique to capture

traceability information among heterogeneous artifacts. Filho and colleagues [82] propose a traceability model based on eXtensible Markup Language (XML) to generate traceability links between UML diagrams and *i** models. The authors developed a tool to interpret the rules and check them against the two XML files, and if a certain rule is satisfied, it generates a traceability link for that rule.

Regarding the articles that discussed domain specific traceability such as [60], [65]–[72], [74], [76]–[78], [81]–[83], they are only concerned with certain types of trace links that link the artifacts under study. For instance, Sardinha and colleagues [76] discussed identifying trace links between code aspect and early aspect. Xiaofan [72], provided an automated mechanism for identifying trace links between documentation and code. Ziftci and colleagues [61] suggested a method for enhancing the semantic of traceability links involving the validation of test cases and other artifacts. Strasunskasa and colleagues [75] provided a method for discovering only dependency links in Model Driven Engineering. Concerning the methods that provide an in-depth classification for trace link, Paige and colleagues [48] suggested a rigorous method for identifying trace links in Model Driven Engineering but their article does not cover the Requirements Engineering and Systems Engineering domains.

The articles that discuss modeling traceability of heterogeneous artifacts are discussed in chapter 5, section 5.2.

The articles that are concerned with traceability tools conform to the articles that discuss domain specific modeling and domain specific trace links. For instance, Goknil and colleagues [58] described a traceability tool for generation and validation of trace links between requirements and architecture. Yrjonen and colleagues [59] suggested a traceability tool between non-functional requirements in Model Driven Engineering. We believe that these tools are produced as a result of the abovementioned domain specific modeling techniques. Other articles [54], [56], [59] provided evaluations or suggestions about traceability tools in general.

3.5.3 Traceability Reviews and Surveys

The traceability reviews and surveys that we reviewed provide us with an answer to **Question 3**; they provide comprehensive information about the work done on traceability, which can be summarized as follows:

- Analyze the current state of the art in traceability management in Requirements Engineering and Model-Driven Engineering.
- Present common definitions, challenges, and techniques for traceability in Software Engineering.
- Show validation of industrial practices on traceability through series of interviews.
- Provide insights on traceability practices among practitioners.
- Highlight the approaches for maintaining, automating, and managing traceability information.
- Track requirements at different phases of the product development life cycle.

We can conclude from the results shown in Table 5 that the majority of the reviewed research papers are focusing on general traceability terms and concepts (59%), while the next larger group of papers includes papers that focus on modeling traceability and traceability link only (37.5%). Although there are some paper discuss heterogeneity of artifacts, we found eight articles that discuss modeling traceability of heterogeneous artifacts [48], [50], [51], [68], [73], [88]–[90]. The modeling approaches in these papers are discussed in detail in sections 5.1 and 5.2. The ideas they describe can be incorporated in our traceability model design.

The research effort on modeling traceability among heterogeneous artifacts and characterizations of trace links is minimal. The complexity of systems nowadays requires designing a traceability model that must be oblivious of the heterogeneity of the model's elements to be traced.

Also, we found there is a need to define a trace links taxonomy, which can provide semantics to accommodate different situations and different possibilities or even new ways of characterizing trace links. For instance, characterizing trace data is likely to be domain, organization, or even project dependent. A solution should also provide semantics for any link that exists between heterogeneous artifacts whether these artifacts are coming from the Requirements Engineering, Model Driven Engineering, or Systems Engineering domains. A traceability modeling solution should combine all existing trace link types in a way to eliminate duplications of the current classifications. Some consolidation of existing trace link semantics modeling solutions is warranted. We will elaborate on these findings in the subsequent chapters.

3.6 Summary

In this chapter, we reviewed the research on traceability between the years 2000-2016 in order to answer the proposed questions regarding traceability in Software Engineering and Systems Engineering. Our search aims to identify the articles that discuss traceability concepts, types, modeling techniques, and tools. In terms of modeling traceability, the review investigated the articles that discuss generic and domain specific traceability modeling. We found that domain specific traceability modeling is preferred in situations in which traceability is needed between certain types of artifacts. For instance, tracing requirements to their corresponding use cases. On the other hand, generic traceability modeling is required in tracing heterogeneous artifacts of complex systems. Our review shows that the existing generic modeling techniques have some drawbacks, for instance, some modeling techniques lacks extensibility (i.e., limited to certain types of artifacts or trace links), others do not propose a traceability model (e.g., discuss only the need for a generic traceability model or propose some modeling requirements).

Regarding the traceability tools, we found most of the proposed tools accommodate traceability for domain specific artifacts (i.e., their usage is limited to certain types of artifacts). We believe these tools are byproducts of the articles that discuss domain specific traceability. Finally, with

respect to the surveys and reviews, these articles help the reader gain a better understanding of traceability concepts, definitions, and classifications.

4 Trace Links Classification

In general, relations between artifacts are classified based on the development phase or the abstraction level (i.e., horizontal and vertical). However, other classifications are introduced to fit the Requirements Engineering, Model Driven Engineering, and Systems Engineering needs. The trace links classifications which we found are either problem oriented, i.e., tailored to special cases and are not applicable within a general context (e.g., between requirements and source code), or target one domain only (e.g., Requirements Engineering or Model Driven Engineering). This chapter summarizes our effort in collecting and organizing these classifications in order to build a trace links taxonomy. The classifications have some inconsistences and duplications in many locations, however, we will tackle these issues during the implementation of the trace links taxonomy in chapter 8.

4.1 Requirements Engineering Classifications

In Requirements Engineering, relationships between artifacts are discussed extensively [5], [19], [20], [91]–[102]; among these papers, we found two papers that discuss trace links classifications [20], [96]. Spanoudakis and Zisman [20] classified requirement traceability links into eight categories which include various link types based on their support to certain software activities such as analysis, validation, or supporting stakeholders decisions. A summary of these links is shown in Table 6 which includes the following types:

- The dependency relations relate artifacts in which the existence of one artifact relies on the existence of the other. This type can be used to relate requirements to each other, or requirements and design elements (artifacts). Dependency relations are one of the most widely used in Requirements Engineering and have different uses and forms [20]. For instance, Xu and Ramesh [99] use dependency relations in workflow management systems between business process objects, decision objects, and workflow system objects. Dependency relations

Table 6: Classifications for RE trace links

	Dependency types	Evolution types	Generalization types	Satisfaction types	Other types
[19]					Contribution
[90]	Requires-feature-in				Overlap
[99]	Dependency				Rationale
[104]	Dependency				
[100]	Dependency				
[101]	Dependency				
[102]	Causal-dependency conformance	Non-causal conformance			
[91]				Satisfy Derive Refine	
[92]	Developmental	Temporal	Containment		Adopt
[93]	Correspondence				
[105]					Rationale
[37]				Satisfy Establish Contribute	
[103]					**Inconsistency**
[82]					Adopt
[20]	Require-Execution-Of, Can Partially-Realize			Overlap, Require-Features-In	

are used in product and service families [94] to support the management of variability, i.e., ensuring that the changed artifacts reflect the intended system functionality. Knethen and colleagues [103] suggested their use between documentation entities such as requirements and use cases and logical entities such as functions for fine grained impact analysis. Pohl and Alexander [100], [104] use them to link requirement scenarios and code, and Riebisch and Philippow [101] use them to support the design and implementation of product lines. Other forms of dependency links are suggested in the literature. For instance, Spanoudakis and colleagues [98] refer to them as requires-feature-in relations, as they link parts of use case specifications to customer requirements specifications. Also, they are called causal conformance by Maletic et al. [102], who use them to link documents that represent an implied ordering (e.g., bug reports cannot be produced before implementation report). Gotel and colleagues [92] referred to them as developmental relations which are used to trace requirements to other artifacts in another phase of the development lifecycle. Finally, they are

referred by Constantopoulos et al. [93] as correspondence relations which link requirements, design, and code artifacts.

- The evolutionary relations link requirements in which one requirement replaces another. This category contains the replace, based-on, formalize, and elaborate trace links. Pinheiro showed the use of replace and abandon trace links during requirements evolution. A requirement is replaced by another if a mistake is discovered, and the original requirement will be abandoned. Gotel [92] called the evolution relations temporal relations which refer to linking requirements in term of their historical order. Maleic and colleagues [102] called the evolution relations as non-causal conformance relations to link documents which conform to each other.
- The generalization/refinement relations show how complex system components can be divided into other artifacts, or how one artifact can be refined by another. In Ramesh and Jarke's classification [96], generalization/refinement is considered a dependency abstraction link. Gotel [92] refers to them as containment relations since they are used to link composite artifacts and their components.
- The satisfiability relations link artifacts that are constrained by each other, e.g., a requirement that complies with the conditions of another requirement. This type is classified as a product-related trace link to relate requirements to design artifacts [96]. Satisfiability has sub-types such as the establish (cardinality 1-1 between two artifacts) and contribute (cardinality 1-m between artifacts) relations [37]. Pinheiro [91] defined satisfiability based on derivation, e.g., if a requirement is satisfied then its derivation is satisfied, and refinement, i.e., if a requirement refines another requirement, then satisfying the first requirement, implies satisfying the second. Spanoudakis et al. [98] use the overlap relations in an analysis model to relate use cases and classes. Gotel and Finkelstein [92]

called them adopts relations; they are used between artifacts in which a target artifact embeds information of the source artifact.

- The conflict relations link two artifacts that have a conflict, such as two requirements that are conflicting with each other [96]. Special types of conflict relations such as based-on, affect, resolve, and generate are used to provide conflicts resolution between conflicting artifacts. Kozlenkov and Zisman [95] referred to conflict relations as inconsistency relations.
- The rationalization relations link two artifacts in which one of them captures the rationale behind the creation or evolution of the other. Letelier [105] used this type to relate rationale specification artifacts (e.g., decisions, assumptions) to software specifications at different levels of granularity (e.g., document or part of a document, diagram, or a model). Rationalization relations are used also to relate design rationales to design artifacts [99].
- The contribution relations relate requirements and their stakeholders [98], for instance, linking requirements to the stakeholders who contributed them.

Another classification for trace links in Requirements Engineering is introduced by Ramesh and Jarke [96], see Table 7. The authors' classification is based on a study about the use of trace links by different organizations that involve high-end and low-end users with respect to their

Table 7: RE Process-related and Product-related classification

Ref	Product-Related Links		Process-Related Links	
[96]	**Satisfaction**	**Dependency**	**Evolution**	**Rationale**
	Satisfy, Interface-With, Defined-By, Created-By, Allocated-To, Derived.	Temporal, Depend-On, Trace-To Contain, Used-by, Performed-by	Evolve-To, Modify, Define, Elaborate, Derive	Generated-By, Resolve, Supported-By, Affect

traceability practices. They classified traceability links into two main categories: process-related and product-related links. The process-related links can be discovered by observing the history

Table 8: Model Driven Engineering trace links classifications

<table>
<tr><th>Ref</th><th colspan="5">Trace Link Type</th></tr>
<tr><td rowspan="4">[97]</td><td>Implicit</td><td colspan="4">Explicit</td></tr>
<tr><td rowspan="3">Update, Query, Creation, Composition, Deletion, Model-to-Model, Model-to-Text</td><td colspan="3">Model-to-model</td><td>Model-to-non-model artifact</td></tr>
<tr><td colspan="2">Static</td><td>Dynamic</td><td rowspan="3">Satisfy
Allocated-To, Explain, Perform, Support</td></tr>
<tr><td>Consistent-with</td><td>Dependency</td><td rowspan="2">Call, Notify, Generate, Synch-With</td></tr>
<tr><td></td><td>Refine, Import, Export, Usage, Is-A, Has-A</td></tr>
</table>

of operations performed in a process. The product-related links describe the relationships between artifacts independently of their creation. Furthermore, the authors identified sub-categories of these two main categories. The process-related category is divided further into evolution links and rationale links, which we described earlier. On the other hand, the product-related links are decomposed into two main types: satisfaction links and dependency links, which we also described earlier. The authors deduced other types of relations from the abovementioned categories based on the use of low-end and high-end users. Model Driven Engineering Classifications

In Model Driven Engineering trace links are generated explicitly by adding additional code into the transformation, or implicitly through the transformation tool [97]. Paige and colleagues [97] classified Model Driven Engineering trace links into implicit and explicit trace links (Table 8). The implicit trace links are classified based on *Query*, *Transformation*, *Composition* (merging), *Update*, *Deletion, Creation*, M*odel-to-text*, and *Model-to-model* operations. The explicit trace links are classified as *Model-to-model* links which relate Model Driven Engineering artifacts with each other, and *Model-to-artifact* which relate Model Driven Engineering artifacts with non-Model Driven Engineering artifacts such as linking a UML model to its requirement(s). The model-to-model links are further classified into *Static* and *Dynamic* links. A *Static* link represents a relationship that stays the same over time between model elements such as *Consistent-with* (e.g., two models remain consistent with each other), and *Dependency* in which

the structure and meaning of one model depend on another model. This type is further classified into the following trace links: *Is-A* (sub-typing), *Has-A* (e.g., references), *Part-Of, Import, Export, Usage*, and *Refinement*. A *Dynamic* link represents a relationship that might evolve over time.

This category has several types of links such as *Call* (e.g., a model calls the behaviors provided by another), *Notify*(e.g., changed artifacts that need intervention), and *Generate* (e.g., links two models where one model produces the other).

The model-to-artifact category contains the satisfy trace link which indicates that an artifact such as a requirement is satisfied by a model, allocated-to which relates information in a non-model artifact to a model that represents that information, performs which relates a task to a model that carries the task, explains and supports trace links which are used when a model is explained by a non-model artifact.

4.2 Systems Engineering Classifications

In Systems Engineering, Mason et al. [106] introduced a traceability taxonomy that includes the directional and temporal traceability. They extended the definitions of vertical and horizontal traceability by introducing the terms *micro, macro, inter, and intra*. *Micro* and *macro* differentiate traceability within and across decomposition levels. *intra* and *inter* differentiate traceability within and across system descriptions (i.e., interactions between systems). For instance, the inter-micro-horizontal traceability refers to the ability to describe and navigate relationships across system descriptions, within a decomposition level, between development or assessment artifacts of the same type. Temporal traceability represents links between synchronized artifacts, for instance, a link between an artifact and its subsequent revised artifact which is produced based on an event.

Table 9: Trace links types in Systems Engineering

<table>
<tr><th>Ref</th><th colspan="6">Trace Link Type</th></tr>
<tr><td rowspan="4">[97]</td><td rowspan="4">Temporal</td><td colspan="5">Directional</td></tr>
<tr><td colspan="3">Vertical</td><td colspan="2">Horizontal</td></tr>
<tr><td colspan="2">Micro</td><td>Macro</td><td>Micro</td><td>Macro</td></tr>
<tr><td colspan="2">Inter / Intra</td><td>Inter / Intra</td><td>Inter / Intra</td><td>Inter / Intra</td></tr>
<tr><td>[11]</td><td>SysML</td><td>Requirement - Requirement</td><td>Req. Model Element- Requirement</td><td>Design Element - Requirement</td><td colspan="2">Test Element-Requirement</td></tr>
<tr><td></td><td></td><td>Derive</td><td>Refine</td><td>Satisfy</td><td colspan="2">Verify</td></tr>
</table>

In addition, Systems Engineering employs the SysML trace links since it is used for modeling Systems Engineering complex systems. The classification of trace links between model elements follows the same Requirements Engineering pattern. Requirements in SysML [11] are classified into different categories using a Stereotype. For instance, requirements can be stereotyped to represent operational, functional, performance, design constraints, reliability, and maintainability requirements. The SysML requirement diagram models these requirements in hierarchies. Therefore, the semantic of a trace links in SysML depends on the type of the relationship between two requirements, or between a requirement and a design element or a test case for example. There are four main types of trace links in SysML: a derived trace link links a requirement to its derived one; a satisfy trace link relates a requirement to a design element that satisfies it such as a use case; a verify trace link, links a requirement to a test case that verifies it; a refine trace link links a requirement to a design element that refines it. It is important to differentiate between the usage or the semantic of derive and refine trace links. A derive trace link is used to link two requirements while a refine trace link links a requirement to a design element that refines it or vice versa such as a text requirement with an activity diagram [107], Table 9 summarizes the trace links types in Systems Engineering.

4.3 Summary

We have seen various definitions for traceability in Requirements Engineering, Model Driven Engineering, and Systems Engineering. These definitions are based on the type of the traceable

artifacts involved. For instance, in Model Driven Engineering model transformation, some artifacts are models which conform to the same metamodel, therefore, the definition of traceability is customized to fit this context. Accordingly, there are variations in the types of trace links that relate these artifacts which results in various classifications for trace links in Requirements Engineering, Model Driven Engineering, and Systems Engineering domains. Although there are similarities and duplications in trace link semantics among these domains, we could not find a specific research that combines all trace links in one taxonomy. We will discuss in chapter 8 how these links can be combined into one taxonomy that provides a well-defined semantics for each trace link.

5 Traceability Modeling in the Literature

The work in this chapter is split into three sections: section 5.1 discusses domain specific traceability models, and section 5.2 discusses generic traceability models, all from the literature. The reference modeling solutions that we discuss here were found during our systematic literature search (chapter 3). Section 5.3 provides our analysis of the features of both types.

5.1 Domain Specific Traceability Modeling

Domain specific traceability model captures case specific traceability links with well-defined semantics [89]. This section discusses the efforts in domain specific traceability (Meta)modeling. Pavalkis and colleagues [49] defined a traceability metamodel for linking artifacts in an instance of the Business Process Model Notation (BPMN) [108]. To define traceability links, the authors relied on one UML extension mechanism, namely derived properties: a derived property extends the UML metaclass *Property* and can be specified with an OCL [109] expression for instance. They defined traceability link rules between specific BPMN model elements (i.e., *resources* and their *process*), between *participants* involved in *messages* (i.e., message sender and receiver). In order to provide tool support for their BPMN traceability technique, the authors relied on the MagicDraw Domain Specific Language engine [110]. The authors argued the proposed approach is extensible and customizable since new rules, possibly involving BPMN elements that were not originally considered by the authors, can be defined using derived properties. However, the approach is not extensible or customizable enough to completely solve traceability among heterogeneous artifacts from different domains. It is specific to BPMN. Also, derived properties may not be enough to cover each single traceability model needs. For instance, this model cannot accommodate vertical

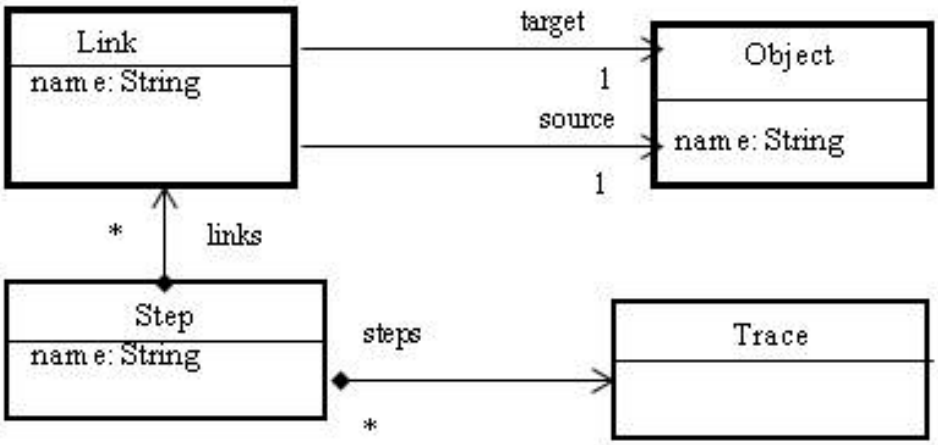

Figure 2: Transformation chain traceability model, source [81]

traceability (i.e., linking artifacts from a BPMN model to other model types), it only accommodates horizontal traceability links that link only elements within a BPMN model.

For validation purposes, the authors defined a couple of traceability links that one would need when using the BPMN notation. They showed how the links would be defined using their solution, and how they would be modeled with MagicDraw. The authors, however, do not show that their solution would help for any kind of traceability link one would need outside of a BPMN context.

Falleri and colleagues [81] defined a traceability metamodel for recording traceability information during model refactoring, where a model conforming to a certain metamodel is transformed, possibly through several refactoring/transformation steps, into an improved, refactored model that conforms to the same metamodel, see Figure 2. They employed Kermeta [111], a tool to capture the traceability information between the source and target models during a model transformation. In the context of a model transformation involving several transformation steps, the authors model a trace as a sequence of steps that make up the transformation. Each step contains a series of traceability links between one source model element and one target model element. This metamodel is good for capturing traceability links during model transformation; especially it can capture a sequence or a chain of links between

source and target artifacts. However, it is domain specific and can only do that. Also, it requires

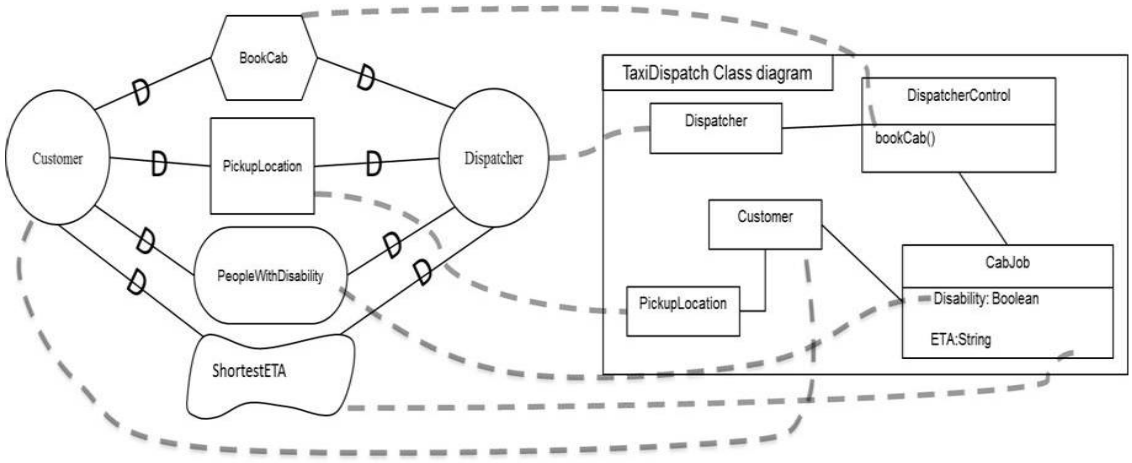

Figure 3: Traceability Example: *i (excerpt) model (left), UML (excerpt) class diagram (right), traceability links (grayed dashed lines), source** [51]

that the source and target artifacts conform to the same metamodel.

In addition, the traceability metamodel does not provide any semantics or constraints on the type of the trace links. The validation of the metamodel was performed by first writing a simple transformation code in Kermeta that maps each UML class into a database *table*, and each class attribute into a database *column*. The result of the mapping is then used to generate a directed graph that shows the trace links between the artifacts of the two models using *graphviz* [112], an open source graph visualization software.

Filho and colleagues [82] propose a light-weight XML-based approach to generate bidirectional traceability relations between UML and *i** models. The authors identified a set of traceability relations between organizational models expressed in the *i** language and use-cases, and class diagram elements expressed in the UML, see Figure 3. For example, actors in *i** can be linked to actors in the use-case diagram. The approach is based on converting the *i** and UML models into XML[113] files (i.e. *i**.XML, UML.XML) and writing traceability rules in XML. The authors developed a tool to interpret the rules and check them against the two XML files, and if a certain rule is satisfied, it generates a traceability link for that rule. Although Filho and colleagues provide an approach to generate traceability links between two

heterogeneous models (i.e., *i** and UML), their approach is tailored only to these two types and cannot be extended to include other models types.

5.2 Generic Traceability Modeling

A generic traceability (Meta)model has the ability to capture the traceability relationships between any types of model elements [89].

Paige and colleagues [48] described an approach for defining semantically rich trace links between models, where the models themselves may be constructed using diverse modeling languages. Trace links are semantically rich because their types conform to a case-specific traceability metamodel, and the case-specific metamodel should be accompanied by a set of case-specific correctness constraints, which express validity requirements that cannot be captured by the metamodel itself. The reason for having a case-specific traceability metamodel is to prevent from generating illegitimate links. Otherwise, as the authors argue, a general-purpose trace metamodel can connect any number of elements of any type in any model. The authors used a method called Traceability Elicitation and Analysis Process to elicit and analyze the traceability relationships within a Model Driven Engineering process. The elicitation process involved studying the domain and available scenarios to help identify trace links. The analysis involved understanding the trace links' semantics and their relationships to other trace links. The research of Paige and colleagues resulted in a taxonomy of trace link types (see [48] for details). For validation purposes, Paige and colleagues used a method for identifying trace links in the Requirements Engineering phase, which they split into early activities, modeling with *i** [52], and later activities, modeling with the UML (specifically the *Class* diagram).

Their approach has some drawbacks, whereby the types of traceability links from *i** artifacts and the UML *Class* diagram artifacts need to be in the metamodel itself, making the metamodel difficult to evolve to accommodate new artifacts, or new types of models (recall our requirements). Particularly, if ten different types of traceability links need to be accounted for,

which can be considered a small number given that for instance a link between an *i* actor* and a *UML Class* is considered as one type (one metaclass), then the metamodel contains ten different traceability link metaclasses; if traceability links must be classified according to orthogonal classifications, which is very likely according to other authors, then the number of traceability link metaclasses equals the cross product of the sizes of the classifications. Moreover, the model cannot accommodate traceability in case of a model-to-model transformation.

Anquetil and colleagues [90] introduced a traceability reference metamodel that supports general traceability for aspect-oriented and model-driven Software Product Line (SPL). It is used to create an infrastructure for the development of software systems that share some commonalities. The SPL metamodel in Figure 4 can be used to follow artifacts in forward and backward directions through any single development step (i.e., horizontal traceability) and between development steps (i.e., vertical traceability); it shares the common traceability aspects among SPL projects. The metamodel is composed of a *TraceModel* that stores *TraceableArtifacts* and *TraceLinks*. The *TraceableArtifacts* metaclass represents source or target artifacts. A *TraceableArtifact* has a Universal Resource Identifier (URI) that describes the actual place of the artifact and a unique identifier (id) to the traceable artifact inside this location (e.g., a requirement inside a document). The *TraceLink* metaclass represents an explicit relationship between a set of source artifacts and a set of target artifacts. The semantics of each *TraceableArtifact* and each *TraceLink* are expressed via a *TraceableArtifactType* and a *TraceLinkType* respectively. The *Scope* and *ScopeArea* metaclasses are used to prevent illegal links between certain artifacts. The *Scope* may contain many Scope areas, each defining which type(s) of source and target artifacts are allowed to be linked together. The *TraceContext* metaclass contains the justification behind the use of a *TraceLink* or *TraceableArtifact.*

The authors claim that this metamodel is generic and extensible for any specific link types. However, the model has some drawbacks: (a) it does not consider traceability between model transformation artifacts, and (b) it does not accommodate a more rigorous link type classification in which they assume a trace link can belong only to one type, whereas, in some cases, a trace link can have more than one type (i.e., multiple inheritance).

Mate and colleagues [88] proposed a traceability model based on the model-driven architecture for tracing user requirements and data sources in Data Warehouses. The development of a data warehouse follows three approaches: First, a bottom-up approach is used in which every element used for the implementation of the data warehouse comes from a single source only (i.e., requirements or data sources). Second, a top-down approach, which is the opposite of bottom-up, focuses on the user requirements while ignoring the data sources. Third, a Hybrid approach merges the previous two approaches and requires information on user requirements

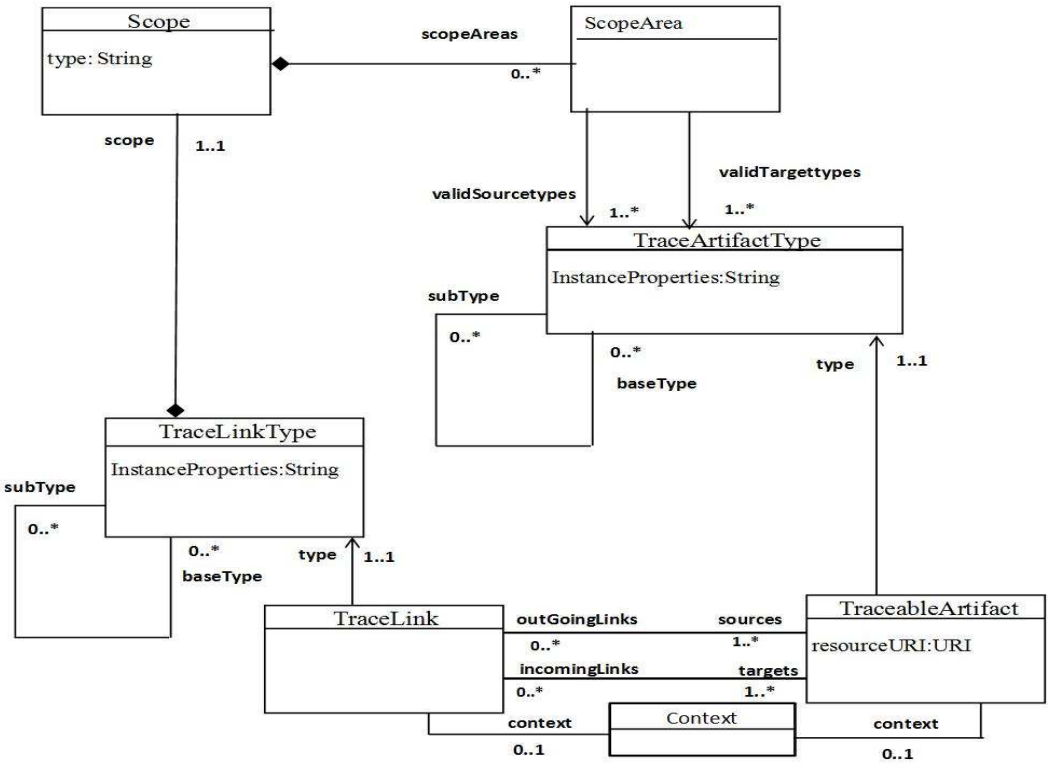

Figure 4: SPL traceability metamodel, source [90]

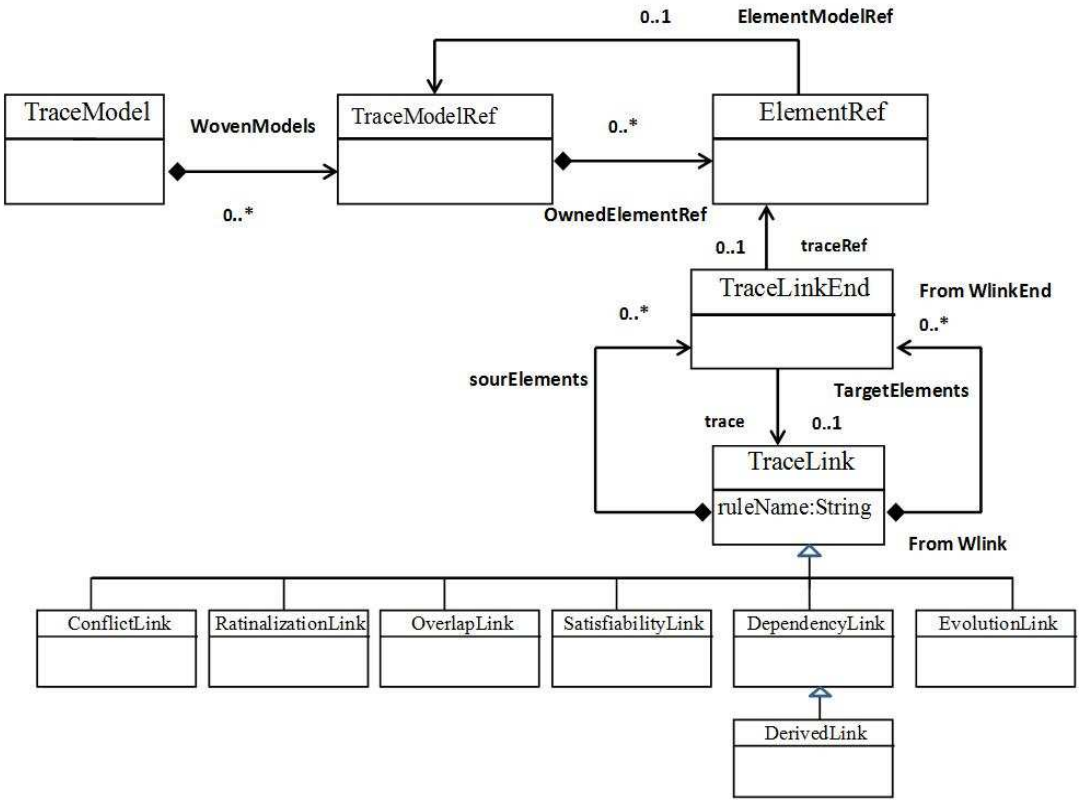

Figure 5: Traceability Metamodel for Data Warehouses, source [88]

and data sources. Therefore, there is a need to trace which parts of the data warehouse relate requirements to data sources.

The data warehouse traceability metamodel in Figure 2 shows a *TraceModel* class that has zero or more woven models represented by the *TraceModelRef* class. Each *TraceModelRef* has zero or more references (the *ElementRef* class) that identify the elements linked by the traces. Class *TraceLinks* defines the relationships between the elements in the woven models. The *TraceLinkEnd* class represents the source and target artifacts for a trace link. The *TraceLink* class is a generalization for many trace link types such as evolution, dependency, overlap, conflict, satisfiability, and rationalization links, which the authors defined.

This metamodel is domain specific since it concerns only with certain types of links that target data warehouse applications. It cannot accommodate traceability between artifacts that require different trace link types such as notify or call links. The metamodel has no sense of direction on the trace links between source and target artifacts as it does not specify which artifact is the

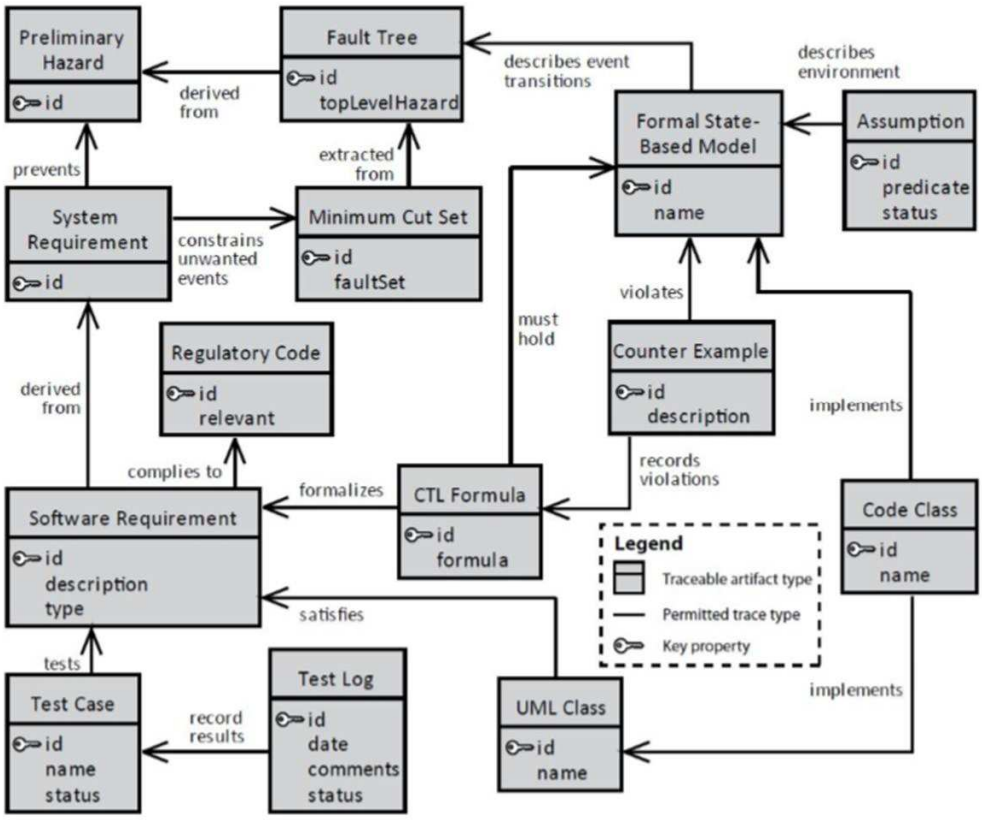

Figure 6: TIM traceability model for critical systems, source [51]

source and which artifact is the target, and it does not impose any constraints on the type of artifacts that can be linked.

Taromirad and colleagues [51] proposed a traceability model for capturing relationships between heterogeneous artifacts in safety critical systems. Their approach focused on creating a Traceability Information Model (TIM) to manage traceability information for software projects of multiple domains. It provides the necessary steps to obtain traceability information from different sources and to record it in the TIM model. The model shows how to trace artifacts produced during software development to system engineering artifacts. The TIM model in Figure 6 shows that software requirements can be traced to (derived from) a system requirement which can be traced to a preliminary hazard artifact. The TIM model is built on top of the other models that contain the artifacts to be traced. The authors proposed four steps for collecting and capturing the traceability information by the TIM model: First, for each

domain, identify the types of artifacts that need to be traced based on the TIM model (i.e., software, hardware artifacts). Second, for each domain, complete the inter-domain missing traceability information such as trace links types, artifacts, validation rules or constraints. Third, define and complete the trace link types and the constraints between the multiple domains. Fourth, define the mapping between the multiple domains and the TIM model by specifying how each artifact, trace link, or a constraint relates to TIM model classes. Fifth, once the mapping is complete generate the traceability information of the TIM model.

This model is beneficial for the design of a generic traceability model since it provides a map for identifying the types of traceability information to be collected and recorded. However, the model did not specify for example how source and target artifacts can be linked (e.g., specify cardinality or direction), how to classify a trace link or an artifact, or how to add a constraint to a trace element.

Drivalos and colleagues [89] presented the Traceability Metamodeling Language (TML) for defining the syntax and semantics of traceability metamodels, see Figure 7. In TML, traceability modeling involves the *Trace* metaclass that acts as the root of a TML model and holds all its *TraceLink* instances. The *Context* metaclass captures custom information about a *TraceLink* or a *Trace*. The *TraceLinkEnd* represents the source and target artifacts, necessarily from an Ecore-based model, which can be linked in a *TraceLink*. The *TraceLink* metaclass represents a traceability link between a minimum of two *TraceLinkEnd*. In addition, constraints can be expressed in the Epsilon Validation Language (EVL) [114], an extension of the Object Constraint Language (OCL) [109].

The authors validate their approach with a case study to trace artifacts between a *class* diagram (using a *class* diagram metamodel) and a *component* diagram (using a *component* diagram metamodel). The TML is instantiated in Figure 8 by creating a [89] *ComponentClassTraceMetamodel* instance from the *Trace* class; two instances of *Tracelink*

called *ComponentPackage* and *ServiceMethod,* and four instances of *TraceLinkEnd* called *Package*, *Component*, *Method,* and *Service.* The *ComponentPackage* link is linking a package instance of type *Package* from the class metamodel with a component instance of type *Component* from the component metamodel. Similarly, the *ServiceMethod* link is linking a method instance of type *Method* from the Class metamodel with the service instance of type *Service* from the Component metamodel. In addition, instances of the *Context* and *ContextData* classes are created to specify other attributes and constraints on the trace links.

This metamodel provides the metaclasses that are required to capture traceability information. However, it does not provide enough information about the direction of the trace link, in other words, the metamodel does not specify which of the *TraceLinkEnd* (i.e., artifacts) is a source or a target. In addition, it does not provide a mechanism for capturing traceability information in the case of a model transformation (transitivity of links). The metamodel can handle any

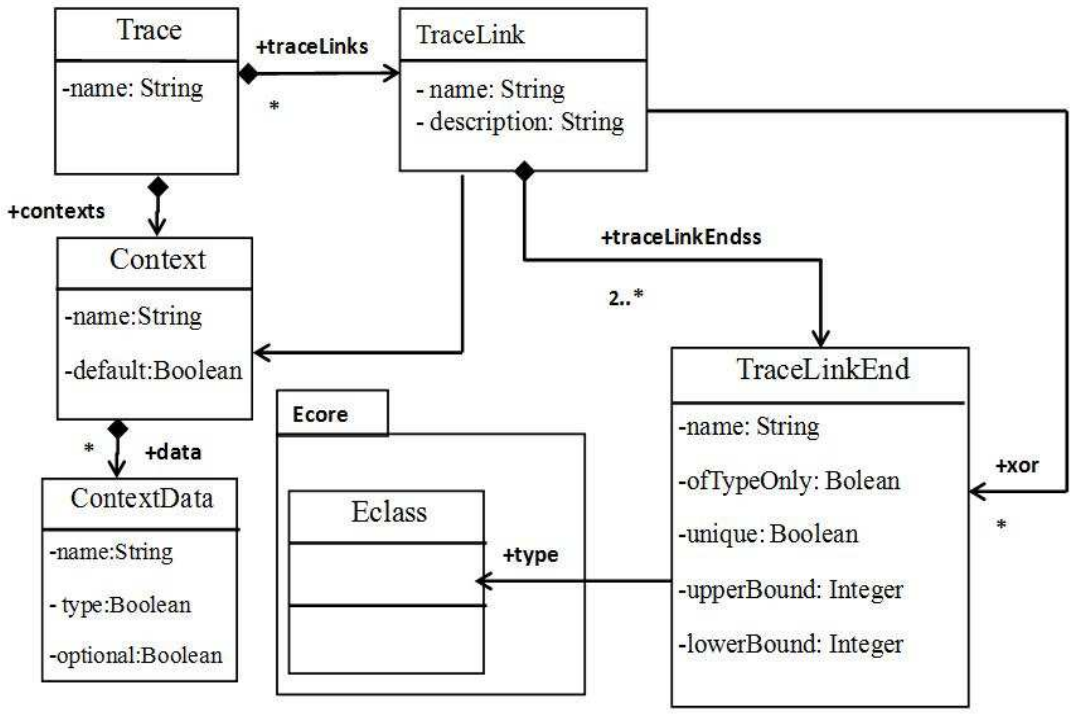

Figure 7: TML Traceability model, source [48]

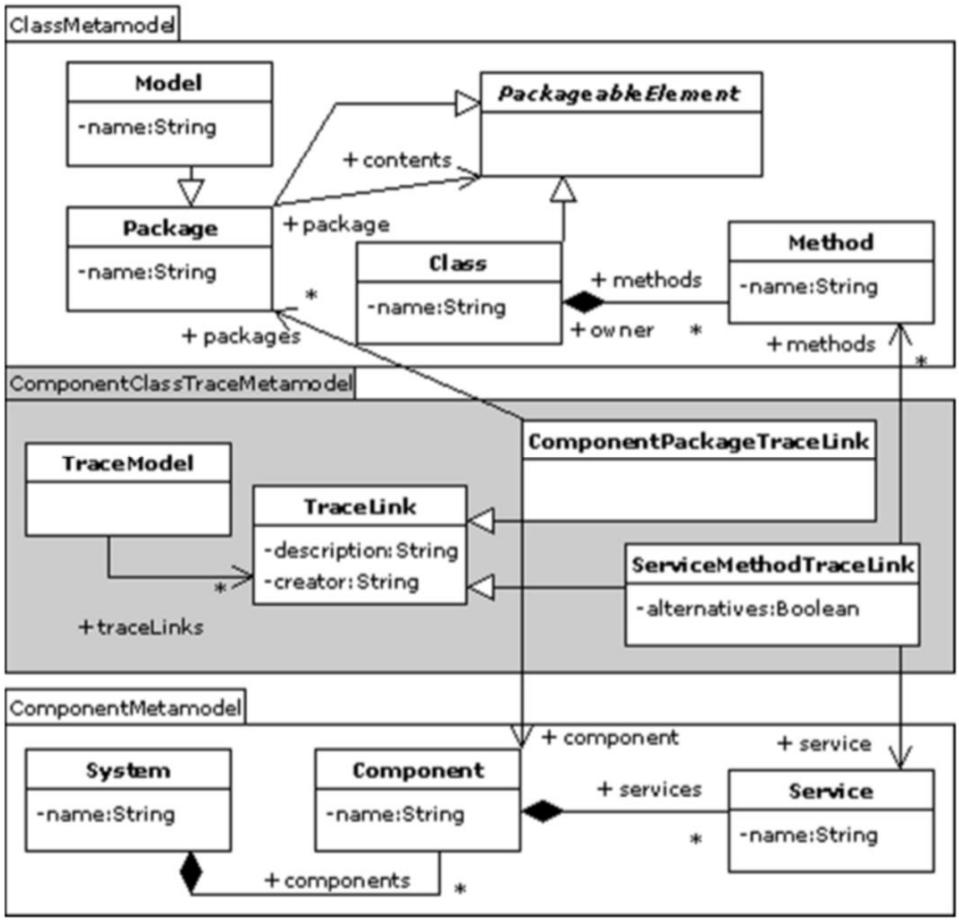

Figure 8: Traceability model for component -package example, source [96]

type of links, even newly created ones since this can be recorded in the description attribute of class *TraceLink*. However, only artifacts in MOF-based models can be linked.

Dubois and colleagues [50] proposed a requirement traceability model called the DARWIN4Req for tracing heterogeneous artifacts in automotive systems. The proposed traceability model is composed of three interrelated models connected to each other. The first model is the requirements model which represents the initial system's requirements. The second model is the solution model which represents the design artifacts that satisfy the requirement model. The solution model is implemented using one of the standard languages: UML, SysML, or MARTE [115]; which all have a MOF-based metamodel. The third model is the verification and validation model which is used to ensure that the application conforms to the requirements specifications. It integrates other models such as Simulink for testing purposes and is connected to the requirements and solution models. The authors assumed three types of

trace links for linking requirements to each other, which are the *copy, derive, and decompose*. In addition, they assumed, linking requirements to model using *satisfy*, *refine*, and *verify* trace links. The validation of the model is done through a simple case study from the automotive industry for the anti-blocking system. The case study specifies the requirements of the anti-blocking system. Moreover, it traces the requirement model to the solution model and verification and validation model. We believe this solution is inflexible since the solution is restricted only to specific types of trace links. We need a flexible traceability solution that allows the use of any type of trace links without restrictions.

Asuncion and colleagues [73] proposed an automated technique for capturing trace link semantics among heterogeneous artifacts. The idea behind their technique in capturing semantic trace links comes from e-Science in which data provenance are collected while data sets are being processed. This technique allows the capturing of trace links by analyzing user input events such as keyboard inputs. The artifacts are assumed to be sequentially or concurrently generated or edited. Although the authors claim this technique can capture trace links semantics among heterogeneous artifacts, there is no proposed (Meta) model to clarify this technique. Therefore, we cannot confirm whether the solution can satisfy our needs.

5.3 Analysis of Existing Research on Modeling Traceability

Research papers on traceability are divided into two categories: (a) papers that focus on aspects that are necessary for building traceability (Meta)models such as trace links semantics, traceability rules, traceability definitions, and traceability schema specification [6], [9], [18], [19], [22], [23], [26], [89], [106], and (b) papers that focus on the design of domain specific and generic traceability models and (Meta)models [48]–[51], [81], [82], [89], [90], [116]. Additionally, some of the papers lack the ability to specify the semantics of trace links [49], [81], although traceability link semantics have been discussed by others [9], [48], [80], [89].

Table 10: Results of analyzing existing traceability models

Reference	Traceable Models	Metamodel Technology	Tool Support	Validation	Extension to new link types without changing metamodel	Important design features
[48]	Ecore based	UML class diagram	Eclipse	Partial instantiation, couple of case studies	No	Trace links classifications, linking heterogeneous models
[49]	BPMN	UML derived property	MagicDraw	Partial instantiation, one case study	Yes, but limited to what can be done with derived properties	New traceability rules and relations in BPMN
[89]	MOF-based models	UML class diagram	Eclipse	Partial instantiation, one case study	Yes, but limited to MOF	Modeling link types
[81]	MOF-based models	UML class diagram	Kermeta	Partial instantiation, one case study	Yes, but limited to MOF	Sequence of links in model transformation
[82]	Heterogeneous	XML	Prototype tool.	Partial instantiation, one case study	No	Linking of heterogeneous models
[90]	Heterogeneous	UML class diagram	Eclipse	Partial instantiation, one case study	Yes, but limited to a type and subtype only	Trace links classifications, linking heterogeneous models
[88]	Heterogeneous Models	UML class diagram	Eclipse plug-in	Partial one case study	No	Linking of heterogeneous models
[50]	Heterogeneous Models	UML class diagram	Papyrus	Partial simple case study	No	Assumed heterogeneous artifacts.
[51]	Heterogeneous artifacts	UML class diagram	ECOR	Doesn't exist	No	Specifying a traceability metamodel on the top of multi-domains

Table 10 summarizes the features of the research papers that are important for our context, specifically: the types of models that can be traced, the technology being used to represent traceability information, the possibility of tool support for the proposed approach, the kind of validation that was performed, whether the solution (i.e., metamodel) can be extended to new traceability links and various modeling languages without having to change the solution itself (recall from the Introduction, this is one of our requirements), and what features we consider are interesting and that we want to incorporate in our own solution. We can conclude that

domain specific traceability models have some benefits; since they model traceability for a specific problem domain, this simplifies model design. However, there are situations in which domain specific modeling falls short for satisfying the user's need, such as modeling traceability of heterogeneous artifacts. We found out that we cannot find a solution to our problem: accommodating artifacts that belong to widely different kinds of models; accommodating various classifications of traceability links; accommodating new classifications that may be needed in the future.

5.4 Summary

This chapter presents the research in domain specific and generic traceability modeling. We have analyzed both categories and provided the characteristics of each modeling technique. Although domain specific modeling is useful for relating specified artifacts, it cannot accommodate complex systems that involve heterogeneous artifacts. On the other hand, the generic traceability modeling techniques that have been discussed are useful to trace heterogeneous artifacts. However, there was no discussion about each model requirements. Also, the validation criteria for these model was either partial or does not exist. The benefits and drawbacks of each technique were presented in the analysis section.

6 Traceability Survey

This chapter describes our effort in understanding the requirements of a generic traceability model by designing a survey that aims to collect information related to the traceability of heterogeneous artifacts. This survey complements our systematic literature review [117] in chapter 3 about the traceability of heterogeneous artifacts. The feedback that we get from the survey and the results of the systematic literature review play important roles in supporting the design decisions of a generic traceability model. The survey is carried out according to the guidelines for conducting a software engineering survey. According to Robson [118], a Software Engineering survey has two parts; survey planning and carrying out the study. Survey planning consists of ordered steps which start by defining the objective of the survey, identifying the research questions, identifying the target audience, and designing the survey. On the other hand, carrying out the study involves publishing the survey, collecting data, analyzing the data, and reporting the results. We provide here a description of all these aspects.

6.1 Survey Planning

The survey planning starts by stating the research question. Our research question comes from the thesis research questions explained in section 1.1 which is related to modeling traceability of heterogeneous artifacts. Particularly, we are concerned with designing a traceability model that should be extensible (i.e., accommodate future traceability needs without changing the design of the model). In addition, it should be able to relate any artifact type (i.e., homogeneous or heterogeneous) in any direction and at different levels of granularity. Based on these needs, we searched the literature and described our findings in section 3.5. However, we would like to have an insight about traceability of heterogeneous artifacts from the point view of (industrial) practitioners. Therefore, we believe that feedback from industry professionals can support a better traceability model design.

6.2 Survey Objectives

The objective of this survey is to collect data related to traceability of heterogeneous artifacts in Requirements Engineering, Systems Engineering, and Model Driven Engineering. According to the classifications of surveys by Wohlin and colleagues [119], this survey lends itself to the exploratory type since we need to have an insight about important features for a generic traceability model. To achieve our objective, the survey targets professionals from industry who apply traceability in their projects throughout systems development. The feedback from the survey can be used to understand the capabilities of existing traceability tools and enrich the design of a generic traceability model, it can be used for supporting the design of our traceability model. Conducting an online survey is a good option since the survey can reach a wide range of participants from various domains and locations. The survey aims to find answers about the following research questions.

Q1 . What are the possible types of traceable artifacts that should be captured?

Q2 . How can we improve the identification of trace links between traceable artifacts?

Q3 . What are the types of traceability tools used in industry?

Q4 . What are the capabilities and shortcoming of existing traceability tools?

Q5 . What features a generic traceability model should have?

6.3 Targeted Audience and Population

The selection criteria for the survey audience is constrained by two factors. First, the survey is very technical, which implies the selected audience should be professionals who usually work in systems development such as analysis, design, quality assurance, or similar domains. Second, they should have some experience of applying traceability on projects at their workplace. Therefore, the size of the audience is large (i.e., software engineers with different expertise), however, the population (i.e., software engineers who apply traceability) that fits the second criteria is comparatively low. We specified an explicit constraint in the consent

letter which prevents participants who don't practice traceability from continuing the survey. In addition, these two factors limit the total number of participants in our survey.

6.4 Survey Questionnaire Design

The survey questionnaire is aligned with the survey objectives that come from the research questions. We follow the guidelines proposed by Linaker and colleagues [35] for designing a survey questionnaire in order to achieve sound results and conclusions from the survey research. There are several factors that we consider for this matter. For instance, we paid attention to devise questions in order to collect accurate information about features of traceability model, we considered the question (open, close) type to achieve our goals. In addition, we set the

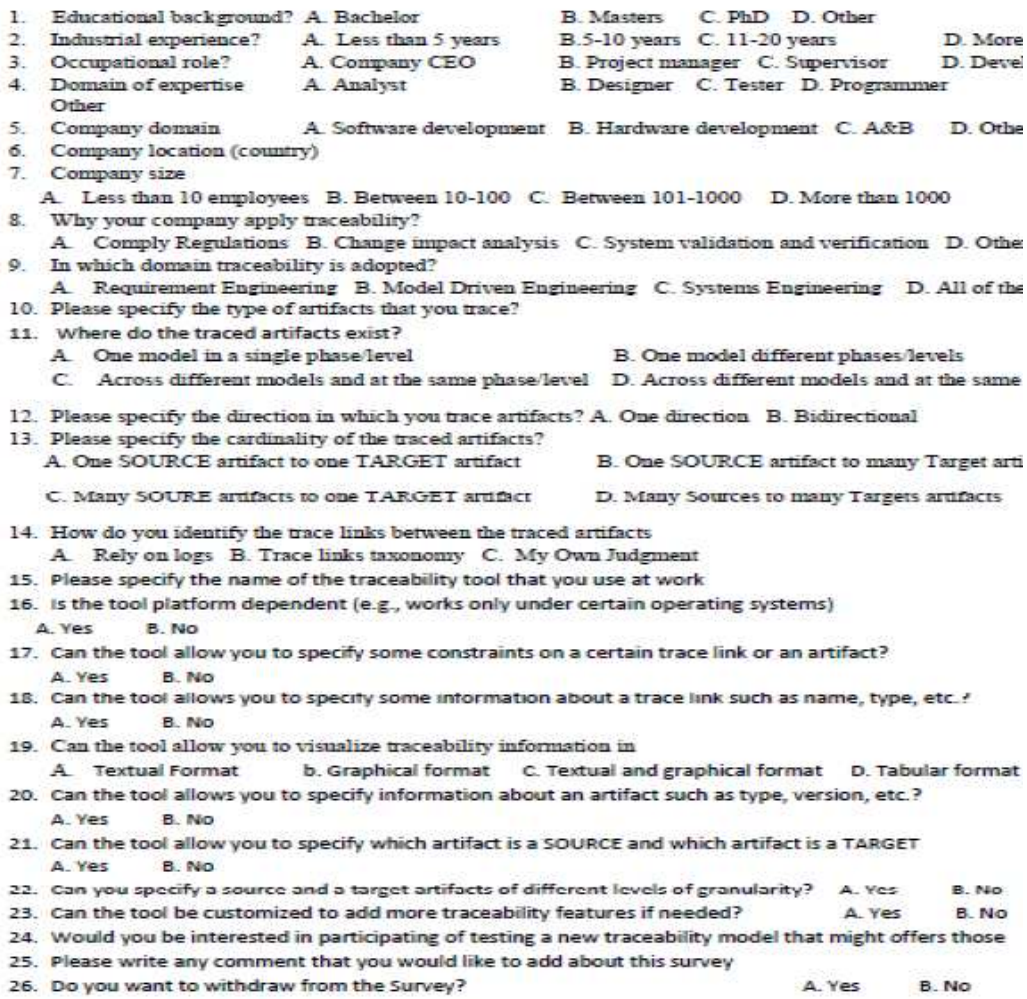
1. Educational background? A. Bachelor B. Masters C. PhD D. Other
2. Industrial experience? A. Less than 5 years B.5-10 years C. 11-20 years D. More
3. Occupational role? A. Company CEO B. Project manager C. Supervisor D. Devel
4. Domain of expertise A. Analyst B. Designer C. Tester D. Programmer Other
5. Company domain A. Software development B. Hardware development C. A&B D. Othe
6. Company location (country)
7. Company size
   A. Less than 10 employees B. Between 10-100 C. Between 101-1000 D. More than 1000
8. Why your company apply traceability?
   A. Comply Regulations B. Change impact analysis C. System validation and verification D. Othe
9. In which domain traceability is adopted?
   A. Requirement Engineering B. Model Driven Engineering C. Systems Engineering D. All of the
10. Please specify the type of artifacts that you trace?
11. Where do the traced artifacts exist?
    A. One model in a single phase/level B. One model different phases/levels
    C. Across different models and at the same phase/level D. Across different models and at the same
12. Please specify the direction in which you trace artifacts? A. One direction B. Bidirectional
13. Please specify the cardinality of the traced artifacts?
    A. One SOURCE artifact to one TARGET artifact B. One SOURCE artifact to many Target arti
    C. Many SOURE artifacts to one TARGET artifact D. Many Sources to many Targets artifacts
14. How do you identify the trace links between the traced artifacts
    A. Rely on logs B. Trace links taxonomy C. My Own Judgment
15. Please specify the name of the traceability tool that you use at work
16. Is the tool platform dependent (e.g., works only under certain operating systems)
    A. Yes B. No
17. Can the tool allow you to specify some constraints on a certain trace link or an artifact?
    A. Yes B. No
18. Can the tool allows you to specify some information about a trace link such as name, type, etc.?
    A. Yes B. No
19. Can the tool allow you to visualize traceability information in
    A. Textual Format b. Graphical format C. Textual and graphical format D. Tabular format
20. Can the tool allows you to specify information about an artifact such as type, version, etc.?
    A. Yes B. No
21. Can the tool allow you to specify which artifact is a SOURCE and which artifact is a TARGET
    A. Yes B. No
22. Can you specify a source and a target artifacts of different levels of granularity? A. Yes B. No
23. Can the tool be customized to add more traceability features if needed? A. Yes B. No
24. Would you be interested in participating of testing a new traceability model that might offers those
25. Please write any comment that you would like to add about this survey
26. Do you want to withdraw from the Survey? A. Yes B. No

Figure 9: Traceability Questionnaire

questions in a certain sequence to allow for smooth transitions between answers. The survey is composed of 26 questions classified into demographic questions and traceability questions. The traceability questions are further classified into three groups which gather information about trace links and their types, artifact types, and traceability tools and their characteristics. The survey was published online on February 11th, 2018 and closed on April 25th, 2018. It was hosted by SurveyMonkey [120]. It was approved by the ethics committee at Carleton University, Canada under certificate number 108423, see Appendix A. The survey link is https://www.surveymonkey.com/r/G52KKLT. The objective of the survey is to achieve the followings:

- Answer our research questions which are drawn from the thesis research questions.
- Support the finding of our systematic literature review about traceability model requirements.
- Determine whether traceability practices in industry are aligned with the theoretical approaches.
- Find a solution that can satisfy the traceability requirements initiated by our industrial partner, which requires capturing the traceability of heterogeneous artifacts.

6.4.1 Demographic Questions

The demographic questions determine the educational background, occupational role, domain of expertise, total industrial experience, and company domain, location, and size, see questions 1-4 in Figure 9. The feedback of the questions contribute indirectly to our research questions since they can provide us information about the expertise of the participants and their companies' domain. It is important to mention that the participants of our survey are not only software engineers, but also professionals who practice traceability in their companies. We believe that the more we know about participants' demographic data, the better assumption we can make about our solution. For instance, the occupational role, domain of expertise, and

industrial experience of the participants are equally important indictors to measure the quality of the survey feedback. On the other hand, education is a good indicator that reflects knowledge but it does not necessarily reflect knowledge in traceability.

6.4.2 Traceability Questions

The feedback that we get from traceability questions contribute to Q2, Q3 and Q5 which reflect information about the domains in which traceability is adopted, what types of artifacts are traced and in which direction. Also, it provides answers about the cardinality of the relations between the traceable artifacts, and how trace links can be identified. The answers also provide insight about the features of the traceability tools used in industry which can enhance the design of a generic traceability model. We have the following types of questions

- **Questions about Artifacts, trace links and their types**: these questions investigate the different types of artifacts, the cardinality of the relationship between them, the phase or model type they belong to, and the direction of traceability as shown in Table 11.
- **Questions about traceability tools**: these questions investigate the traceability tools and their ability in capturing and visualizing different traceability information, see

Table 11: Questions about artifacts and trace links

Artifacts and Trace Links		
Question	**Question number from Figure 9**	**Comment**
Artifacts type	10	Such as source code, test case, diagram, requirement, hardware
Existence of artifact	11	Traceable artifacts are exist within a phase, a model, across different models or phases.
Direction of tracing	12	Unidirectional, bidirectional.
Cardinality between source and target artifacts	13	One to many, many to many, one to one.
Trace links identification	14	Based on a taxonomy, judgment, standards.

Table 12: Questions about traceability tools

Question	Question Number from Figure 9	Comment
Tool ability to specify properties of an artifact or a trace link.	17, 18, 20	Add metadata for a trace link or an artifact, historical data (version), constraints
Tools ability for graphical visualization	19	Visualize the connection between trace links and artifacts
Tools ability to specify source and target artifacts	21, 22	Which artifact is a source and which artifact is target at different levels
Tool extensibility for more features	23	Add non existing features such as constraint on a trace link, characterization of an a artifact

Table 12. In addition, some of these questions examine whether some of these tools can be customized to meet other traceability needs.

- **Miscellaneous questions**: these questions collect data about the tool name, platform, and why the company applies traceability; also, they allow participants to add comments, or whether to withdraw from the survey.

6.5 Publishing the Survey

Prior to publishing, the host website agreed to send this survey to professional through its network in order to get responses. However, the survey was closed two days after publishing due to the difficulty of finding the right participants; only five participants joined. Therefore, we had to handle this situation by posting the survey on the *LinkedIn* Social network. We searched the profiles of different types of developers such as analysts, designers, and testers and sent them personal letters asking them to participate. We sent around 200 letters to potential participants; we were able to collect results from 37 participants.

6.6 Survey Analysis

We analyze the results of the survey according to the categories provided in section 6.4. The results reveal important aspects that provide us with a deep understanding about traceability practices in industry. The feedback is employed for answering our research questions proposed at the beginning of this chapter.

6.6.1 Analysis of Demographic Feedback

The feedback can measure how the answers of the participants are close to the reality. The chart in Figure 10 shows the participants' demographic data. The figure shows that the participants vary in their experience level, domain of expertise, and occupational role. However, all of them experience traceability at their companies. The number of participants for each individual answer in these categories shows diversity in the educational level, occupational role, domain of expertise, and industrial experience. We believe this diversity can provide us with rich information that can assist our model design. Furthermore, we try to understand the reasons for adopting traceability in these companies, the feedback shows that ten participants are coming from companies that practice traceability to comply with regulations (i.e., mandated), which implicitly implies traceability is used for change impact analysis, validation and verifications, and other purposes. Finally, the data is collected from respondents from various companies in nine countries, not shown in the chart (USA, Canada,

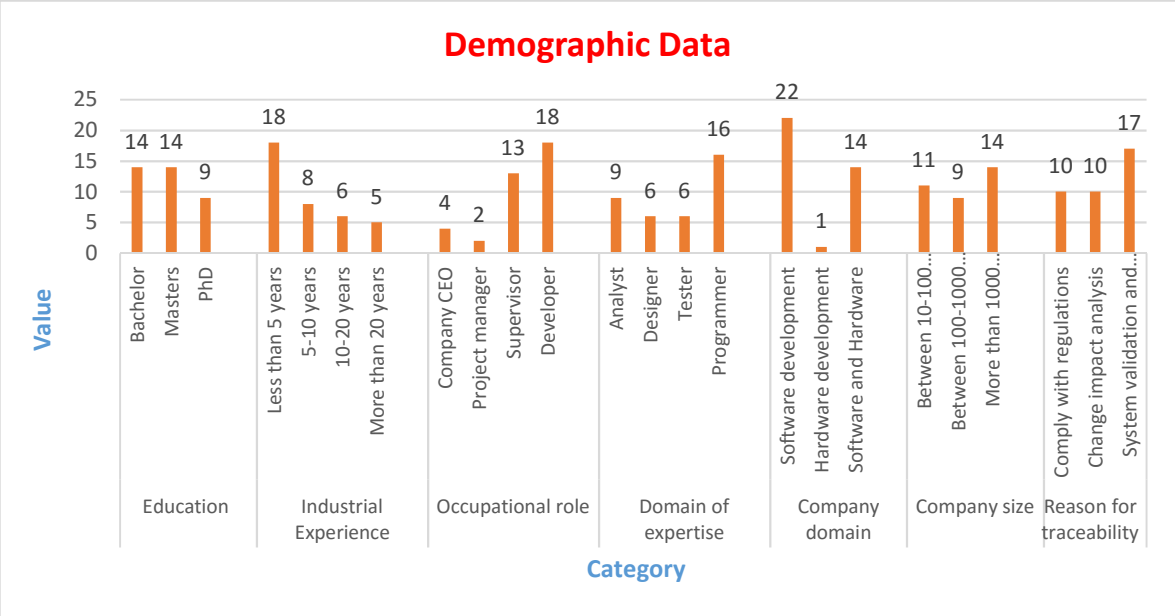

Figure 10: Feedback from demographic questions

Saudi Arabia, Italy, Jordan, the Bahamas, UAE, China, and UK), 14 of the participants are coming from companies which, each, employs more than 1000 persons. The feedback from the

demographic questions provides us with confidence for answering Q1, Q2, Q3, Q4, and Q5 collectively. It provides us with an indication that, given the participants' background and their domain of expertise, they have the right background and expertise to adequately answer the survey, to provide accurate and useful answers.

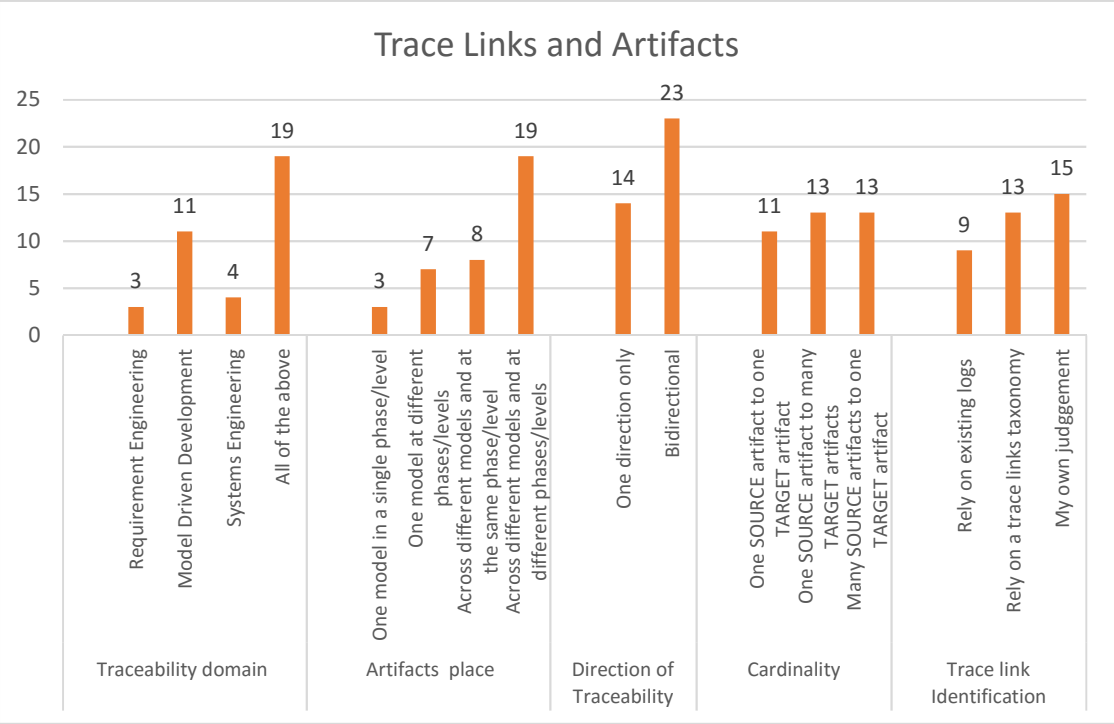

Figure 11: Feedback of trace links and artifacts questions

6.6.2 Analysis of Artifacts and Trace Links Feedback

As depicted in Figure 11, the participants practice traceability in Systems Engineering and Software Engineering. One interesting result is that that around 40% of the participants (15 person) use their own judgment for identifying trace links between artifacts. We argue this can result in inconsistencies in the identification of trace links since no reference exists. Regarding the traceable artifacts, the results show different types of artifacts are used, not shown in the figure. The artifacts are related to data flow, design elements, verification and analysis, Software Engineering requirements, and progress of project elements, survey products, mobile application components, employee activities, network elements, customer requirements, functional and non-functional, test cases, business requirements, high level design documents,

detailed design documents, code, test strategies, meeting minutes, emails, shared discussions, feature to develop, performance elements, use cases, bugs, documentation, big data platform performance, services (service monitoring), personal identifiable information, code paths that require extracted information for debugging, system design, system hardware, system software, and deliverable documentation, architecture components. These artifacts can be further classified into categories that facilitate referencing to them. Also, the figure shows that some traceable artifacts can exist in one model, one phase/level, while other artifacts can exist across different models within the same phase or at different phases/levels. For instance, test cases and test plans can exist in one model but at different phases, test plans belong to low level design in the design phase and test cases belong implementation phase. With respect to the traceability direction, the feedback indicates that some artifacts are traced in one direction, others can be traced in both directions; this is based on the requirement of the application under development. Also, regarding the cardinality, the results show different ways for tracing source and target artifacts such as one source artifact to one target artifact and one source artifact to many target artifacts. In conclusion, the feedback about the use of trace links and artifacts reflects how traceability is practiced in industry. It provides us with indication to the type of artifacts and their cardinalities which gives answers for Q1 and Q2.

6.6.3 Analysis of Traceability Tools Feedback

The feedback about traceability tools is shown in Figure 12. The result shows different traceability tools for different purposes. The participants reports the following tools, not shown in Figure 12: IBM Rational DOORS (3 companies), Traceability Matrix (2 companies), ORCANOS [121] (2 companies), In-house tools (3 companies), JIRA [122] (2 companies), Sparx Enterprise architect (2 companies) [123], IBM Rational RequisitePro (1 company) [124], ExcelTFS (Traceability Matrix for Agile, 1 company), Rally (traceability Matrix for Agile, 1 company), Subversion [125] (1 company), and GitHub [126] (1 company). We classify the tools

into four categories for better understanding. First, some companies use dedicated traceability tools (13.5%) for tracing requirements and managing changes across the development lifecycle. For instance, The IBM Rational Dynamic Object Oriented Requirements System (DOORS) [127] is used for this purpose. Second, some companies are relying on version control websites (27.1%) such as Subversion or GitHub for managing traceability. Third, some companies are using simple applications (16%) like Excel to create traceability matrix for managing traceability. Fourth, some companies are using their own in-house tools (19.2%) to manage traceability during the development of their applications. Figure 12 indicates some interesting results about the properties of the traceability tools. A high percentage of the participants indicates that their tools can be customized to meet certain requirements, specify constraints on an artifact, specify metadata about trace links or an artifact, and specify source and target artifacts. However, these characteristics are not available in one single tool. We studied some of these tools in detail to understand their features, the following tools have been chosen from the abovementioned categories:

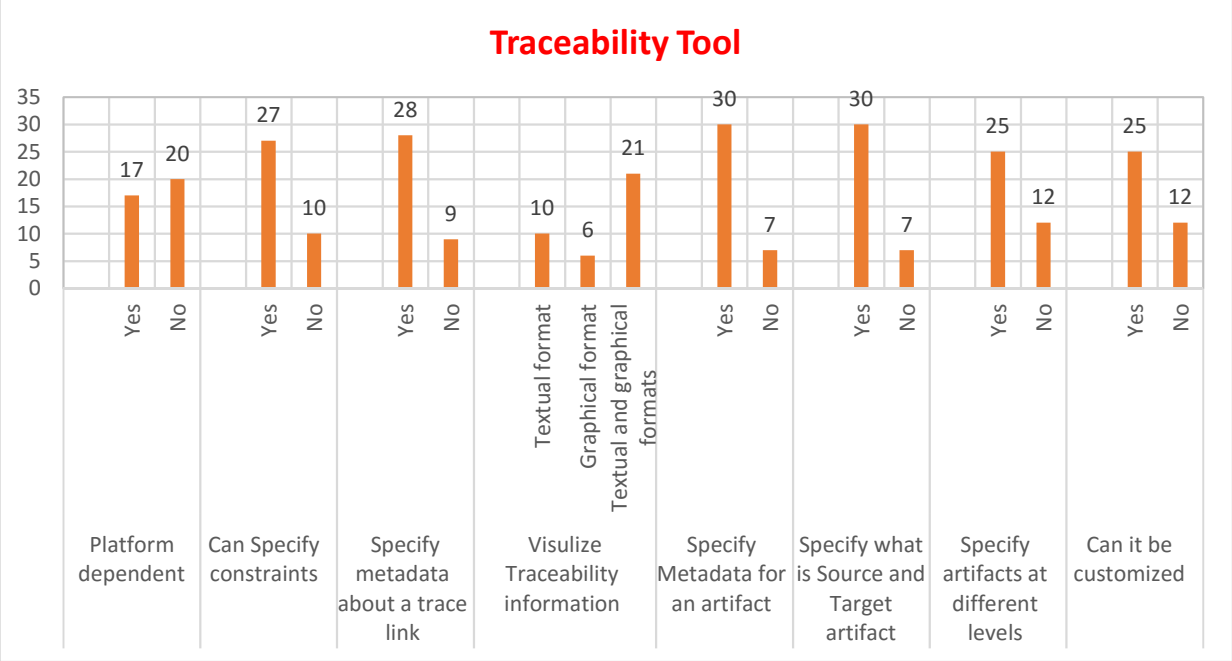

Figure 12: Features of traceability tools

- The IBM DOORS tool is very popular in managing requirements, we tested a web-based trial version of the IBM Rational DOORS Next Generation and noticed it has only a limited set of trace links that a user can choose from. The set contains trace links that can be used to trace requirements in Requirements Engineering. The tool cannot accommodate other types of trace links, for instance, there are very specific trace links related to Systems Engineering and Model Driven Development artifacts that are not available for the user. Also, some trace link types that support traceability at different levels of granularity such as horizontal or vertical traceability are not provided in DOORS. However, the tool allows the user to specify some constrains on an artifact or a trace link using the Domino Extensible Language, the tool has excellent features for displaying traceability information in column or graphical views. The tool provides support for version control, it can attach metadata for an artifact that shows the initiator of the artifact, the creation date, and the modified date. The trace links are also shown in forward and backward directions. The navigation between related artifacts is done through hyperlinks that allow the user to navigate between artifacts.
- Excel and ORCANOS are simple traceability tools that allow the user to link requirements by constructing a traceability matrix. For example, ORCANOS allows the user to link requirements and display them in a column format. It allows also the generation of traceability matrix reports that can be used to perform change impact analysis and generate inconsistency alerts. Companies who adopt an agile development process use these tools because they are simple to use.
- Apache Subversion and GitHub are open source version control tools in whose artifacts are linked implicitly through a set of command issued by a user such as commit. These applications allow the collaborations among many developers that do not exist in one location. Users can submit their files via these applications, the files are stored in

separate directories in which each file is tagged by some information. The information represents the name of the file, its owner, a timestamp indicating the date and time of file creation and modification, and other information about all people involved in the change. Version control applications are beneficial for other reasons besides tracking; for instance, rollback allows users to have access to the previous version of a certain file(s).

- In-house traceability tools: the names of these tools are not disclosed by the participants, however, we understand the specifications of these tools from the individual responses of the participants. The tools are platform specific, the user can specify source and target artifacts. Some tools allow the user to specify constraints, others cannot. Moreover, the traceability information in some tools can be visualized in textual form, other tools can show the information in textual and graphical forms.

Most of the abovementioned tools are domain specific, they cannot accommodate the traceability of heterogeneous artifacts from different domains. For instance, the IBM DOORS is a well-known traceability tool; it is good for Requirements Engineering, however, it does not support some trace links for Systems Engineering artifacts. Also, the In-house traceability tools are designed to serve the traceability of certain artifacts based on specific needs. Finally, the version control tools are not meant for traceability, however, they can be used to record some changes about artifacts over a period of time.

In conclusion, the feedback about traceability tools provides us with answers for Q3 and Q4 by understanding existing industrial traceability tools, their characteristics, and their shortcomings.

6.7 Conclusion

In conclusion, the survey provides interesting results about traceability practices and traceability tools. The feedback of the survey varies among participants. There is no one

company that applies all traceability practices, which is expected since companies are practicing traceability for different reasons. This also applies to the traceability tools, since they are used to capture traceability information for different purposes. The results of our analysis about traceability tools and artifacts are similar to the results that we obtained from our systematic literature. The results reveal the following concerns that we need to consider during the design of our traceability model:

1- The need for trace links that supports all development domains.
2- The need for recording version information about artifacts and trace links.
3- The need for a customization feature that allows the user to specify different types of trace links, artifacts, or reporting and visualization options.
4- The need for a tool that is not domain specific and which can capture traceability information for Requirements Engineering, Model Driven Engineering, and Systems Engineering.

7 Generic Traceability Model Design

This chapter introduces the core of our work in which we propose requirements for a generic traceability model, and traceability model design that can facilitate capturing traceability information of heterogeneous systems. A context of use is discussed in section 7.1. The requirements of a generic traceability model are stated in section 7.2. Section 7.3 describes our traceability model, and section 7.4 provides an initial discussion of the validation of the traceability model.

7.1 Context

Since we are interested in systems that are realized through software and hardware solutions, we extend the Model Driven Engineering traceability definition and consider traceability as any relationship that exists between artifacts involved in the Systems Engineering life cycle. Based on this definition, we are proposing a new traceability model that can address the issues we stated in the introduction chapter, primarily the fact that we typically need to link artifacts that come from widely different sources, including but not restricted to: structured and un-structured natural language, source code, domain-specific modeling languages either formally described with an abstract syntax and a concrete syntax or not, and Ecore based languages. We also want to accommodate any taxonomy of traceability links engineers may want to use.

As a result, the model should not make any assumption about the types of artifacts that can be linked (heterogeneous artifacts), or how they should be classified (various possible taxonomies), and therefore it should not make any assumption about the semantics those links could have according to engineers' needs. In other words, the model should facilitate extensibility to allow new types of links, with possibly new semantics, to be created between new types of model artifacts.

At the same time, our objective is to find a solution that can offer such a level of extensibility without requiring the model to be extended (i.e., only the instance model can be modified). Indeed, extending the model is always a possibility but then the tool support needs to be extended too, and this has a cost; if links have already been created, then the instance of the (old) model needs to be transformed into an instance of the new model, which has a cost too and we believe this is not practical in the general case.

7.2 Requirements for a Generic Traceability Model

We need a traceability model that must be oblivious of the heterogeneity of the models whose artifacts are traced. Based on our needs, we put forward a set of requirements that shall be satisfied by a generic traceability model.

Req 1. The model implementation shall be independent of any tool, language, or framework that used to create artifacts.

Req 2. The model shall allow modeling traceability between artifacts of similar or different types (i.e., homogeneous and heterogeneous artifacts).

Req 3. The model shall allow vertical traceability between artifacts across different models, phases, or levels of abstraction.

Req 4. The model shall allow horizontal traceability between artifacts within the same model, phase, or level of abstraction.

Req 5. The model shall allow forward and backward traceability between a source and a target artifact.

Req 6. The model shall specify source and target artifacts of a traceability link.

Req 7. The model shall allow modeling traceability between source and target artifacts of different cardinalities.

Req 8. The model shall allow adding constraints to a trace, trace link, or an artifact using a constraint language.

Req 9. The model shall allow applying more than one or more characterization to an artifact or trace link.

Req 10. The model shall allow modeling traceability between artifacts at different levels of granularity (i.e., different conceptual or decomposition levels).

Req 11. The model shall allow capturing traceability information between models during model transformation.

Req 12. The model shall be flexible such that it allows accommodating a new trace link, artifact, constraint, and characterization without changing the model itself.

Req 13. The model shall allow the user to capture information related to version contro.

The abovementioned requirements assume that a traceability model should not rely on the fact that artifacts are instances of an MOF-based language. We typically need to link artifacts that come from widely different sources. We are also looking for a model that can accommodate any taxonomy of traceability links engineers may want to use. In other words, it should allow different, possibly new, ways of characterizing trace data. As a result, the model should not make any assumption about the types of artifacts that can be linked (heterogeneous artifacts), or how they should be classified (various possible taxonomies), and therefore it should not make any assumption about the semantics those links could have according to engineers' needs. The model should not change when new artifacts, coming from newly created modeling notations need to be traced, or when new classification taxonomies need to be used. One can view these two constraints as having to identify a traceability model that can be extensible, to accommodate new models, new artifacts, different, possibly new ways of characterizing them, without having to change the model itself (only its instance would change).

We assume users to have three roles, similar to other technologies such as networking, as they require different kinds of expertise and they expect different services from such a generic traceability model: A super-user would be in charge of defining the legal taxonomies,

characterizations, and constraints. The constraints that we refer to are those that would be deemed necessary, given a specific use of taxonomies and characterizations in a specific context, to ensure instantiations of the modelling solution are valid. This would require a deep understanding of the traceability model and those taxonomies/characterizations/constraints as dictated by the context (i.e., domain, organization, team, project); An engineer would be in charge of tooling, for example, to feed trace information from the various tools that are used to create heterogeneous artifacts to be traced, enforce taxonomies defined by the super-user. This would require a deep understanding of the technology being used to provide tool support for our model and its use (e.g., an MOF-based representation of our model in an Eclipse plug-in of some modeling platform); a domain expert who would use the technology and tool support to create traceability information between heterogeneous artifacts and enquiry about this information.

7.3 Traceability Model Description

The design of our model in Figure 13 is based on our need for a solution that satisfies requirements specified in section 7.2. Moreover, our design is inspired by the work of many researchers who designed their own traceability metamodels [48]–[51], [81], [82], [88]–[90], as well as authors who discussed the importance of incorporating semantics to the traceability links by adding constraints [9], [48], [80], [89], [90] since we want to benefit from all those related works. However, as illustrated previously, the solutions to traceability modeling we have found in the literature have one or more of the following drawbacks: the technique is tailored to a specific domain such as model transformation [81] or BPMN [49]; the technique is tailored to a specific kind of model from which artifacts are traced, such as MOF-based models [89]; the technique does not allow the specification of additional types of traceability links; the technique does not facilitate extensions to new types of traceability links and when this is possible, extensions may lead to a large number of metaclasses (e.g., [48]).

Consequently, we propose a generic traceability model (Figure 13) that consists of the followings. The *TraceabilityRoot* class acts as the root of the traceability model. Its purpose is to hold the traceability information about the artifacts under study. The set of artifacts under study is typically the responsibility of the domain expert. However, one can reasonably expect these will include artifacts created during a specific software/system development. A *TraceabilityRoot* contains instances of *TraceElement* (abstract class), which are in fact instances of *TraceLink*, *Trace*, or *Artifact*. The *TraceElement* class is associated with class *Characterization* to allow the characterization of any given trace element according to any taxonomy (or taxonomies) the user wishes to employ. A trace element has an attribute *uri* which references an instance of an *Artifact, TraceLink, or a Trace* instance, and it can be considered as a unique ID for a *Trace* instance. We assume that trace link types and artifact types are stored

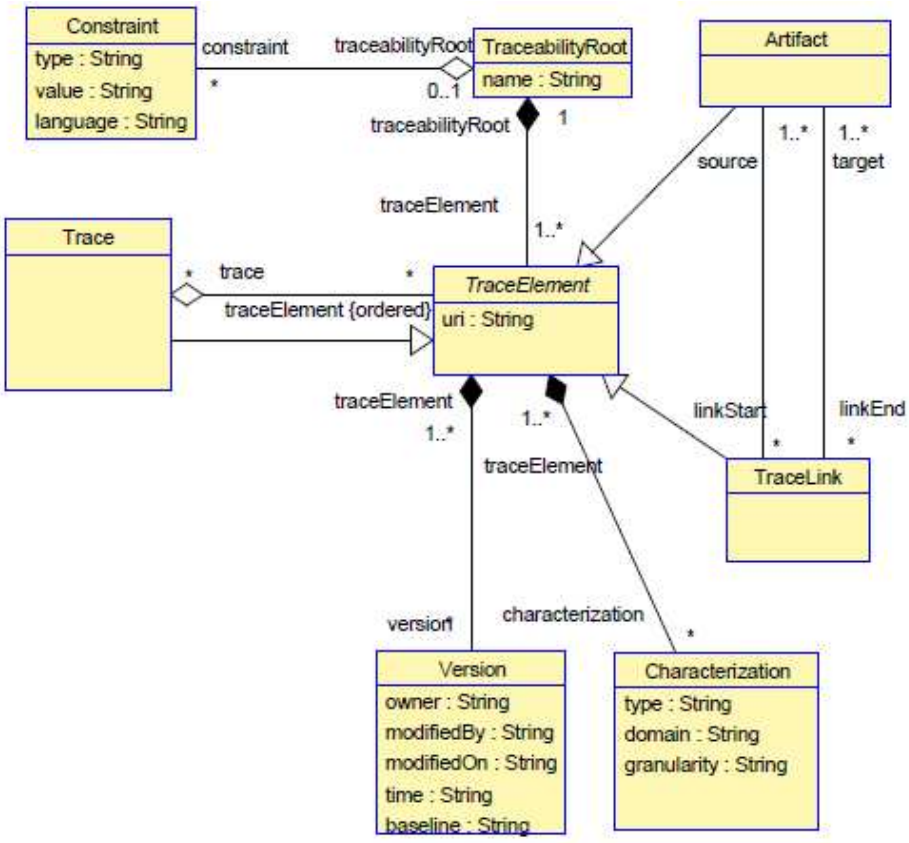

Figure 13: Generic traceability model

in a taxonomy that allows easy referencing of these elements. These ideas are borrowed, merged, and integrated from previous works [9], [48], [89], [90], though not all come from a single of these works (we integrate previous solutions).

A *Trace* represents a sequence of chained trace elements, generated during a sequence of model transformations (e.g., as modeled by Falleri and colleagues [81]), or simply to represent the transitive nature of some traceability links where the target artifact of a link becomes the source artifact of another link. We opted for a composite pattern to provide flexibility to the user of the model in handling a *Trace* as a series of either *TraceLink* or *Artifact* instances. We decided to further specify in our model design, the chained elements are either all *Artifact* or all *TraceLink* instances, and forbid any mix of *Artifact* and *TraceLink* instances (which is allowed by the model class diagram) since we do not find it useful to have such a mix; on the contrary we felt a mix would hinder reasoning about the *Artifact* and *TraceLink* instances that are involved in a *Trace*. In addition to constraining the types being in the sequence, we specify that in case the *Trace* instance is a sequence of *TraceLink* instances, the target *Artifact* of the i^{th} *TraceLink* instance is the source *Artifact* of the $(i+1)^{th}$ *TraceLink* instance. Similarly, in case the *Trace* instance is a sequence of *Artifact* instances, the i^{th} and $(i+1)^{th}$ *Artifact* instances are the source and target of a *TraceLink* instance. The OCL constraints of the model are provided in the traceability model specification in Appendix B.

The *version* class instance can hold version control information about instances of *TraceElement.* This class reflects the need for cases in which many members involved in the development process. The creation of the *Version* class satisfies Req 13.

The *Constraint* class is used to impose any number of constraints to a *TraceabilityRoot* elements. System artifacts such as requirements are changing continuously, consequently, the links between them are changing, and this might lead to inconsistent or invalid links. Therefore, constraints are necessary to govern the relationship between system artifacts before they are

used. We have chosen to create an association between the *Constraint* and *TraceabilityRoot* classes in order to allow the user to impose any number of constraints at the model level, the creation of the *Constraint* class is to satisfy Req 8.

A *TraceLink* instance represents a traceability link between artifact(s): one or many source artifacts and one or many target artifacts for a trace link. Note that some previous works limit those multiplicities to be strictly 1, though we believe 1..* brings more flexibility, thanks to (inherited) associations to *Characterization* and *Constraint*, to capture information about the relationship between source and target artifacts.

The purpose of *Characterization* is to characterize a *TraceElement* according to zero or several taxonomies. Indeed, we felt that different taxonomies characterizing traceability links according to various dimensions could be of interest in practice: e.g., the notions of horizontal and vertical traceability, the categories and traceability types [9], categories of traceability types specific to Model Driven Engineering development [48]. Moreover, the Characterization class works as the stereotype in SysML which is used to classify requirements in Systems Engineering into further categories such as operational, functional, interface, performance, physical, storage, activation/deactivation, design constraints and other specialized requirements such as reliability and maintainability, or to represent a high-level stakeholder need. Notice that, contrary to other solutions (recall Table 10) we decided to exclude any explicit specification of specific taxonomies from our solution. The advantage is that we are not tied to a specific set of taxonomies, that we can use several taxonomies together, and that our solution is therefore not specific to either taxonomy and can evolve. In short, the domain expert, with the help of the super-user and the engineer, can decide which taxonomies are important to their context. The creation of the *Characterization* class satisfies Req 9.

The drawback is that our solution may appear too generic or permissive: i.e., one can provide meaningless characterization. However, we assume that the users of our model are expert users

and they will not provide any meaningless characterization or that domain experts will constrain solutions (see the class Constraint) with adequate restrictions.

The class *Artifact* represents any traceable unit of data such as a UML class diagram, a message in a UML sequence diagram, a block in a block diagram, a natural language requirement, a PDF standard document. One very important difference with many other attempts at modeling traceability links regarding the artifact specification is that our artifact specification is not tied to any language with which the artifact being linked is modeled: e.g., it is not tied to Ecore languages as in TML [89].

Constraint instances can be used to enforce some structural integrity of the model instance, i.e., of the traceability information [48], [89]. For instance, one can specify that only certain types of artifacts can be linked together, thereby forbidding links between other kinds of artifacts; one can specify that only specific characterizations can be linked to a *TraceLink* (e.g., a trace link cannot be at the same time horizontal and vertical). Since such constraints are domain, organization or project specific, they cannot be all specified in the model and must be specified by the super-user and the engineer under the supervision of the domain expert. To that end, the *Constraint* class provides attribute *type* to identify the constraint, attribute *value* to specify the constraint itself, and attribute *language* to specify the language in which the constraint (i.e., *value*) is written to then allow an algorithm to automatically trigger the right constraints evaluation engine: e.g., the *type* could equal "OCL" and the *value* could be an OCL expression.

We argue that we are proposing a traceability model that is more generic than previously published solutions since it can accommodate, by design, tracing artifacts that come from widely varying model types (see Req 2, Req 3, Req 10), tracing new kinds of artifacts (because *Artifact* is not tied to any other model), from possibly new kinds of models, and that those artifacts can be characterized in any possible way the super-user sees fit (see Req 9 and Req

12). Extensibility without changing the model is ensured by the abstract notion of *Characterization*, which can be tailored (i.e., instantiated) to specific needs and can be constrained thanks to class *Constraint* to enforce the structural integrity of the model instance (see Req 8 and Req 12). It is interesting to note that the level of complexity of our solution, for instance in terms of the number of classes, associations, and attributes, is similar to that of other solutions (e.g., [89], [90]), though it is more generic and addresses our needs, contrary to those other solutions.

Our Traceability model design employs the specifications of the Meta Object Facility (MOF) and UML. The MOF architecture specifies four levels: M3, M2, M1, and M0, see Figure 14**,** left side. According to MOF specifications [128], every model element on every level is strictly in correspondence with a model element in the layer above it. For instance, a UML class in the M1-level must conform to a UML Metaclass in the M2-level. However, since there is no other layer, M3 conforms to itself. Our traceability model is at level M1 which conforms to the UML metamodel that exists at level M2, see Figure 14, right side.

We argue that there is no need to specify our model at the M2-level for many reasons. First, if our model were to exist at the M2-level (i.e., Metamodel), then the M1-level model will be a domain specific language. This means that M1 classes will be specific types of the M2 traceability Metaclasses and will be a smaller set. Moreover, the M0 instances of the M1 will

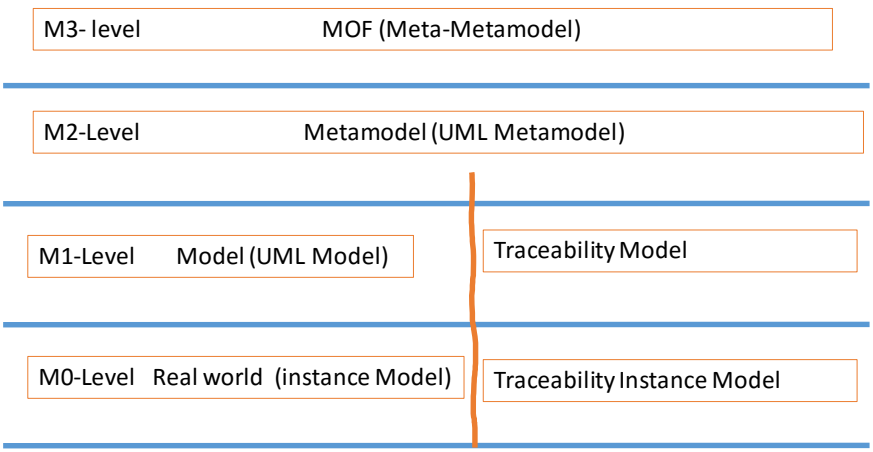

Figure 14: The relation between traceability model and MOF architectures

be only the instances that are related to M1 class set. This violates our requirement for an extensible model. In order to clarify this issue in more details let us assume the following scenario: Assume that the M2 Level has all the traceability classes of Figure 13, and assume that we specify only the *Artifac*t and the *TraceLink* classes based on M2-level. An object model (M0) from the M1 will have only instances of the *Artifact* and *TraceLink* classes. Suppose we need to characterize an *Artifact* or a *TraceLink instance*, we cannot do that since there is no corresponding *Characterization* class at M1 Level. Somebody might argue that why not specifying in M1 all the classes that we need to instantiate in the future, this means we are specifying in M1 all the Metaclasses that are already exist in M2. This is undesirable since we are duplicating M2 classes in M1.

7.4 Validation of Traceability Model by Construction

Since our model is generic, its validation is not trivial because we cannot define a threshold for the number of test cases that would be required to prove its generality. Therefore, we envisioned a two-fold validation of our model: validation by construction as discussed below, and validation through experimentations which we will discuss in chapter 9.

With respect to our validation by construction, we show in Table 13 the rationale behind the creation of every class, attribute, constraint, or association such that the model can capture traceability information based on the context we set in section 7.1. In addition, the trace links taxonomy that we are proposing in chapter 8 allows multilevel trace link classifications that can be incorporated into our traceability model. Thus, our traceability model can accommodate diverse types of models that have diverse types of artifacts and trace links.

Table 13: Traceability Model design rationale

Class/Association/Attribute	Req#	Design rationale
TraceabilityRoot class		To hold the traceability information (i.e., traces, artifacts, trace links) of systems under study.
TraceLink class		To provide a link between a source and a target artifact.
Trace class		To hold a sequence of traced elements (i.e., artifacts or trace links).
Constraint class	Req 8	To provide flexibility and extensibility to the traceability model by applying any number of constraints on an artifact, trace link, or trace through the *TraceElemen*t class. Instances whose constraints are violated can be dealt with automatically.
Characterization class	Req 9	To provide flexibility and extensibility to the traceability model by applying any number of characterizations to an artifact, trace link, or trace through the *TraceElement* class.
Version Class	Req 13	To provide flexibility for capturing historical data about artifacts, trace links, or transformations.
TraceElement class		A generalization of *Artifact, TraceLink,* and *Trace* classes. Its purpose is to simplify the traceability model design by allowing a characterization of any instance of an *Artifact*, *TraceLink,* or *Trace*, and imposing constraints on these classes.
Associations between *Artifact* and *TraceLink* classes	Req 4, Req 5, Req 7, Req 6,	To allow one-to-one, one-to-many, and many-to-many relationship between source and target artifacts. In addition, it provides a direction for the association (i.e., which artifact is a source and which artifact is a target).
Association between *Trace* and *TraceElement* classes	Req 11	To constrain the generated sequence of trace elements in case of model transformation to be an ordered set of either artifacts or trace links, and forbid any mix of them.
Association between *TraceabilityRoot* and *TraceElement* classes		To allow the traceability model to hold any number of artifacts, trace links, constraints, traces, or characterizations.
Attributes of type *String*.		The rationale for using the String type for most of the class attributes is to move the complexity to the tools to ensure the validity of string values. We envisioned this design from the design pattern (boundary, control, entity) for UML classes in which a tool must enforce the use of legal stereotype strings.
Attribute *resourceURI* of type URI		To specify the exact location of a traceable artifact within a model, such as a locating a requirement inside a document. This is envisioned from the design of Anquetil et al. [90]

7.5 Tool Support and Integration

Domain experts value traceability when the system is complete and they try to understand whether all requirements have been met [9]. The design of a traceability tool is usually based on a traceability metamodel. However, systems' users have difficulties integrating the traceability information with the traceability metamodel as a result of different data formats produced from heterogeneous CASE tools. Therefore, it is of great importance to have a traceability framework that processes different data formats into a unified format in order to support traditional traceability management tasks such as querying, visualizing, and analyzing

the data. Automation of capturing traceability information is also vital especially if the system under study is complex and has a large number of diverse artifacts and trace links.

We envision our traceability framework as a collection of integrated tools that can support the capturing, storing, and reasoning about traceability information: see

Figure 15. We consider employing the Open Service for Lifecycle Collaboration (OSLC) in order to support flexibility and usability of our traceability framework. Indeed, OSLC specifications allow tools integration using linked data, and therefore seems to be an adequate technology to rely on for our solution. There are other solutions that proposed linking traceable artifacts with traceability tools. For instance, Rahman and Amyot [129] proposed a solution that integrates the IBM DOORS with models artifacts generated from a traceability domain specific language. A special DOORS library called DXL script is generated from the models artifacts and then exported into the IBM DOORS.

We implemented our trace links taxonomy using the *Fluent Editor* tool which allows for writing the specifications of the taxonomy using English like text. The tool allows for exporting and importing the taxonomy specification in various data formats such as RDF, XML, and OWL. Moreover, we offer publishing the taxonomy online in order to allow traceability domain experts to provide feedback about existing trace links classifications.

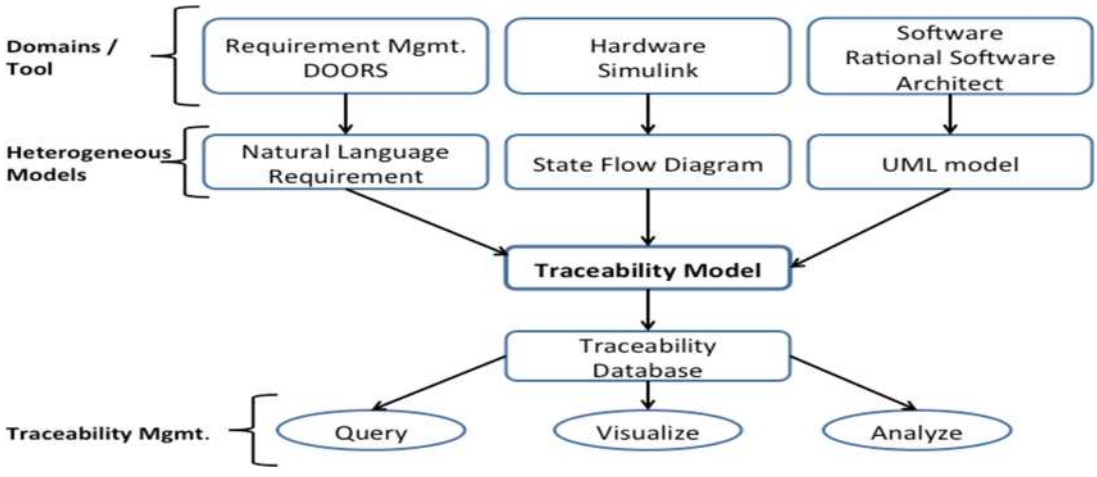

Figure 15: Proposed traceability framework, source [90]

Regarding our traceability model we also used simple techniques; we build our traceability model by writing the specifications for all classes and associations and cardinalities in a simple script. The script file is read by an open source UML based tool (USE) [130] and generates the model classes and associations accordingly. However, the implementation of our model is not tied up to this tool or to a specific operating system; it can be implemented using other tools operated under UNIX or Windows operating systems such as Eclipse. Moreover, we use the USE tool for the instantiation of the object model by writing a plain text script that describes the instances of the required object model(s). Therefore, we avoid the manual instantiation of the object model, which is error prone, and we ensure the integrity of the object model.

8 A New Encompassing Traceability Link Taxonomy

We showed in chapter 4 our literature review about existing classifications of trace links. These classifications do not consider a unified classification that combines all trace links related to Requirements Engineering, Model Driven Engineering, and Systems Engineering domains. This chapter introduces our effort in providing a unified trace links taxonomy that can be utilized in these domains. The chapter discusses four main facets. First, it shows the existing classifications of trace links and their drawbacks. Second, it presents a set of requirements for a trace links taxonomy; these requirements are proposed based on our needs and the drawbacks of existing trace link classifications. Third, it presents a technique to build the taxonomy by employing the Link data and the Resource Description Framework (RDF) [131]. The taxonomy can be configured with traceability models using the Open Service for Lifecycle Collaboration (OSLC) [132]. Finally, it provides validation criteria for validating the taxonomy requirements and validating the solution through an example.

8.1 The Need for a Trace Link Taxonomy

Software traceability provides a means for capturing the relationship between software artifacts at different levels of abstractions and across multiple domains. Software artifacts can be produced during Requirements Engineering, Model Driven Engineering, and Systems Engineering. Establishing relationships between these artifacts requires different types of trace links each having precise semantics. Unfortunately, there is a lack of consensus among software practitioners for defining precise trace links semantics. This is an issue since using different, either overlapping or conflicting semantics for trace links can have an adverse effect on product quality [96]. Our aim is to build a trace link taxonomy that has well-defined semantics and encompasses all possible types of trace links in the Requirements Engineering,

Model Driven Engineering, and Systems Engineering domains. This is important for many reasons. First, in Requirements Engineering, many artifacts are produced during requirements elicitation, analysis, and validation, hence, require different types of trace links with different semantics. Second, in Model Driven Engineering, which permits model transformation, a large number of trace links are required to link artifacts in source and target models, some of which are generated automatically while others require manual generation; relating these artifacts requires well-defined semantics for trace links too, which are slightly different from what one can define in Requirements Engineering or Systems Engineering. Third, in Systems Engineering, the development of a complex system involves the generation of heterogeneous artifacts as a result of using different modeling tools for modeling different aspects of the system, from different disciplines (e.g., electrical, software). Fourth, comprehending the rationale for creating different types of trace links among artifacts at different levels of granularity requires well-defined trace links semantics. Fifth, there are situations that require many types of trace links in the same domain but for different purposes. For instance, when linking two requirements, a requirement derived from another requires a different trace link than a requirement clarified by another. Sixth, the meaning of a trace link can be viewed differently by different stakeholders. For instance, a trace link between a requirement and a design element may be viewed by a designer as a constraint the requirement imposes on the design element, while an end user might view the same link as a design element produced by the requirement [133]. Finally, with the various types of modeling tools across different domains, it is a necessity to have a trace link taxonomy that can be integrated with other APIs. In other words, we need a portable taxonomy that can be integrated easily with other tools.

Based on our systematic review (chapter 3), we identified some research papers that define traceability and traceability relations [9], [17], [19], [20], [23], [44], [96], [106], other papers that classify or identify some types of trace links [18]–[20], [44], [48], [91]–[94], [96]–[102],

[104], [106], [134], [135], and some papers that discuss the need for trace link semantics [37], [48], [105], [136], [137]. Although these papers provide valuable information on traceability definitions and classifications, we couldn't find any paper that suggests a technique for building a trace links taxonomy that combines the trace links from all domains into a unified taxonomy. Most of these studies are confined to defining trace links and their semantics only for a specific problem or domain, i.e., solutions are problem or domain specific. For instance, there is a great deal of effort on classifying traceability links and their usage in Requirements Engineering [20], [96], though classifications only apply to Requirements Engineering.

8.2 Simple, Motivating Example

The heterogeneity of artifacts that are involved in the development of a complex system requires various types of trace links. The variations between the Requirements Engineering, Model Driven Engineering, and Systems Engineering domains require different types of trace links to relate their artifacts. There are situations in which ambiguity exists in capturing traceability information among artifacts as a result of the absence of a reference model that describes the various types of trace links and their exact purposes. We discuss the traceability rules that relate artifacts from the *i** metamodel, which capture *early-phase requirements*, to the *UML-Class* metamodel, which captures *late-phase requirements* [82]. The *i** metamodel focuses on the strategic *Actor* relationships. It consists of Strategic Dependency (SD) and Strategic Rationale (SR) models. The SD model represents system's *Actors* and the relationships (dependency) among them. It captures the rationale (why) of *Actor's* activities. The rationale (i.e., the "why") obtained from the *i*-SD* model can be used to construct the *late-phase* requirements (i.e., requirement specifications) using the *UML-Class* model.

The *i** metamodel contains the meta-classes *Actor, Resource, SoftGoal, HardGoal, and Task*. The *UML-Class* metamodel has the meta-classes *Class, Attribute, and Operation*. *Actor* and *Resource* in the *i** metamodel are mapped through traceability links to the *Class* metaclass in

the *UML* metamodel. Also, the *SoftGoal* and *HardGoal* in the *i** metamodel are mapped through traceability links to the *Attribute* in the *UML* metamodel. Finally, a *Task* in the *i** metamodel is mapped through traceability links to the *Operation* in the *UML* metamodel.

In this example, determining the relationship between any two Model Driven Engineering artifacts mandates the use of Model Driven Engineering trace links in which each has a specific semantic. However, other semantics may apply based on the domains of the artifacts, the purpose of the relationship, the user's need, or the artifacts level of granularity. As a result, a user might not be able to choose the correct trace link without a taxonomy that accounts for all these situations. For instance, the relationship between the *Actor* and the *Class* in our example can have different semantics based on the level of granularity or user's need. At a high level of abstraction, if the user needs to specify that the two related artifacts are Model Driven Engineering artifacts, then the *MDE* type should be selected to show both artifacts are of an MDE type. At a low level of abstraction, if the user needs to specify a relationship can be established between two instances of the *Actor* and *Class* only if they have the same name, then the *Consistent-with* trace link must be selected, which implies the names of the instances must be identical. In this example, two valid trace links with different semantics can be chosen but for different purposes. Therefore, implementing these trace links in one taxonomy not only facilitates their selection, but also allows the user to build a complex relationship between related trace links which implies producing complex semantics from simple ones.

Moreover, we can discuss similar scenarios when relating Requirements Engineering artifacts to Model Driven Engineering artifacts. For instance, relating a requirement to a Model Driven Engineering model implies different trace links selections with various semantics. Therefore, the intended trace link taxonomy is not only describing the trace links and their usage but also it is a knowledge base where we can build complex rules about trace links and reason about them.

Table 14: Trace links classifications in Requirements Engineering, Model Driven Engineering, and Systems Engineering

Ref#	Requirement Engineering Links Types										
[93]	Product-Related				Process-Related Links						
	Satisfaction		Dependency		Evolution		Rationale				
	Satisfy, Interface-With, Defined-By, Created-By, Allocated-To, Derived.		Temporal, Depend-On, Trace-To Contain, Used-by, Performed-by		Evolve-To, Modify, Define, Elaborate, Derive		Generated-By, Resolve, Supported-By, Affect				
Other RE References (using the same name or a different name)											
[20]			Dependency		Evolution Replace, Based-on, Formalize, Elaborate		Refine\ Generalizatio	Satisfaction	Overlap	Conflict Based-on, Affect, Resolve, Generate	Rationale / Contribution
[19]											X
[98]			Requires-feature-in						X		
[99]			X								X
[104]			X								
[100]			X								
[101]			X								
[102]			Causal-dependency Conformance		Non-causal conformance						
[25]								Satisfy Derive Refine			
[92]			Developmental		Temporal		Contain		Adopt		
[93]			Correspondence								
[105]											X
[37]								Satisfy Establish Contribute			
[103]										Inconsistency	
[82]									X		
Model Driven Engineering Classifications											
[97]	Implicit	Explicit					Model-To-non-model artifact				
	Update, Query, Creation, Compositio, Deletion, Model-to-Model, Model-to-Text	Model-To-model				Dynamic	Satisfy Allocated-To, Explain, Perform, Support				
		Static				Call, Notify, Generate, Synch -with					
		Consistent-with	Dependency								
			Refine, Import, Export, Usage								
Systems Engineering Classifications											
[106]	Temporal	Directional									
		Vertical		Horizontal							
		Micro	Macro	Micro	Macro						
		Inter Intra	Inter Intra	Inter Intra	Inter Intra						

8.3 The Drawbacks of Existing Classifications

As evidenced by the discussions of trace link classifications that we have discussed in chapter 4 and show in Table 14, existing classifications of trace links have the following drawbacks:

- Each classification is either problem specific or domain specific. For instance, the classifications of Ramesh [96] and Spanoudakis and Zisman [20] are related to Requirements Engineering classification, the classification of Paige and colleagues [97] is a Model Driven Engineering classification.
- Classifications are inconsistent with respect to their interpretations of link semantics. They often refer to the same semantics with different names. We conjecture this is a side effect of the first drawback. For instance, Spanoudakis and Zisman [20] classified is-a as a generalization link (second reference in Table 14), while Paige and colleagues [97] classify it as a dependency link (one but last reference in Table 14). Spanoudakis and Zisman use dependency for another purpose.
- Classifications are redundant, which we conjecture is also a side effect of the first drawback. For instance, in Table 14 the *rationale* trace link appears in Requirements Engineering classification [96] as part of product-related trace links and elsewhere [20] as a separate type.
- Classifications do not integrate all usages of a trace link across different domains. In other words, the purpose of a certain trace link does not necessarily appear in the classifications of all domains, though there is no reason to believe this should be the case. For instance, we believe it makes sense to use the *rationale* or the *evolution* trace links in the Requirements Engineering, Model Driven Engineering, and Systems Engineering domains.

- There is no tool support for these classifications that would allow a user to navigate or to query about certain links across different domains. Some classifications such as Ramesh's work [96] shows a tool support for the Requirements Engineering classification.

8.4 The Taxonomy Requirements

The abovementioned limitations encouraged us to think about a new method for building a taxonomy that shows all trace links types and the possible relationships between them. In light of the previous discussion, we envisioned the following requirements for the new taxonomy:

TRQ 1: the taxonomy shall provide semantic specifications for trace links that relate various artifacts types in different domains and at different levels of granularity.

TRQ 2: the taxonomy shall catch the need for different types of users (e.g., analysts, designers, programmers, testers).

TRQ 3: the taxonomy shall allow the specification of a trace link only once and relate it to different domains without duplications.

TRQ 4: the taxonomy shall be flexible to allow users to add new trace links or properties without changing its.

```
// URI defenition
1.  <?xml version = '1.0' encoding = 'UTF-8'?>
2.  <rdf:RDF xmlns=http://www.ontorion.com/ontologies/TraceLinksTaxonomy#
3.  xml:base=http://www.ontorion.com/ontologies/TraceLinksTaxonomy
4.  xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#
5.  xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
6.  xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
7.  <owl:Ontology rdf:about="http://www.ontorion.com/ontologies/TraceLinksTaxonomy#" />
    //Classes definition
8.  <owl:Class
9.  <!-- http://www.ontorion.com/ontologies/TraceLinksTaxonomy#Dependency -->
10. rdf:about="http://www.ontorion.com/ontologies/TraceLinksTaxonomy#Dependency">
11. <rdfs:subClassOfrdf:resource="http://www.ontorion.com/ontologies/TraceLinksTaxonomy#ProductRelatedLink" />
12. <rdfs:subClassOf rdf:resource="http://www.ontorion.com/ontologies/TraceLinksTaxonomy#ReLink" />
13. <rdfs:subClassOf rdf:resource=http://www.ontorion.com/ontologies/TraceLinksTaxonomy#Static
14. </owl:Class>
```

Figure 16: RDF Code for describing trace links

TRQ 5: the taxonomy shall be portable enough to allow easy access for local users (i.e., connected to a private network) or global users (i.e., connected to the Internet). (This will facilitate tools integration.)

TRQ 6: the taxonomy shall not be tailored to a specific environment or application.

8.5 Employing the RDF in Building the Taxonomy

We propose a trace link taxonomy that integrates all existing trace link classifications in a structure that satisfies the requirements proposed in section 8.4. The taxonomy utilizes the OSLC [132] and the RDF [131] for relating all trace links. This idea is borrowed from the semantic web technology in which arbitrary data are linked in a flat structure using the RDF, and referenced by the Uniform Resource Identifier (URI). The proposed structure is a non-hierarchical network of identifiable resources that can be referenced or browsed using the URI. A URI can reference any resource or element such as documents, images, services, a UML diagrams, or a group of other resources by assigning it a unique reference.

The RDF has three components: the *subject* (resource), the *predicate* (property), and the *object* (property value). The *subject* is the element that needs to be described with an assigned unique identifier, the *predicate* represents the characteristic or feature of that element and an *object* is the value of that feature. The *object* in turn can be a *subject* that has other properties, which form nested subjects. RDF files are written using the RDF/XML format or the Web Ontology Language (OWL) which are common formats for writing ontologies. For instance, the code for describing the trace link taxonomy in RDF can be written as shown in Figure 16. The figure has two parts. The top part (lines1-8) defines the taxonomy (i.e., schema, syntax, data types). In this part, lines 2 and 8 indicate the namespace of the taxonomy which represents its URI written in RDF and OWL. The reason for defining another URI in OWL can be seen clearly in the bottom part (lines 9-14). OWL is used to validate the semantics of anything written in RDF. In other words, RDF can be used to describe anything; however, to ensure correct relationships

between things written in RDF, OWL must be used to specify these relationships. Therefore, the *owl* tags in lines 10 and 14 imply a relationship exists between the *dependency* trace link and the other trace links, i.e., *product-related-link, re-link, and static* which are enclosed within the OWL tags. The URI of any trace link in the taxonomy is the concatenation of the taxonomy URI and the trace link name. For instance, the URI of the *dependency* trace link is http://www.ontorion.com/ontologies/TraceLinksTaxonomy#Dependency.

The benefit of using the URI is its uniqueness since this URI is a unique identifier for the *dependency* trace link within this taxonomy. In addition, it can be combined with other URI's anywhere to reason or query about the trace link across many namespaces.

The rationale for employing RDF in creating the trace link taxonomy is manifold:

- The RDF eliminates trace links redundancy among different domains, which means resources can be described only once and referenced as many times as we need.
- The RDF supports multiple inheritance in situations in which a trace link is classified under more than one category.
- The RDF data is portable, it can be transformed into many formats such as XML, HTML, and OWL [138].
- The RDF is a knowledge base language. Unlike other portable languages such as XML, which is used to encode information, RDF can be used to build a data model with a set of rules.
- The RDF data can be visualized graphically as a directed graph, an undirected graph, or a tree.
- Using the RDF, the taxonomy can be built by referencing trace links from local repositories or external resources such as the Internet.
- The RDF provides the reusability of the same data by different users, which adheres to the principle of open linked data.

- The use of the non-hierarchical RDF structure can provide easy navigation; a user can reference any trace link in the hierarchy without having any knowledge about its parent(s) or siblings.
- Using the RDF is a step toward standardization and providing semantics for trace links in Requirements Engineering, Model Driven Engineering, and Systems Engineering.
- The RDF data provides simplicity of access since it is machine-readable data that can be shared with others.
- Using the RDF, it is easy to reason (e.g., what, who) about any trace link in the taxonomy.
- Using the RDF, it is easy to query a taxonomy using query services such as SPARQL [139], and the query can be customized based on user's needs.

8.6 Taxonomy Design and implementation

There are two essential components that should exist in order to build our trace link taxonomy: (a) provide a set of controlled vocabulary (Metadata), which is a collection of terms that have well-defined descriptions across contexts, and (b) identify the relationships between these terms, which constitute the taxonomy. A taxonomy, or Ontology in a broader context, is the knowledge domain which is represented by the collection of terms and the relationships between them [138]. An Ontology can be defined using the OWL [138] which is an extension of RDF. Many organizations standardized their controlled vocabulary and made them available freely for use on the net such as the Friend of a Friend (FOAF) [140] which has standard vocabulary/Ontology for social networks across the web, and the Description of a Project (DOAP) [141] for describing open source software projects.

8.6.1 Taxonomy Design

Our design method relies on our systematic literature review to collect all the terms that refer to trace link types and process them according to the following rules:

- Identify all articles that discuss trace links classification in Requirements Engineering, Model Driven Engineering, and Systems Engineering.
- Identify the terms that describe general trace links types. Usually, these are nouns or adjectives that describe the relationship between artifacts such as *dependency*, *evolution*, and *vertical*.
- Identify the terms that describe a relationship between specific types of artifacts. These relationships between any two artifacts are generally instances of the association between such artifacts. They are usually represented as verbs, for instance, perform, generate, and depend-on.
- Provide a naming convention for the general types. We have done that by screening all the terms that refer to the same type or a relation and give it a unique name. For instance, we considered the *evolution* and *evolutionary* terms as identical terms that refer to a general type (i.e., class), which we choose to call *evolution*. Moreover, we considered the terms *perform*, *performs*, and *performed* as identical terms that refer to a specific relationship (i.e., instance), which we call *perform*.
- Provide a set of properties for instances. Each instance must have unique values that differentiate it from other instances. We limit the properties to include the following: *id*, *name*, *usage*, *type*, and *definition;* although other properties can be added. The *id* is a unique string represented by the *URI* of a class or an instance, it is the concatenation of the taxonomy URI followed by the '#' followed by the class or instance name, which is generated automatically. The *name* represents the instance name, and the *type* represents the *Class name*, the *usage* shows in what situations a class or an instance can be used, and the *definition* of the specified class or instance.

8.6.2 Design Decisions

The taxonomy design decisions are based on the drawbacks of the existing classifications and the requirements stated in section 8.4. We have considered the following design decisions:

- With respect to a trace link or a relation, we differentiate between a *leaf* and a *type* in our taxonomy. A *leaf* is a node that represents a trace link or a relation name that can't have any sibling and cannot be refined anymore; it is like an instance of a class. A *type* is a trace link or a relation that has at least one sub-type or a leaf. In Figure 17, for instance, *Usage* [97] is a leaf since it does not have any sibling, while *dependency* is a *type* since it has leaves such as *Usage, Export* [20], [97]. Leaf names in the taxonomy start with capital letters, while types names starts with small letters as shown in Figure 17. We rely on the classifications provided in Table 14 for the taxonomy design. In other words, we did not make any change or move any trace link from any domain. The changes that we made are providing common names for the trace links or relations that have identical purpose, usage, and definition.
- We removed redundant trace links or relations that share a purpose or usage. This is achieved by finding all trace links or relations that have identical usage or purpose and then creating one node that specifies them. For instance, in the *dependency* type that

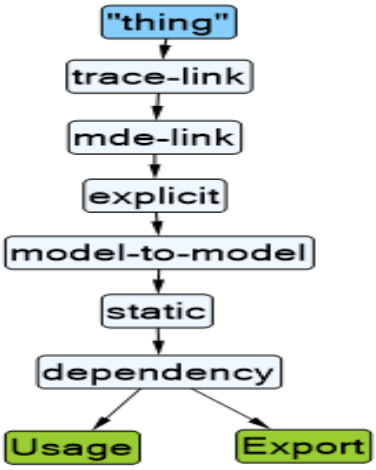

Figure 17: Trace links examples: leaf and type

exists in Requirements Engineering [20], [96] and Model Driven Engineering [97] classifications, we created a shared node which represents the *dependency* relation and showed its hierarchical classification with respect to each domain as shown in Figure 18. Also, we found that Some trace links lead to the same meaning such as *Generalize*

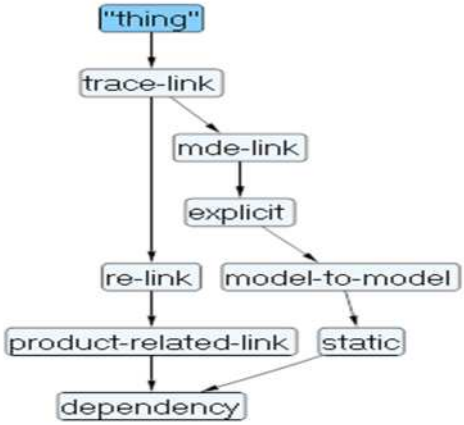

Figure 18: The dependency relation in the taxonomy

and *Is-A*, and *Containment*, *Contain,* and *Has-A*, we kept only one term and removed the rest.

- With respect to trace links semantics, we specified this through two different methods: first by specifying the relationship between nodes, this can be seen clearly in Figure 18; it shows that *dependency* is a *static* trace link which is a type of *MDE* trace link, at the same time, it is a *product-related* trace link which is a type of *RE* trace links. The second method is applying a set of properties for each trace link that specifies its *ID*, name, usage, definition as mentioned in section 8.6.1.

8.6.3 Taxonomy Implementation

We build the taxonomy by following the previous steps and employing the RDF. We used the Fluent editor application [142] for coding the rules of the taxonomy. The editor provides features for authoring complex ontologies that use controlled English as a language for knowledge modeling. It allows users to import and export the knowledge model into different

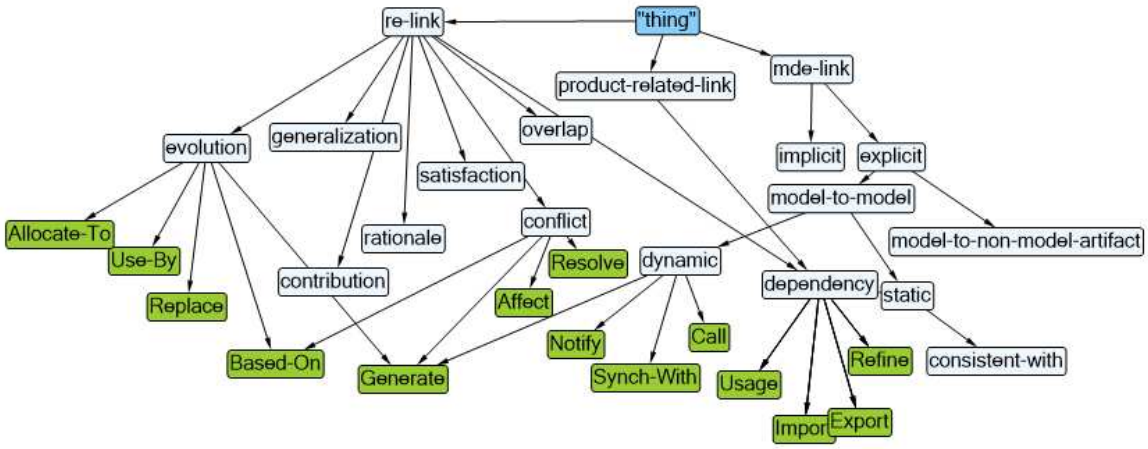

Figure 19: The trace links taxonomy (excerpt)

formats such as RDF, XML, and OWL. In addition, it supports building and visualizing ontologies as interactive diagrams or trees. Finally, the application allows integrating ontologies with the R Language [143], in which quantitative and qualitative analysis can be performed.

The diagram in Figure 19 depicts an excerpt of the taxonomy, it shows a partial view for trace links across the Requirements Engineering, Model Driven Engineering, and Systems Engineering domains and the relationships between them. As the figure depicts, on the top level of the diagram, there is the root of all trace links in the taxonomy, which is represented by the word "*thing"*. Next, there is the *trace-link* element. A *trace-link* is a general type that represents any trace link type in all domains. It has three sub-types, i.e., re-*link, mde-link, and se-link* for the respective trace links types in Requirements Engineering, Model Driven Engineering, and Systems Engineering domains. Following the path of any of these sub-types, we can reach the leaves which represent the trace links of that domain. Each trace link has a set of values, not shown, that define its semantic. We should mention that any trace link is defined once in the taxonomy but it might belong to two or more classes or domains. For instance, at the top left corner of Figure 19, the *Allocate-to* trace link belongs to the *model-to-artifact* and to *evolution* categories. This design eliminates trace links redundancy. The

complete taxonomy can be visualized by running the taxonomy script file which is provided in Appendix B. The script contains complete specifications for all the relationships that relate the trace links to each other. In addition, we included a *read me* file that explains how to run the script in *Fluent Editor* application and generate the whole taxonomy.

8.7 Taxonomy Validation

The taxonomy can be validated by a) ensuring that it satisfies the taxonomy requirements that we proposed in section 8.3 in order to resolve the issues about existing classifications, and b) it can accommodate any traceability problem.

8.7.1 Taxonomy Compliance Criteria

We proposed the following compliance criteria in order to ensure that all requirements can be met:

- **TaxCr 1**: Trace Links redundancy. This criterion supports the claim of the taxonomy requirement **TRQ 3** which requires the specification of a trace link should be specified only once in the taxonomy but it can be related to different domains through different paths.
- **TaxCr 2**: The capability of the taxonomy to accommodate the classification of trace links related to different users in Requirements Engineering, Model Driven Engineering, and Systems Engineering. This criterion supports the claim that the taxonomy can have all types of trace links. This criterion supports the claim of the taxonomy requirement **TRQ 7** which requires that the taxonomy shall integrate all, at least all those we have identified from a systematic search in the literature and therefore from prominent literature, trace link types that can relate the artifacts produced during

system analysis, design, coding, and testing, and during model-to-model transformations.

- **TaxCr 3**: Trace links consistency. This criterion supports whether a trace link has only one definition. This criterion supports the claim of the taxonomy requirement **TRQ 1** in which we specified a set of properties for each trace links that defines its usage or purpose, its name, type, and definition. These properties provide a semantic for each trace link. Moreover, we established relationships between trace links that exist across different domains. Therefore, a certain trace link can be reached through many paths from different domains.
- **TaxCr 4**: Extensibility and maintainability. This criterion supports the claim that it is simple to add a trace link or a property to the taxonomy without changing the existing design. This criterion supports the claim of the taxonomy requirement **TRQ 4**. The flat hierarchy nature of the taxonomy allows inserting any new trace link type in the taxonomy without the need for changing the taxonomy specification. The new specification can be added anywhere in the RDF code. The same argument is applied when we want to add a new property for trace link types; the added properties will apply to all trace links in the taxonomy.
- **TaxCr 5**: Tool Support. This criterion shows whether the taxonomy data can be saved, configured, or exported to other formats or applications. This criterion supports claim of the taxonomy requirement **TRQ 6**. The taxonomy data can be exported in many universal formats such as HTML, XML, and OWL. In addition, the OWL provides not only data serialization but also validation for the taxonomy semantics.
- **TaxCr 6**: The simplicity of referencing the taxonomy data locally or globally. This criterion supports claim of the taxonomy requirement **TRQ 5**. Any trace link in the taxonomy can be referenced by using its own URI.

8.7.2 Validation by Example

Since we do not have an existing taxonomy for trace link types except for the one that we build, our validation cases are based on testing whether our taxonomy can accommodate different solution needs. Therefore, the validation cases are constructed based on the type of the artifacts that can be linked, their level of granularity, and whether the taxonomy can accommodate the traceability rules between the two metamodels that we have discussed in section 8.2. However, we should mention that the validation of the taxonomy will not be complete without being configured with a traceability model. Therefore, the following validation cases are proposed to validate whether the taxonomy can accommodate the traceability between the *i** and UML-Class metamodel based on the traceability rules between instances of the metamodels.

TestCase 1: Relate the model that represents the early-phase requirements to the model that represents the late-phase requirements.

TestCase 2: Relate instances from the *i** metamodel and *UML-Class* metamodel that must have identical names.

TestCase 3: Relate instances of the *i** metamodel that must be mapped to a *Class* instance in the *UML-Class* metamodel.

TestCase 4: Relate *Task* instances from the *i** metamodel to their corresponding *Method* instances in the *UML-Class* metamodel.

TestCase 5: Relate *SoftGoal* and *HardGoal* instances from the *i** metamodel to their corresponding *Attribute* instances in the *UML-Class* metamodel.

In order to validate these cases, we created two object models for the *i** and *UML-Class* as shown in Figure 20. The gray dashed lines in the figure represent possible trace links between the instances of the two models. The following instances are linked together.

- The *i** object model is linked to the UML-Class object model.
- The *Resource* instance (e.g., PickupLocation) from the *i** object model is linked to the *Class* instance (e.g., PickupLocation) from the *UML-Class* object model.
- The *Task* instance (BookCap) from the *i** object model is linked to the *Operation* (bookCap) in the class *DispatcherControl* in the *UML-Class* object model.
- The *SoftGoal* instance (*ShortestETA*) in the *i** object model is linked to the *Attribute* (Disability) of class *CabJob* in the *UML-Class* object model.
- The *HardGoal* instance (*PeopleWithDisability*) from the *i** object model is linked to the *Attribute (ETA)* inside the class *CabJob* in the *UML-Class* object model.

We configured the two models with our taxonomy to obtain the required trace links between source and target artifacts as shown in Table 15. The values in the source and target columns represent a *metaclass: instance*. For instance, *Actor: Customer* means a *Customer* is an instance

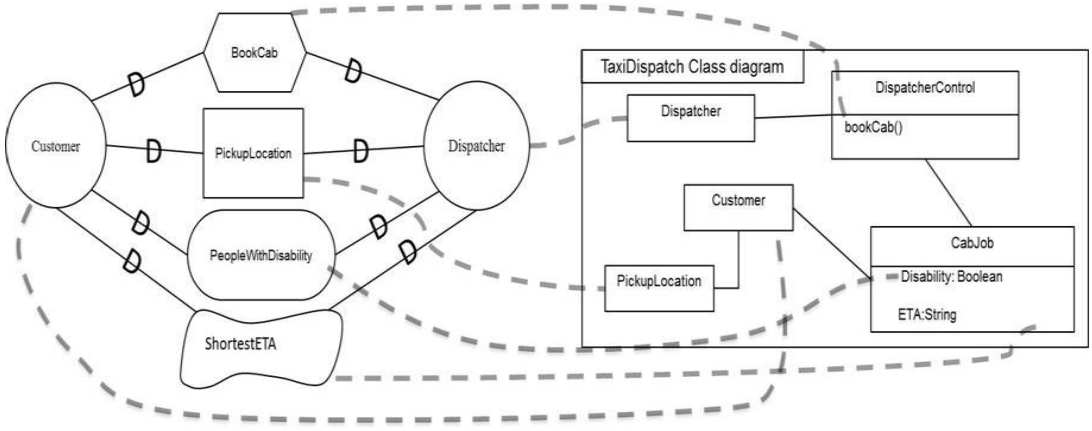

Figure 20: Traceability Example: *i (excerpt) model (left), UML (excerpt) class diagram (right), traceability links (grayed dashed lines** [51]

Table 15: Trace links between instances of the i* an the UML-Class models

Case	i* artifacts (source)	UML artifacts (target)	Trace link	Trace link uri
TestCase 1	i* model	UML-Class model	re-link	http://www.ontorion.com/ontologies/TraceLinksTaxonomy/re-link
TestCase 2	Actor: Customer	Class: Customer	Consistent-with	http://www.ontorion.com/ontologies/TraceLinksTaxonomy/mde-link/Explicit/model-to-model/static#Consistent-with
TestCase 3	Resource: PickupLocation	Class: PickupLocation	Consistent-with	http://www.ontorion.com/ontologies/TraceLinksTaxonomy/mde-link/Explicit/model-to-model/static#Consistent-with
TestCase 4	Task: BookCab	Method: bookCab	Satisfy	http://www.ontorion.com/ontologies/TraceLinksTaxonomy/mde-link/model-artifact#Satisfy
TestCase 5	HardGoal: PeopleWith Disability	Attribute: Disability	Satisfy	http://www.ontorion.com/ontologies/TraceLinksTaxonomy/mde-link/model-artifact#Satisfy

of the *Actor* metaclass. Table 15 depicts the following trace links. We showed these rules previously in Figure 3 and the figure is repeated here in Figure 20.

- The trace link is *re-link* satisfies **TestCase 1** since the purpose of this validation case is to link the two models that represent early and late phase requirements at a higher level of abstraction without looking at their instances.
- The trace link *consistent-with* satisfies **TestCase 2**. We obtained the semantics of this link based on the type of artifacts and the traceability rule in **TestCase 2** which requires the names of the source and target artifacts must be the same.
- The trace link *Consistent-with* satisfies **TestCase 3**. In this case instances of *Actor* or *Resources* can be linked to an instance of a *Class*. The reason for choosing *Consistent with* is the traceability rule that requires the name of the *Class* instance must be the same as the name of the *Actor* or the *Resources* instances.
- The trace link *Satisfy* satisfies **TestCase 4** since the target method *bookCab()* satisfies the source *Task*, BookCab.

- The trace link *Satisfy* satisfies **TestCase 5**. This is configured based on the traceability rule of the validation case since the *Disability* instance satisfies the *HardGoal* instance; *PeopleWithDisability*.

In conclusion, this simple example shows how our taxonomy can accommodate different traceability situations based on the traceability rules between the *i** and the UML-Class metamodels. It is important to mention that we use the *uri* to reference taxonomy trace links.

8.8 Summary

Existing trace links classifications have some drawbacks that affect their usage in relating artifacts from different domains. We proposed a trace link taxonomy that can be used as part of a traceability framework to capture traceability information among artifacts. The taxonomy is built be linking local and global, network resources (taxonomy elements) to form a flat hierarchical structure. We implemented the taxonomy using the RDF technique, which allows referencing elements using their URI. This technique provides many advantages over other classical techniques such as relational or hierarchical structures. It offers interoperability and portability of data among different platforms. Moreover, data are readable by human and machines as well, and it can be transformed into several textual and graphical formats. This can solve the problem of unformatted outputs that are produced by different modeling tools. RDF allows many languages to be referenced using URI references; now we can have a universal machine-processable language similar to XML. The taxonomy can be extended by adding more trace link properties and trace link types, though we believe that thanks to our systematic literature review, we likely have collected and integrated most of that information. Researchers in software engineering and traceability are invited to build upon this taxonomy. Trace link data then can be filtered and edited. We believe this taxonomy can be used as a base for standardizing trace links semantics.

9 Model Validation

This chapter presents the evaluation of the traceability models that we found in literature against our validation criteria. The aim of the validation is to examine whether any of these models complies with the traceability requirements that we proposed in section 7.2. Section 9.1 proposes a series of validation criteria for the traceability requirements and section 9.2 explains the how compliant existing models are to our requirements.

9.1 Validation Criteria

This section discusses the validation criteria that we propose to find out whether the existing traceability models comply with the generic traceability model requirements presented in section 7.2. We want to achieve two objectives: (a) check whether existing traceability models can satisfy all the requirements collectively, (b) ensure that our traceability model can address any drawbacks of existing solutions.

ValCr1. Model shall not be restricted only to trace artifacts of a specific domain or specific metamodel. This validation criterion is related to the requirement Req 1.

ValCr2. Model ability to capture traceability information among homogeneous or heterogeneous artifacts. This validation criterion is related to the requirement Req 2.

ValCr3. Model ability to trace artifacts at the same level or at different levels of abstraction and/or decomposition. This validation criterion is related to the requirements Req 3 and Req 4.

ValCr4. Model capability to link source and target artifacts having different cardinalities. This validation criterion is related to the requirement Req 7.

ValCr5. Model ability to allow the addition of one or more constraint to an artifact or a trace link. This validation criterion is related to the requirement Req 8.

ValCr6. Model ability to characterize a trace element. This validation criterion is related to the requirement Req 9.

ValCr7. Model ability to identify the source and the target from two related artifacts. This validation criterion is related to the requirement Req 6.

ValCr8. Model ability to capture traceability information during model transformation. This validation criterion is related to the requirement Req 11.

ValCr9. Model ability to link two artifacts in both directions, forward and backward. This validation criterion is related to the requirement Req 5.

ValCr10. Model ability to assign or capture version information to a traced element (i.e., an artifact, trace link, trace). This validation criterion is related to the requirement Req 12.

9.2 Validation Cases for Existing Traceability Models

We have discussed several traceability (Meta)models in chapter 5 that aim to provide a mechanism for capturing traceability information between artifacts during the development of software systems. Although each model has some useful features for capturing traceability information among certain artifacts, there is no one model that combines all these features which are required by our problem. Therefore, we propose a test suite that tests the existing model features and checks it against the validation criteria. First, we will describe the validation cases, then we will show whether each model passes or fails these cases based on the validation criteria.

TrTstCase 1: Check the characteristics of traceable artifacts. It checks whether a traceability model traces only specific types of artifacts.

TrTstCase 2: Relate one requirement in the analysis phase to a test case in the testing phase. It checks the ability of a traceability model to link artifacts produced at different phases (vertical traceability).

TrTstCase 3: Trace a requirement to a use case and vice versa. It checks whether a traceability model can link two artifacts; *source* and *target*, in *forward* or *backward* directions (i.e., bidirectional traceability).

TrTstCase 4: Trace one requirement in analysis phase to a use case. It checks whether a traceability model can relate a *source* artifact to a *target* artifact. (1-to-1 cardinality).

TrTstCase 5: Trace one requirement in analysis phase to a use case and a test case. It checks whether a traceability model can link a *source* artifact to more than one *target* artifacts. (1-m cardinality).

TrTstCase 6: Relate a safety requirement to a Simulink model that realizes it. It checks whether a traceability model can relate two heterogeneous artifacts.

TrTstCase 7: Apply a constraint to classes in a Class diagram. It checks whether a traceability model allows a user to apply a *constraint* to an artifact. In this particular instance, the constraint is that no two classes have the same name in a package.

TrTstCase 8: Apply the following constraints to a trace link that relates a requirement and a use case:

- Each requirement should be linked to only on use case.
- Deleting a requirement must delete the trace link that linked it to a requirement.

It checks whether a traceability model allows a user to apply a constraint or more to a *trace link*.

TrTstCase 9: Make a reference to a trace link or an artifact. It checks whether a traceability model can allow a user to uniquely reference a *trace link* or an *artifact* using an identification number, a name, or a reference to its location.

TrTstCase 10: Specify the *Is-A* relation between two classes, *ClassA* and *ClassB,* during the transformation of a class diagram to Java code. It checks whether a traceability model can allow

a user to specify an *explicit* trace link between artifacts of two models during model transformation.

TrTstCase 11: Specify a version to a requirement that is derived from another requirement. It checks whether a traceability model can allow a user to record version control information to a trace element (an *artifact*, a *trace link*, or a *transformation*).

TrTstCase 12: Specify the relation between two requirements as product-related and dependency. It checks whether a traceability model can allow a user to apply more than one type to a trace link based on the purpose of the link or the level of granularity.

TrTstCase 13: Relate a requirement to a trace link such that the requirement is the source and the use case is the target. It checks whether a user can specify in a trace model the source and target artifacts. We believe these validation cases can address our traceability needs based on the traceability requirements we described earlier. Moreover, these cases can show that existing traceability models are inadequate to satisfy our traceability needs.

9.3 Analysis of the Validation Results

Based on our analysis of existing traceability models in chapter 5, we use the validation criteria to validate existing traceability solutions. The result of our validation is depicted in Table 16: Validation of existing traceability models. The table contains 13 validation cases and ten validation criteria that validate nine traceability solutions. As indicated by the table, the validation case TrTstCase 1 is not satisfied by any traceability model since they are domain specific and designed to accommodate specific traceability needs. In addition, these models impose constrains on the traceable artifacts. For instance, the traceability solution by Falleri and colleagues [81] requires that traceable artifacts should conform to the same metamodel. We provided a comprehensive discussion about such drawbacks in sections 5.1 and 5.2. The validation case TrTstCase 10 is satisfied only by one solution [51] since none of the other traceability solutions can capture traceability information for model transformation as required

Table 16: Validation of existing traceability models

Validation Criteria	Validation Case	Satisfied by								
		[90]	[82]	[81]	[89]	[48]	[49]	[88]	[50]	[51]
ValCr1	TrTstCase **1**	No	No	No	No	No	No	No	No	No
ValCr3	TrTstCase 14	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
	TrTstCase 5	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes
ValCr4	TrTstCase 4	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	TrTstCase 5	No	No	No	No	No	No	Yes	Yes	Yes
ValCr5	TrTstCase 7	Yes	Yes	Yes	No	No	No	Yes	No	No
	TrTstCase 8	No	No	No	No	No	No	No	No	No
ValCr6	TrTstCase 9	No	No	No	Yes	No	No	Yes	Yes	No
	TrTstCase 12	No	No	No	No	No	No	No	No	No
ValCr7	TrTstCase 13	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes
ValCr2	TrTstCase 6	Yes	Yes	No	No	No	No	Yes	Yes	Yes
	TrTstCase 4	Yes	No	No	No	No	No	Yes	Yes	Yes
	TrTstCase 5	No	No	No	No	No	No	No	Yes	Yes
ValCr8	TrTstCase 10	No	No	No	No	No	No	No	No	Yes
ValCr9	TrTstCase 3	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
ValCr10	TrTstCase 11	No	No	No	No	No	No	No	No	No
Total Passed		**8**	**6**	**5**	**4**	**4**	**5**	**10**	**8**	**10**
Total Failed		**8**	**10**	**11**	**12**	**12**	**11**	**6**	**8**	**6**

by the requirement Req 11. The table also indicates that the validation case TrTstCase 8 did not pass since none of the existing solutions consider adding constraints to an artifact or trace link. Also, the validation case TrTstCase 11 which is related to assigning a version to a trace element and

TrTstCase 12 which is related to characterizing a trace element are not passed by any traceability model. In conclusion, the validation cases

TrTstCase 1, TrTstCase 8, TrTstCase 9, and TstCase 11 were not passed by any existing traceability model. Moreover, there are only two traceability models that pass a maximum of 11 of our 16 cases. The rest of the traceability solutions passed at most 50% of the validation cases.

In conclusion, these results of the validation cases imply two things: first, since the existing solutions fail some validation cases, our traceability model shall provide adequate solutions for these drawbacks. Second, although some of these models claim to trace heterogeneous artifacts, they fail short on important features such as model extensibility, or tracing artifacts at different levels of granularity.

10 Validation by Example—An Avionics Case Study

This case study follows the guidelines suggested by Runeson and Höst [144]. It is considered to be an independent analysis because the data comes from published papers and Airbus specification documents. This chapter covers the objectives of the case study, data collection, case study and problem description, and validation.

10.1 Objectives

Our case study concerns capturing the traceability information of the requirements and artifacts related to the operations of the Slats and Flaps of the Airbus family of aircrafts (A319, A320, and A321). System engineers found inconsistency in the computer-generated data that trigger movements of the Slat/Flap transmissions. This case study represents a real-world industrial problem that requires traceability in order to study the impact of data inconsistency on the safety of flights in those aircrafts. The objective is to capture the traceability information for the Slat/Flap operations.

10.2 Data Collection

The case study is concerned with identifying the requirement(s) and artifacts involved in the operations of Slat/Flaps that are mainly controlled by the Slat/Flap Control Computers (SFCCs). The following documents [85] provide the sources of our data:

- Standard Specification Document: Produced by Airbus industries, this contains the requirements covering the full aircraft system.
- System Definition Document: Produced by BAe Airbus Company, this contains the flight control specifications.
- Equipment Specification Document: It describes the Slat/Flap mechanical drive specifications that are produced be an external vendor.

- Interface Document: Produced by an external vendor, this contains the mechanical and electrical interface parameters for the Slat/Flap control system.
- System Structure Document: Produced by an external vendor, this contains lower level specifications about the functional and behavioral specifications of Slat/Flaps.

10.3 Case Study Description

The Flight Control Systems (FCSs) in Airbus aircraft provide the functionality for take-off and landing, and consist of three types of computers: the Elevator and Aileron Computers (ELACs); the Spoiler and Elevator Computers (SECs); and the Slat and Flap Control Computers (SFCCs). Redundancy of operations on each computer's type is important to provide system's reliability in the event of disasters, see Mason [85]. In this case study, we focus only on the SFCC, which controls the movements of Slats and Flaps. The functionality of Slats and Flaps is described by Hutchins [145]:

"The wings of airliners are designed to enable fast flight, yet performance and safety considerations require airlines to fly relatively slowly just after takeoff and before landing. The wings generate ample lift at high speeds, but the shapes designed for high speed cannot generate enough lift to keep the airplane flying at low speeds. In order to solve this problem aircrafts are equipped with devices, called Slats and Flaps that change the shape and area of the wing. The part on the leading edge of the wing is called a Slat, while the part on the trailing edge of the wing is called a Flap, both can move along metal tracks built into the wings. They are normally retracted in flight, giving the wing a very clean aerodynamic shape. For slow flight, Slats and Flaps are extended, enlarging the wing and increasing its coefficient of lift. The positions of the Slats and Flaps define the configurations of the wings. In a "clean" wing configuration, the Slats and Flaps are entirely retracted. There is a lower limit on the speed at which the airplane can be flown in this configuration. Below this limit, the wing can no longer produce lift. This condition is called a wing stall. The stall has an

abrupt onset and invariably leads to loss of altitude. Stalls at low altitude are very dangerous. The minimum maneuvering speed for a given configuration and aircraft weight is a speed that guarantees a reasonable margin of safety above the stall speed. Flying slower than this speed is dangerous because the airplane is nearer to a stall. Changing the configuration of the wing by extending the Slats and Flaps, lowers the stall speed of the wing, thus permitting the airplane to fly safely at slower speeds. In order to maintain safe flight at slower speeds, the crew member must extend the Slats and Flaps to produce the appropriate wing configurations at the right speeds".

There are five lever positions in the aircraft shifting gears that are used by the pilot to control the Slat/Flap angle, as shown in Table 17. For instance, when a pilot is taking off and wants to fly at a high angle of attack, he moves the lever from position 3 to position 0, so there will be no retraction for the Slats and Flaps. Under certain conditions, the Slats will not be retracted to position 0, they are held at position 1 until the conditions are safe so they can be retracted to position 0.

10.4 Problem Identification

Slats and Flaps are retracted or extended while an aircraft is flying or landing, which involves synchronization between various software and hardware components in the aircraft. When a pilot changes the Slat/Flap lever to a certain position, a Command Sensor Unit (CSU) converts the selection into a set of discrete electrical signals that are dispatched and validated by each SFCC. Accordingly, each SFCC sends a signal independently to the associated hydraulic

Table 17: Aircraft lever positions

Lever Position	Flight Phase	Flap Angle	Slat angle
0	Cruise	0	0
1*	Hold	0	18
1*	Take-off/ Approach	10	18
2	Take-off/ Approach	14	22
3	Take-off/ Approach	21	22
Full	Land	25	27

solenoid on the Power Control Unit (PCU) in order to allow the hydraulic motors to change the Slat/Flap to the new angle [146]. In A320 and A321 aircraft models, each SFCC produces slightly different outputs as a result of their design by separate vendors. The ambiguity between the outputs of the two SFCCs mandates a traceability for the Slat/Flap requirement through all phases of development in order to measure the effect of the outputs' variations.

The original Slat/Flap requirement for the A321 aircraft states that "Slat retraction shall be inhibited at a high angle of attack". The development of SFCCs that satisfy this requirement is split into two lanes developed independently by two teams that have no interaction with each other. Moreover, the data required to develop each lane is obtained from the Standard Specification document, the Equipment Specification document, and the Interface document. The Standard Specification document is common to both teams, however, the Equipment Specification document and the Interface document are developed by independent vendors. Therefore, creation of the Equipment Specification Documents and Interface document by different vendors leads to variations in the derived requirements from the original one. As a result, an inconsistency in the control signals of the two SFCC lanes can occur. In order to understand the effect of this requirement on aircraft operation, it is important to understand the following concepts (see illustrations in Figure 21):

- The wing chord is the straight line from the leading edge to the trailing edge of the aerofoil (airfoil).
- The Relative Air flow (RAF) is the direction of the air relative to the wing that is not affected by the airfoil. The air affected by the airfoil is the air that is close to it and which is not a relative air flow, see Figure 21.

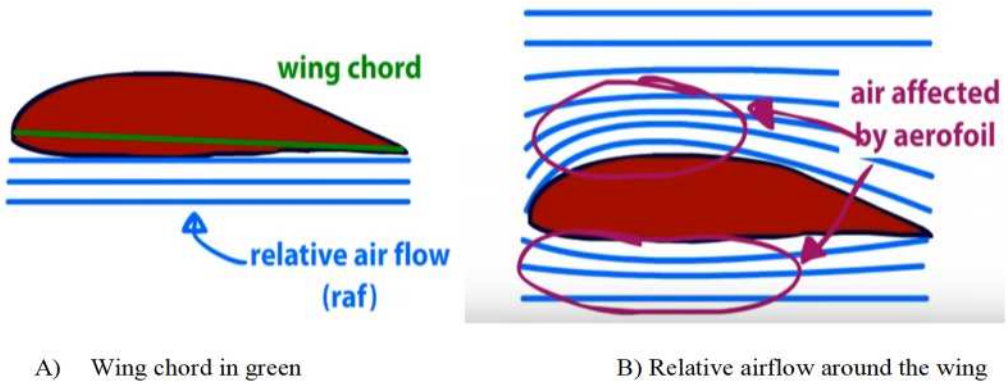

Figure 21: Wing chord and relative air flow, source [151]

- The Angle of Attack (AoA) is the angle between the wing chord and RAF. A small angle of attack can produce a small amount of lift, and a high angle of attack can produce a larger amount of lift.
- The Critical AoA is the angle between the wing chord and the RAF, where any increase beyond this angle will cause a "stall" or generating less lift. Since the only moving part of the wing is the aerlon hanging at its back, it can cause a big change in the AoA while the relative airflow stays the same. Therefore, during aircraft takeoff, a higher AoA is required to achieve greater lift, and thus slat extension is necessary. On the other hand, during aircraft landing slat retraction is necessary to cause less lift.

10.5 Slat/Flap Requirements Specification

The main Slat/Flap requirement derives different requirements at different levels of granularity, the description of the main requirement and its derivations span several documents. We assume that the IBM DOOR requirement management system [127] is used to store the main requirement and its derivations. Moreover, we assume these requirements are refined by UML and Simulink models. The main requirements and their derivations are described below.

- The main requirement of the Slat/Flap RQ 1.1 states that "Slat retraction shall be inhibited at high angle of attack".

- Requirement RQ1.2 is specified in the System Definition Document; it describes further the Alpha-Lock/Speed-Baulk inhibition function for requirement RQ 1.1. The inhibition function prevents the retraction of the slats when the aircraft is operating outside specified flight parameters. The parameters are calculated by the Air Data/Inertial Reference Unit (ADIRU) and supplied to the SFCC.
- Requirement RQ1.3 is specified in the Equipment specification document and provides additional details for RQ1.2. The details show the rationale for the Slat Alpha-Lock/Speed-Baulk function by expressing the conditions that lead to different function states, particularly, the Disable and Engaged. The rationale for adding these conditions is to prevent a stall condition if a pilot accidently selects the zero slats position (see the Slat/Flap positions in Table 17). The Disable state specifies the conditions under which the function cannot be applied. The Engaged state specifies the valid Corrected Angle of Attack (CAoA) and Computed Airspeed (CAS) values for the function.
- Requirement RQ1.4 is derived from RQ1.3 by adding values that lead to any of the specified states in RQ1.3. The Alpha-Lock/Speed-Baulk function is disabled if the slat retraction is set before Alpha is more than 8.5 degrees, or if CAS is less than 148 knots, or if the aircraft is on the ground with CAS below 60 knots. The Alpha-Lock/Speed-Baulk function is engaged if the CAoA is greater than 8 degrees in the A321 aircraft model or the CAoA is greater than 8.5 degrees in the A320 aircraft model. In addition, a constraint is added in order to notify the pilot via the ARINC 429 communication system about this state. The Alpha-Lock/Speed-Baulk function is in the Reset or Not Engaged state if the CAoA value decreases below 7.1 degrees in the A321 aircraft model or below 7.6 in the A321 aircraft model, or the CAS value increases over 154 knots.

- Requirement RQ1.5 provides more details about the functional specifications of RQ1.2. It provides the input and output signals for Slat/Flap retraction that are specified by the Alpha-Lock/Speed-Baulk function in RQ1.2.

10.6 Artifacts Identification

We modeled the specified requirements using the following UML diagrams which include an Activity diagram, a State Machine diagram, and a Component diagram. In addition, we used the Simulink block diagram to model the input/output signals of the Alpha-Lock/Speed-Baulk function:

- State Machine Diagram: The state machine diagram satisfies the different states that are described in RQ1.3, and detailed in RQ1.4. The transitions from one state to another depend on the values of the CAoA and CAS as shown in Figure 22. For instance, in the Disabled state, the Alpha-Lock/Speed-Baulk is not possible if the Slat retraction is set when CAoA is more than 8.5 degrees or CAS is less than 148 knots, or the aircraft is on the ground with CAS below 60 knots. The Engaged state specifies the valid values for CAoA and CAS to apply the Alpha-Lock function. A constraint is added in order to notify the pilot via the ARINC 429 communication system about this state. The state machine diagram consists of the five states: Not Engaged, Disable, Engaged Alpha-

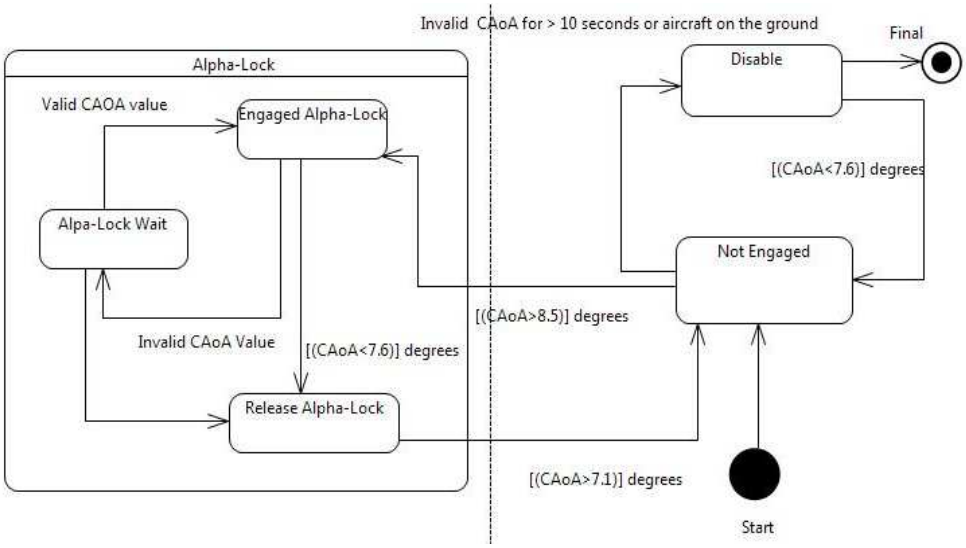

Figure 22: State Machine Diagram

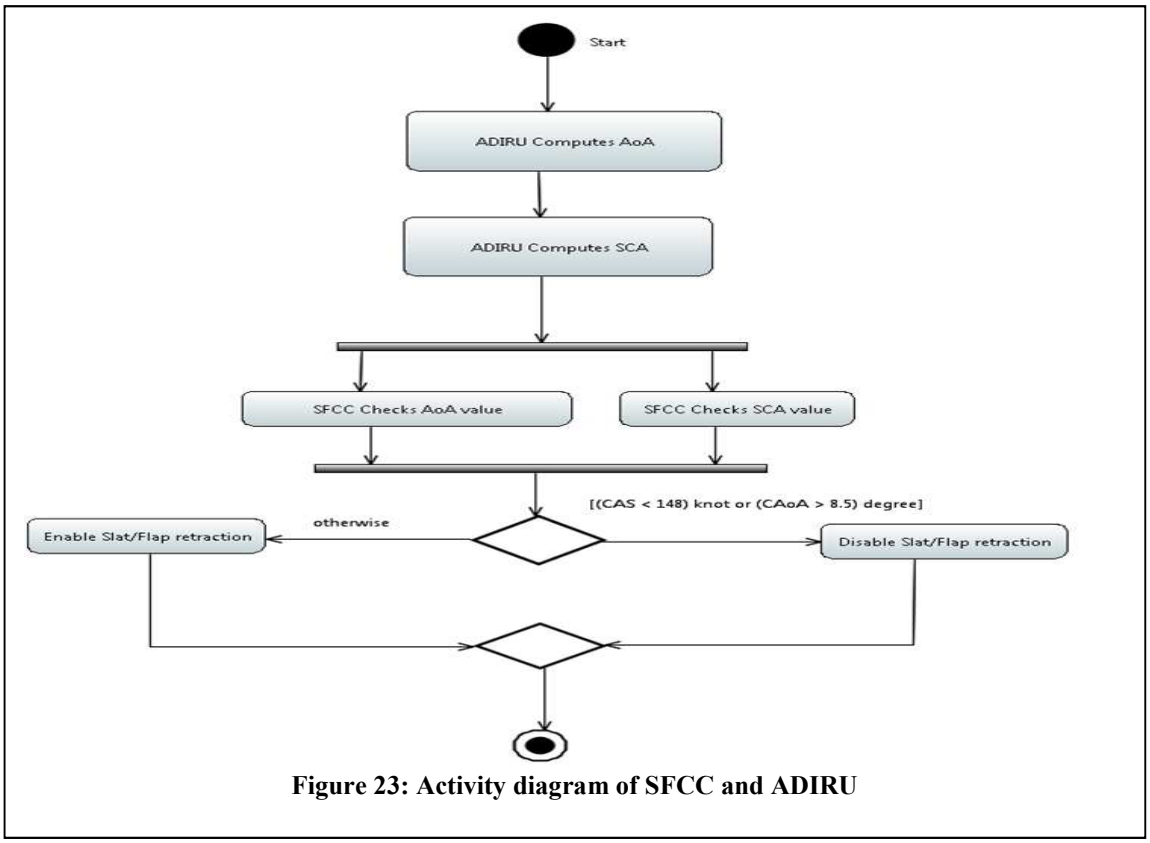

Figure 23: Activity diagram of SFCC and ADIRU

Lock, Alpha-Lock Wait, and Release Alpha-Lock. Another state machine diagram that models the Speed-Baulk of RQ1.4 can be constructed the same way by calculating the value of CAS.

- Activity Diagram: The Activity diagram in Figure 23 shows the different activities performed by the ADIRU and SFCC as stated in RQ1.2. The ADIRU supplies the Corrected Angle of Attack (CAoA) and the Computed Airspeed (CAS) data to the SFCC. The SFCC uses this data to prevent Slat retraction when the CAoA is too large, or the CAS is too low.
- Component Diagram: The Component diagram in Figure 24 shows the relationship between the various physical aspects that realize the Slat/Flap requirements and their derivations. The components interface with each other through different ports. It includes the following components:

  1. Slat/Flap lever: allows the pilot to make his selection to fly at a certain altitude.

  2. Command Selection Unit (CSU): it is responsible for receiving the pilot selection

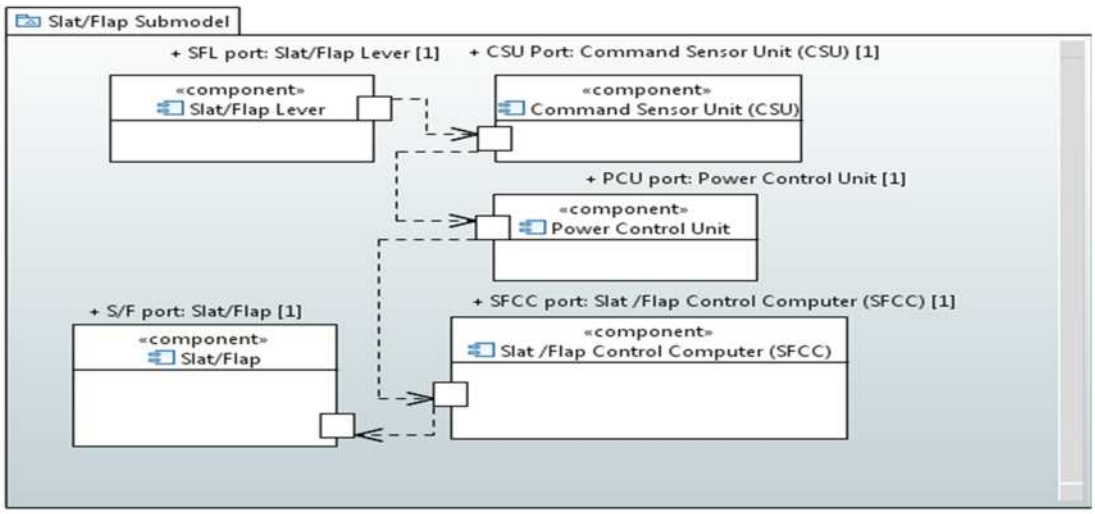

Figure 24: Component diagram for slat flap units

and converting it in to discrete signals.

3. SFCC: it is triggered by the CSU signals; it is responsible for calculating the CAoA and CAS and sending the result to the Power Control Unit.
4. Power Control Unit (PCU): it is responsible for sending signals to the left-hand and right-hand transmissions, which are responsible for tracking Slat/Flap surfaces.
5. Slat/Flap: it receives the signals from the PCU which cause it to extend or retract.

- Signal Flow Diagram: The signal flow diagram provides the input-output signals for

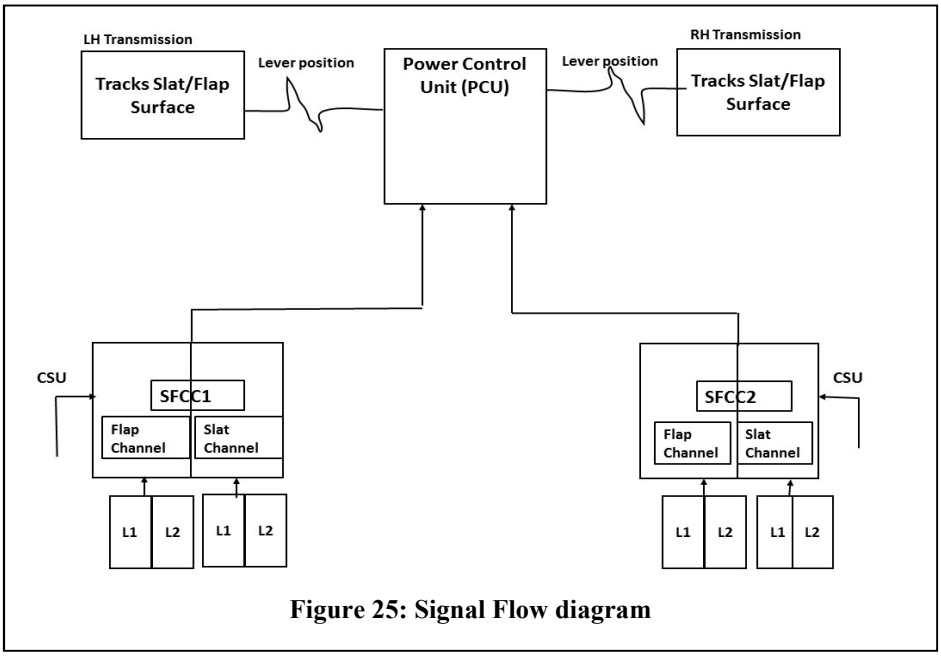

Figure 25: Signal Flow diagram

the Alpha-Lock/Speed-Baulk function as shown in Figure 25. We assume the signal

flow is represented by a block diagram in Simulink. The main block represents the CSU, which receives the pilot's selection as input, converts it to discrete signals, and outputs the signals to SFCC1 and SFCC2, simultaneously. Each SFCC computes the CAoA and CAS independently and sends the results to the PCU simultaneously. The PCU, which is powered by a hydraulic system, receives the CAoA and CAS signals that drive the Slat/Flap movements in the aircraft. The main block represents the Power Control Unit (PCU), which is powered by a hydraulic system, then sends its own signals to the left-hand and right-hand transmissions, which are responsible for tracking Slat/Flap surfaces.

10.7 Identifying Trace Links

Table 18 presents the possible cases for linking the Slat/Flap requirements and artifacts. The trace links among the requirements or artifacts are configured based on the level of granularity, the type of the artifacts, and the purpose for establishing a trace link, according to our trace

Table 18. Identifying requirements and artifacts from the Slat/Flap case Study

No.	Source artifact	Target artifact	Trace link		
			Main type	Subtype	Leaf-type
1	RQ1.1	RQ1.2	re-link	evolution	Derive
2	RQ1.1	RQ1.3	re-link	evolution	Derive
3	RQ1.1	RQ1.4	re-link	evolution	Derive
4	RQ1.1	RQ1.5	re-link	evolution	Derive
6	RQ1.2	RQ1.3	re-link	evolution	Derive
7	RQ1.3	RQ1.4	re-link	evolution	Derive
8	RQ1.4	RQ1.5	re-link	evolution	Derive
9	Activity Diagram	RQ1.2	se-link	Artifact-requirement	Satisfy
10	State Machine diagram	RQ1.4	se-link	Artifact-requirement	Satisfy
11	Component diagram	RQ1.2	se-link	Artifact-requirement	Satisfy
12	Block diagram	RQ1.2	se-link	Artifact-requirement	Satisfy
13	Block diagram	Component diagram	se-link	vertical	Intra-Micro
14	Block diagram	Activity diagram	se-link	vertical	Intra-Micro
15	Activity diagram	Component diagram	se-link	vertical	Intra-Micro

links taxonomy.

As indicated in Table 18, at a coarse level of granularity, a trace link can have either a Model Driven Engineering, Requirements Engineering, or Systems Engineering link type. At a fine

level of granularity more types can be inferred based on the classification specified in the taxonomy. For instance, at a coarse level, the trace link between RQ1.1 and RQ1.2 has the *RE* type since the two requirements are linked together in the elicitation phase. However, in order to specify exactly how RQ1.1 and RQ1.2 are related to each other, a fine-grained traceability is required. Therefore, the trace link between the two requirements is specified as a *Derive* trace link since requirement RQ1.2 is derived from requirement RQ1.1. Other types of trace links are specified in Table 18 based on the needs explained in the Slat/Flap case study. For instance, the requirements are traced also to some design elements such as activity diagrams and block diagrams. Moreover, the case study would involve other artifacts if this needs to be implemented. These artifacts need also to be traced, for instance, a requirement can be traced to code that implements it, or to a test case that verifies it, or a hardware that satisfies it. The table also shows traceable artifacts from different or the same domains. For instance, number 12 in the table shows a requirement from Requirements Engineering domain is traced to a block diagram implemented by SysML language from the Systems Engineering domain.

Table 19: Validation cases for validating the traceability model

validation Case	Description of Validation case	Source Artifact	Target Artifact	Criteria	Object Diagram
1	Apply characarization to a trace link between two artifacts	▪ RQ1.1	▪ RQ 1.2	ValCr6	Figure 26
2	Trace one source artifact to one target artifact	▪ RQ1.1	▪ RQ 1.2	ValCr3, ValCr7	Figure 27
3	Trace one source artifact to more than one target artifact	▪ RQ 1.2	▪ Component diagram ▪ Activity diagram	ValCr4	Figure 28
4	Apply characterization to two artifacts	▪ Block Diagram	RQ 1.3	ValCr6	Figure 29
5	Capturing version control information	▪ RQ1.1	▪ RQ 1.2	ValCr10	Figure 30
6	Trace source and target artifacts in both directions	▪ RQ 1.1	▪ RQ 1.3	ValCr9	Figure 31
7	Apply a constraint into a trace link	▪ RQ 1.2	▪ RQ1.3	ValCr5	Figure 32
8	Trace artifacts belong to different domains/levels	▪ Block daigram	▪ RQ1.3	ValCr2	Figure 33

10.8 Traceability Model Validation

We decided to create validation cases to check whether our traceability model can capture the traceability information for the artifacts and requirements of the Slat/Flap case study. We constructed ten validation cases (Table 19) from the data provided in Table 18. For each validation case we provide a validation case number, description, and the source and target artifact that are involved.

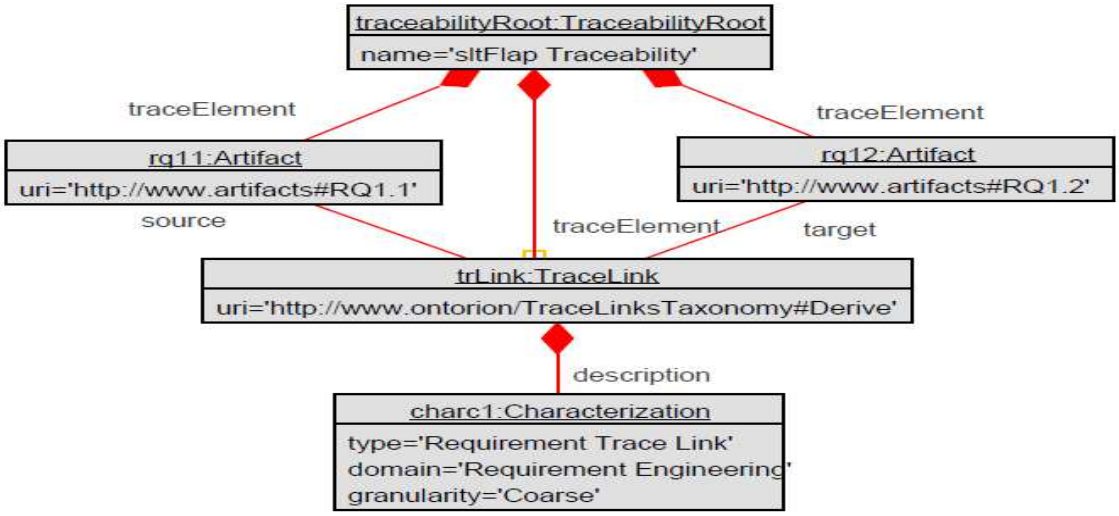

Figure 26: Object traceability diagram for applying characterization to a trace link

We created eight traceability object models that capture the traceability information for the eight validation cases that we present in Table 19. We used the USE UML based tool [130] to instantiate these models. The traceability object model for each validation case is shown seperately for clarity. The validation cases in the table validate the following aspects:

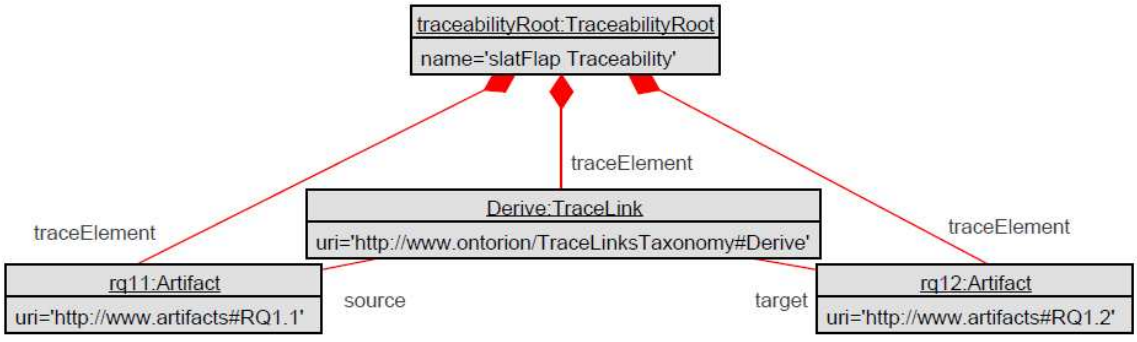

Figure 27: Object traceability diagram for tracing a source artifact to a target artifact

1. Validation case 1: it tests whether the traceability model can capture the traceability

information related to applying characarization to a trace link. The solution is shown in Figure 26. RQ1.1 is the source artifact and RQ1.2 is the target artifact and the trace link between the two requirements is *Derive.* The charaectrization is applied to the *Derive* trace link by adding values for the *type*, *domain*, and *granularity* attributes. This solution is validated by the validation criterion ValCr6.

2. Validation case 2: it tests whether the traceability model can capture traceability information between a single source artifact and a single target artifact. The solution is

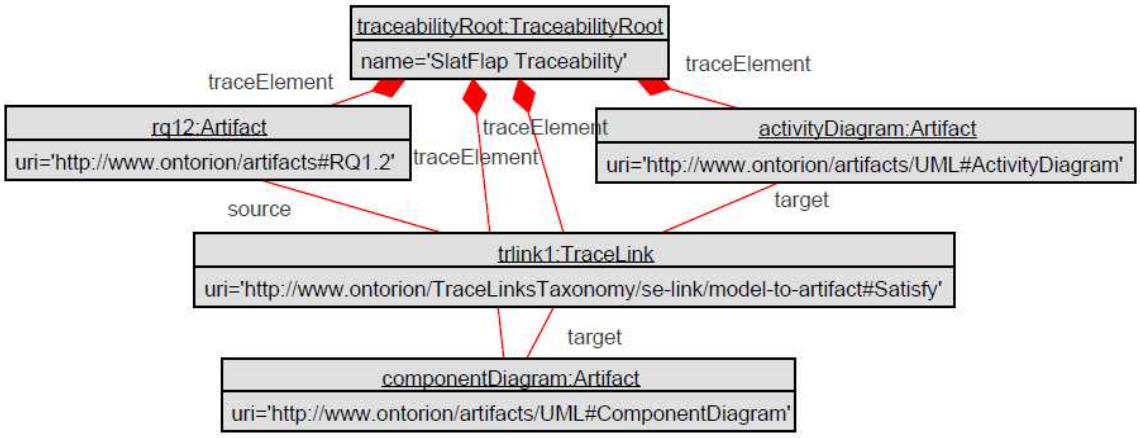

Figure 28: Object traceability model for tracing a source artifact to more than one target artifacts

shown in Figure 27. RQ1.1 is the source artifact and RQ1.2 is the target artifact and the trace link between the requirements is *Derive*. RQ 1.2 is derived from RQ1.1. The *uri* is used to identify the trace links and the requirements. This solution is validated by the validatation criteria ValCr3 and ValCr7.

3. Validation case 3: it tests whether the traceability model can capture the traceability information of tracing a source artifact to more than one target artifact. The solution is depicted in Figure 28, the source artifact is RQ1.2 and the target artifacts are an Activity diagram and a Component diagram. The solution is validated by the validation criterion ValCr4.

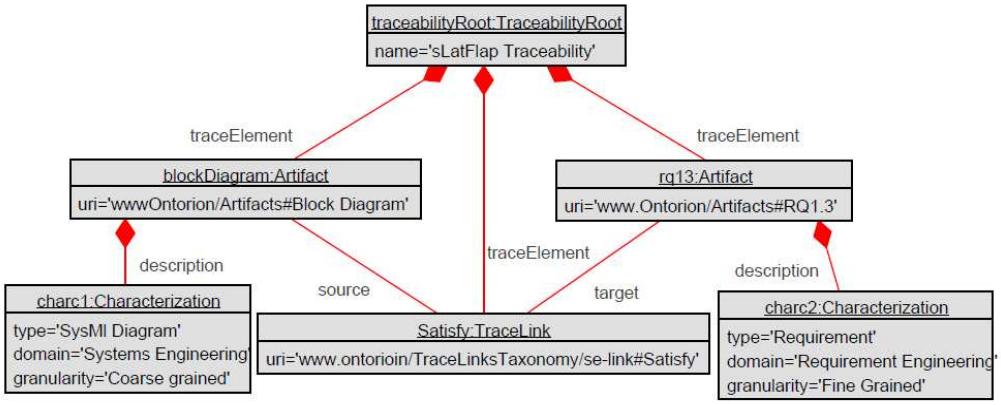

Figure 29: Object traceability diagram for artifacts characterization

4. Validation case 4: it tests whether the traceability model allows characterizations of artifacts. The solution is depicted in Figure 29. We applied characterizations for two artifacts: RQ1.3 and a Block diagram. The solution is validated by the validation criteria ValCr6.

5. Validation case 5: it tests whether the traceability model can capture version control information. It satisfies the validation criteria ValCr10. The result is depicted in Figure 30; version control information is added to block diagram and RQ 1.3.

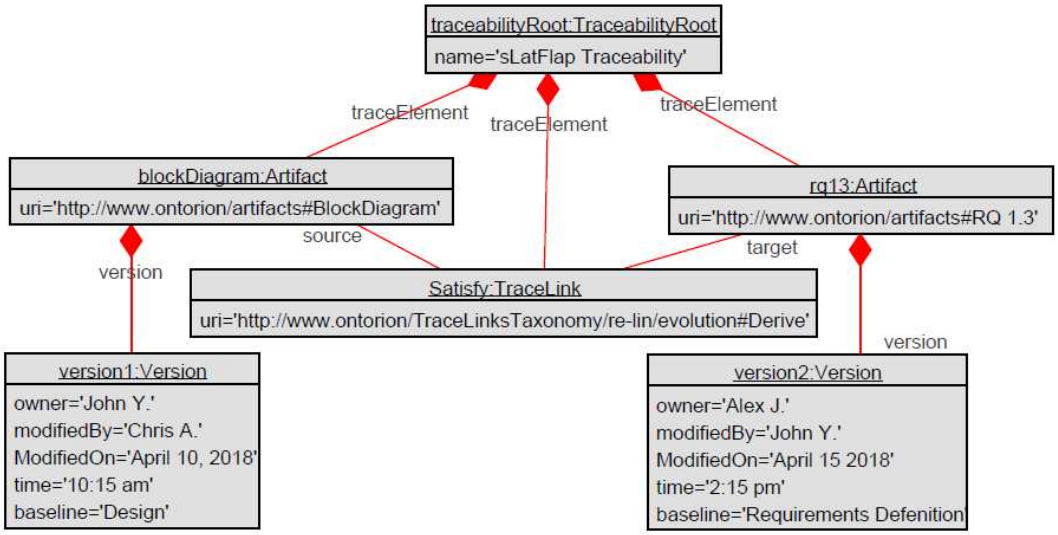

Figure 30: Object traceability diagram for version control information

6. Validation case 6: it tests whether the traceability model can capture the traceability information related to tracing artifacts in forward and backward directions. The diagram

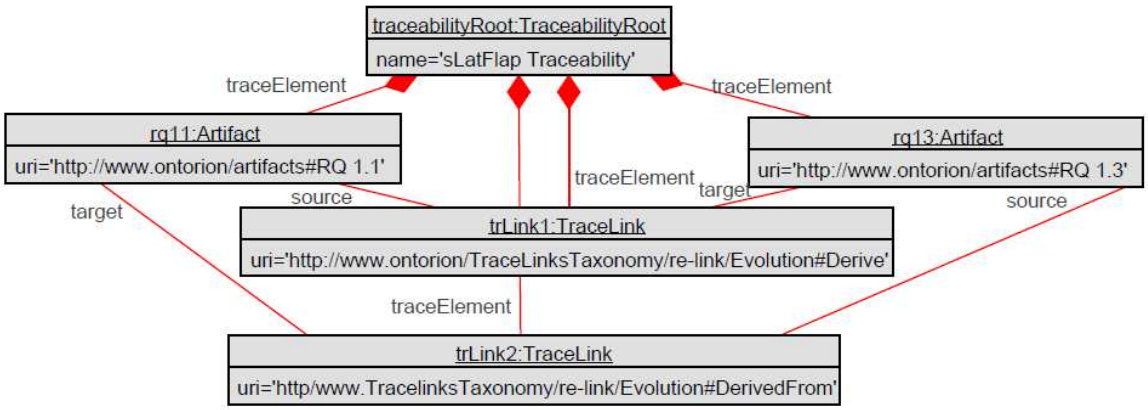

Figure 31: Object traceability diagram for bidirectional traceability between two artifacts

in Figure 31 shows the requirements rq11 and rq13 are traced in forward and backward directions. The forward trace link from rq11 to rq13 is *Derive*, and the backward trace link from rq13 to rq11 is *DerivedFrom*. The validation case satisfies the validation criteria ValCr9.

7. Validation case 7: it tests whether the traceability model allows applying a constraint to an artifact. Since we did not implement our traceability management system, we

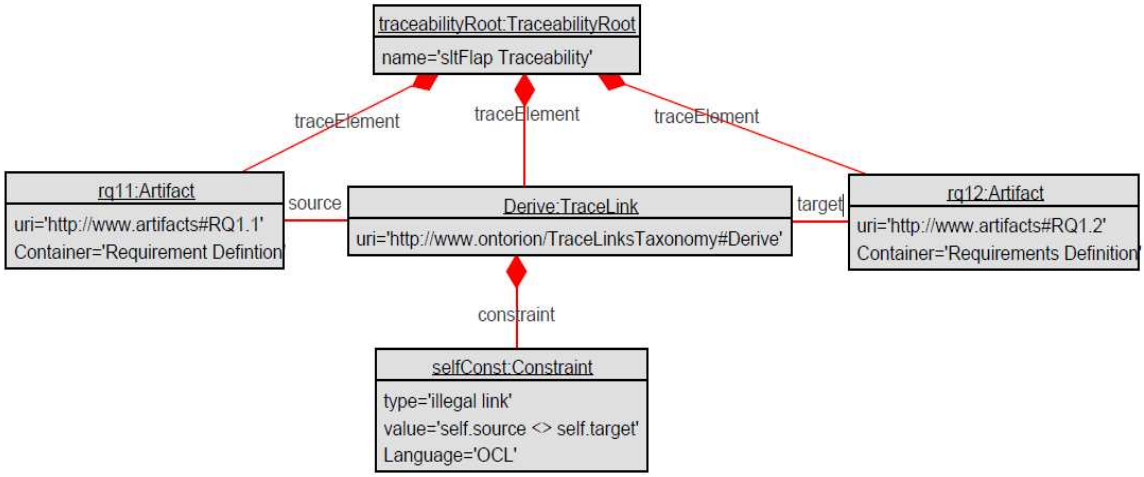

Figure 32: Traceability object diagram for applying a constraint on trace link

simulated this by writing a constraint on the TraceLink class such that the source and target artifacts of a trace link cannot be the same. The validation case validates the validation criteria ValCr5. Figure 32 shows the result of applying a constraint on the *Derive* trace link.

8. Validation case 8: it tests whether the traceability model allows tracing artifacts from

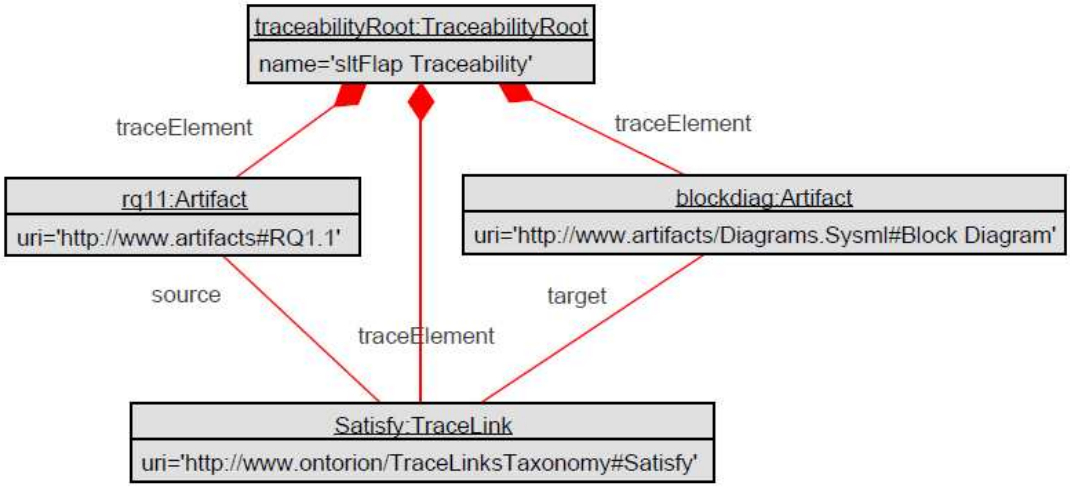

Figure 33: Object traceability diagram for tracing artifacts from different domains

two different domains. This validation case satisfies validation criteria ValCr2. The diagram in Figure 33 shows the object model for linking requirement RQ 1.1 from the Requirements Engineering domain to a SysML Block diagram from the Systems Engineering domain

10.9 Conclusion

This chapter describes a case study from the avionics domain for validating a generic traceability model. The case study provides a realistic Systems Engineering situation in which a requirement produces many artifacts that span across multiple software and hardware components. We showed that the main Slat/Flap requirement produces four other successive requirements. The derived requirements add more details and enforce more conditions that controls the movements of Slat/Flap transmissions. Moreover, we implemented several UML and Simulink diagrams that realize the functionality of these requirements. We chose different traceability scenarios that represent validation cases for our traceability model. These cases require linking requirements and artifacts at different levels of granularity and across domains. Our experimentation shows that our generic traceability model and the trace links taxonomy accommodate all the traceability needs presented in this case study.

11 Threats to Validity

A validity threat refers to the situation in which a researcher might be wrong. This can be identified during validity analysis. Therefore, in discussing validity, we are concerned with the relationship between our experimental conclusions and reality and how the conclusions might be wrong [147]. Researchers in empirical software engineering are often confined with options for validating their new techniques or research. They can either have control on their experimentation by introducing certain parameters (internal setting) or observe the behavior of their experiment in the real world (i.e., external setting) [148]. There is a tradeoff for using any of these settings; in both cases there might be threat to the validity. Wohlin and colleagues [119] suggest four types of validity threats: conclusion, internal, construct and external. Conclusion validity is concerned with how much the experiment setting has an influence on the observed result. In other words, is the treatment we used really related to the outcome? Internal validity focuses on the causal relationship between the treatment and the observed result; it is about how much we are confident that the treatment causes the result. Construct validity focuses on the relation between the theory behind the experiment and the outcome. External validity is concerned with whether we can generalize the results outside the scope of our study even if a causal relation is established.

11.1 Strategies for Mitigating Validity Threats

Our research involves several research methodologies [147] that require validation. As a result, threats to the validity of these methodologies might occur during our experimentation. The objective of our research is to develop a framework for engineering the traceability of heterogeneous artifacts. We apply the following research methodologies in order to achieve our objective:

OB. 1: Collect requirements from our industrial partner that identifies the research problem.

OB. 2: Conduct a systematic literature review about traceability in Requirements Engineering, Model Driven Engineering, Systems Engineering domains.

OB. 3: Conduct a survey to get a feedback from industrial professionals who practice traceability in these domains.

OB. 4: Propose a set of requirements for a traceability model.

OB. 5: Design a traceability model that satisfies the user requirements based on the findings of the literature review, survey, and the proposed requirements.

OB. 6: Design a trace links taxonomy that can be utilized by the traceability model.

OB. 7: Build a case study that can validate the traceability model using the trace links taxonomy.

The strategy that we use to mitigate threats to the validity of the research relies on mitigating the threat for each sub-objective (i.e., OB. 1- OB. 7). Therefore, each sub-objective is validated internally by setting some parameters that mitigate its validity threats. Moreover, we ordered the sub-objectives in a certain sequence such that each sub-objective is validated by the next sub-objective(s) in the sequence. For instance, in order to satisfy the user requirements, we conduct a systematic literature review and a survey in OB. 2 and OB. 3 before proposing the traceability model requirements in OB. 4. The design of the traceability model in OB. 5 is supported by our findings OB. 6 and OB. 7, and the proposed requirements in OB. 4. Also, the design of the trace links taxonomy in OB. 6 is supported by the literature review and the survey findings. The validation of the traceability model and the trace links taxonomy is supported by building the case study in OB. 7.

This chapter describes possible threats to validity throughout the phases of our research. We provide safeguards that can ensure validity and reduce the bias in our experimentation. We

achieved this by considering two aspects: (a) preventing the factors that contribute to producing unwanted outcomes during our experimentation, and (b) ensuring that we are applying the right validation technique that leads to the right results. Table 20 provides a summary for the types of threats that can affect our research and the mitigation techniques we deployed. The table shows the software methodologies that we used during our research such as conducting the survey or the case study. In addition, we showed the possible threats to validity which includes

Table 20: Identified threats to validity

Software Methodology	**Identified threats C=Conclusion I = Internal E = External**			**Techniques for Mitigating Threats**			**Section**
	C	**I**	**E**	**C**	**I**	**E**	
Literature review	X	X	X	• Search string selection • Search major libraries in CS. and Engineering. • Exclude out of context articles • Allow only peer-reviewed articles	Conduct a survey	Traceability Publications that are published after conducting the literature review	11.2
Traceability Survey	X	X		Select the right audience	• Select the right type of questions • Rely on user requirements • Rely on data from literature review		11.3
Traceability Requirements		X			• Rely on user requirements • Rely on data from literature review • Rely on survey responses		11.4
Traceability Model Design		X	X		• Rely on user requirements • Rely on data from literature review • Rely on survey responses • Rely on proposed req. • Validation by construction • Conduct a case study	• Model testing by industry. • Model instantiation	11.5
Trace links Taxonomy					• Rely on user requirements • Rely on data from literature review • Rely on survey responses • Conduct a case study		11.6
Case Study	X	X		• Investigate real-world problem	Follow software Eng. Practices Get technical spec. from the company documents		11.7

the conclusion, internal, and external types; these are represented in the table by the C, I, E, respectively. These threats and mitigation mechanisms are described in the following sections.

11.2 Mitigating Threats to the Validity of the Systematic Literature Review

Traceability in literature is used in different contexts. For instance, there is a traceability for agricultural products, traceability for chemical compounds, and traceability for artifacts in Systems Engineering, Model Driven Engineering, and Requirements Engineering. In our literature review, we are concerned only with articles that discuss traceability of artifacts in the context of Systems Engineering, Model Driven Engineering, and Requirements Engineering. The parameters that we set before and after conducting the review consider several factors that mitigate threats to the validity of our search. First, we constructed a search string that covers traceability of artifacts (i.e., homogeneous or heterogeneous) in Systems Engineering, Requirements Engineering, and Model Driven Engineering domains. This is ensured by combining the word "traceability" with each field using the AND operator and inserting the OR operator between other words (see the search string in section 3.2). Second, we searched the major digital libraries in computing and engineering in order to retrieve the maximum number of publications related to traceability in the abovementioned fields. The libraries include: the IEEE Xplore, ACM Digital Library, Google Scholar, Science Direct, and Springer Link. We decided to use these libraries for the following reasons:

a) They are classified as the top international electronics libraries that reference publications of reputable journals and conferences in the field of computing and engineering.

b) They are typically used by many scholars in academic institutions as main sources for peer-reviewed research papers.

c) They were used by similar reviews and surveys conducted prior to this review [35, 57-61].

d) They are among the digital libraries recommended for systematic literature reviews in Software Engineering [29], [32], [33], [39], [79].

Third, we classified the retrieved articles based on their relevance to the problem domain. This was achieved in two steps: the first step is to remove unrelated articles by scanning the title of each retrieved article and exclude out of context articles. In the second step, we considered only articles that come from authenticated sources such as journals, conferences, and edited scientific books. These two steps reduce the threats to conclusion validity since we want to prevent any unwanted article from being included in our review.

With respect to external threats to validity, we limited our literature review for publications until 2016 which is the date of conducting the review, however, there might by some traceability related publications that might appear after this date. In such case, the findings of these publications were not considered in our research.

Finally, we mitigate internal threats to validity in our literature review by conducting an online traceability survey in order to supporting our findings. The survey focuses on getting a feedback from industry about traceability tools and traceability practices. Chapter 6 provides details about our traceability survey.

To conclude, we try to alleviate the threats to validity in the literature review in five ways: a) construct a search string that can retrieve articles related to traceability, b) search the major libraries in computer science and engineering, c) exclude the out of context articles, d) allow only peered-review articles, and e) conduct a survey about traceability practices and traceability tool which will be described next.

11.3 Mitigating the Threats to the Validity of the Survey

The purpose of conducting the survey is to get feedback from professionals in industry about their traceability practices and traceability tools. We incorporate our findings from the systematic literature review [117] and the needs of our industrial partner at CAE [149] in

constructing the questionnaire. The aim of the questionnaire is to understand the drawbacks of existing traceability tools, the types of traceable artifacts, and the types and usage of trace links. In order to support our survey findings, the results of this survey can be compared with results of the literature review and used further to support the design of our traceability model.

Conducting the traceability survey serves more than one purpose; first, it can support the researcher's assumptions about the requirements of the traceability model and the taxonomy since it targets only industrial professional who practice traceability in their work. Second, the results of the survey can provide us with ideas for proposing traceability model requirements. Third, the collected data, and hence, the proposed requirements can provide us with important suggestions for the design of the traceability model and the trace links taxonomy. This means conducting the survey participate in mitigating the validity threats to the abovementioned.

However, threats to validity can be questioned also with respect to the survey itself. The following questions raise concerns about the validity of the survey itself:

- What is the rationale for using a survey, instead of other data gathering instruments?
- How are the questions of the survey formulated?
- How the audience of the survey is decided?
- How the participants' responses impact our model and taxonomy designs?

Regarding the rationale for choosing the survey, we believe the use of the survey can help gathering qualitative and quantitative data in a systematic way. The survey can offer the use of questions that have subjective or objective responses. Subjective responses represent personal perceptions, feeling, and attitudes; objectives responses are based on facts or personal experience. In contrast to other data collection methods such as interviews and document reviews, the use of online surveys can provide flexibility to the researcher. It can reach global participants, which supports diversity. The data collection can be faster, and the number of participants can be larger.

Regarding the survey questions and the targeted audience, the questions are very technical which support our research objectives, the survey aims to collect specific information about traceability practices and tools. In addition, the survey is not targeting any industrial professional, but instead targets professional who are experts in the (traceability) domain and who are applying traceability in their workplace.

Finally, with respect to the impact of the responses about our model design, we believe the responses can enrich our design since they are coming from practitioners who practice traceability in real world problems, therefore, we can rely on their responses.

The validity of the abovementioned concerns participates to the validity of the survey as a whole. Therefore, our choice for the right audience is considered a conclusion validity. Again, we are not looking for random answers from participants who have no knowledge in traceability or never practice it. Also, we are looking for opinions of industrial professionals not academic opinions since this is covered in the literature review. The choice of questions by the researcher might raise the issue of bias, however, the questions are not representing our opinion. They are based on requirements from our industrial partner, and some data collected from the literature review.

Therefore, we can conclude that alleviating threats in the survey is achieved by: a) selecting the right audience from industry, and b) the choice of the survey questions which is based on customers, needs and literature review, and c) rely on the literature review data and the user requirements to propose the survey questions. Finally, conducting the survey provides another factor for mitigating the threats of the traceability requirements that we are proposing.

11.4 Mitigating the Threats to the Traceability Model Requirements

Our research involves a proposal for generic traceability model requirements. As explained earlier, the requirements are proposed based on customers' needs, and supported by data gathering from the literature review and the traceability survey. Threats to the validity of the

requirements are alleviated through these undertaken software methodologies prior to the proposal of the requirements. These methodologies shape the type of requirements for a generic traceability model and reduces the researcher's bias.

11.5 Mitigating the Threats to the Traceability Model Design

Since there is a possibility of employing the traceability model in real-world applications, we consider the internal and external threats to validity. The internal threats to validity resemble the limitations of the model that prevent it from capturing some traceability information during our experimentation. On the other hand, the external threats to validity represent the situations in which the model cannot accommodate the capturing of traceability information of certain artifacts when it is employed in real world applications.

11.5.1 Internal threats to validity

Internal threats to validity relate to some design issues of the traceability model. We believe these issues can be related to the design of some classes or constraints that prevent the model from accommodating some traceability data. We applied the following techniques during our experimentation in order to minimize the threats caused by these concerns:

- Expressing the research questions: these questions identify the requirements of the traceability model that we want to design, and the kind of traceability information it shall accommodate. The research questions start from the need of our industrial partner to design a traceability model that can accommodate the capturing of traceability information of heterogeneous artifacts. We carried out these requirements to the next level which is the literature review.
- Conducting a systematic literature review: our systematic literature review acts as a first line of defense against the threats to validity of the model design. We study the design of existing traceability models and summarize their benefits and drawbacks. Several ideas from existing models are incorporated into our model design.

- Conducting an online survey: the online survey plays a crucial role in limiting threats to validity of the traceability model. The survey findings represent the traceability model requirements from the point of view of industrial practitioners who are considered as stakeholders. In addition, it increases our confidence that no research bias has occurred during our design.
- Validating the traceability model by construction: the validation by construction is another safeguard against the threats to the validity of our design. We justify the rationale for the creation of every class, attribute, constraint, or association of our model solution. These justifications are based on the collected data from our industrial partner, the context of the problem, the systematic literature review, and the online survey.
- Validating the traceability model using a case study: this is the last approach to safeguard our model against threats. We prepared an industrial case study from the aviation domain that represents a real-world problem. We prepared a set of validation cases that resemble real world requirements. Our traceability model was instantiated based on the requirements specified in these validation cases, we found that all validation cases that have been tested passed.

The second type of threats is related to the constraints of the traceability model which may prevent it from capturing some traceability information. These threats can be minimized by understanding the type of artifacts to be traced. The proposal of using a unique URI to reference requirements or artifacts in general is an example of a mechanism to mitigate threats. We assume that users can use our traceability model to record traceability information about artifacts directly or the traceability information can be read from repositories such as semantic web. This assumption can suit many artifacts but cannot suit every single artifact. For instance, not all requirements are established in a systematic way; requirements might be written in a

pdf or a text file where they need to be parsed in order to identify each requirement. In any case, we can assign a suitable URI value that uniquely references an artifact.

11.5.2 External threats to validity

External threats may occur during the use of the model in a real industrial environment. Mitigating external threats to design validity can be understood partially from our experimentation. However, testing the traceability model by industrial professionals who apply traceability in their workplace and get their feedback can also mitigate threats. The feedback can be taken into consideration to enhance the design of the model. Moreover, the methodologies that we applied from the beginning of our research can have great impact on alleviating the external threats to validity.

11.6 Mitigating Threats to the Validity of the Trace Links Taxonomy

In principle, the threats to the validity of the trace links taxonomy are similar to these discussed for the traceability model. Threats can be internal or external. We followed the same procedure that we used in the case of the traceability model in order to identify those threats and minimize their effects. We conducted a literature review to identify the existing classifications of trace links in the Requirements Engineering, Model Driven Engineering, and Systems Engineering domains. The common trace links across all domains are identified and the duplicated types are removed. We provide a metadata for each trace link that specifies its proper use and semantics. Finally, the use of semantics web to implement the taxonomy can allow future additions of new types without changing the structure of the taxonomy.

11.7 Mitigating Threats to the Validity of the Case Study

Introducing a case study is another methodology for validating the traceability model and the trace links taxonomy. We believe the case study that we employ from the Airbus Company to validate our traceability model and the taxonomy can mitigate threats to conclusion validity since it resembles a real industrial problem. Moreover, we contemplate several factors that

alleviate the internal threats to the validity of the case study. First, this is a real case study not a hypothetical one. We obtained the description of the problem from literature though the technical specifications are obtained from documents of the Airbus Company. Second, the requirements definition of the problem is obtained from the company's documents which implies accuracy of the problem definition; we simply modeled these requirements using UML. Third, there are no assumptions by the researcher other than the assumption about the tools used to model the artifacts in the case study which prevents the researcher's bias.

11.8 Remaining Threats

In the previous sections of this chapter we consider how we mitigated some threats to the validity of our research. However, there are remaining threats about our work, which we discuss below.

11.8.1 Remaining Threats to Survey

The survey questions were constructed based on the drawbacks of the traceability model and tools that we found during our literature review. After we conducted the survey and collected the answers we discovered some remaining threats. First, in some questions the survey does not provide enough flexibility to the participant for his selection. For instance, in Figure 9 question 13 the survey discusses the cardinality of traced artifacts. It provides the participant with four possible options for cardinality between source and target artifacts. There are situations in which the participant might apply in his work more than one option. However, the survey limits the participant to select only one option. Although choosing one option can provide an answer however, the answer is not precise. Second, some questions have binary options (e.g., yes or no options) which might not provide us with enough details about the intended purpose of the question. For instance, in Figure 9 the purpose of question 18 is to identify whether a traceability tool allows participants to specify properties for a trace link such as name, type, usage, or other properties. The participant can choose either Yes or No. If a

participant selected the *Yes* option, his answer is correct but we do not know exactly whether the tool can allow him to specify one property or more than one. Also, we don't know which properties the participant means. Third, regarding the selection of the participants, we tried to eliminate the bias by contracting with a third-party website that can select them based on the specifications of our survey. However, after the survey was published online, the contracted website could not find enough participants for our survey claiming that the survey contains very technical questions that are hard to answer. We had to search through *LinkedIn* for potential participants and invite them individually to fill out the questionnaire. A bias could have possibly been introduced because we choose the participants. We do not claim this set of individuals is representative of a larger set of practitioners, and survey results must therefore be considered by caution as they may not generalize.

11.8.2 Remaining Threats to our Traceability Model

Our traceability model might fall short in capturing traceability information for the following situations. First, we use the URI to reference different types of artifacts. Although this can be applied to many artifacts, there are some cases in which the use of the URI is limited such as referencing a single activity within an activity diagram, or a code segment within a source code. Such cases are hard to assign a detailed enough *URI* so the linked artifact can be identified precisely (e.g., an activity diagram would be linked, not a specific activity in the diagram). Finding the right artifact may require parsing a document that represents an entire diagram (e.g., an XML document) or a large piece of source code. Second, the design of our traceability model does not consider the traceability of safety engineering requirements. For instance, we did not consider the traceability of hazards artifacts that could lead to a system failure. The identification of these artifacts requires a fault tree that can discover the events that could lead to hazards. Studying how and to what extent a fault tree model can be considered yet another heterogeneous model and its elements be linked to with our solution needs to be studied. Third,

regarding the automation of capturing traceability information, we are focusing on the design of the model classes that are necessary for capturing the traceability information of artifacts. Although the automated recording of traceability information is an important issue, our thesis focusses on modeling traceability information and doesn't include the automated capturing of traceability information, which is therefore outside the scope of our thesis. Literature discusses several algorithms for the automation of capturing trace information and its maintenance [5]. Fourth, although we built our traceability model to address drawbacks we identified in existing solutions, although we tested our solution on an industrial case study, we were unable to have practitioners in industry use our solution. Therefore, our model possibly could have some shortcomings that we are not aware of.

Finally, the definitions of heterogeneous and homogeneous artifacts and the differences between them has to be tuned and revisited. We believe the definition of homogeneous and heterogeneous artifacts is tool based on the heterogeneity of tools dependent as discussed by the authors of [21], [51], [87]. Our assumption about heterogeneous artifacts in this thesis is based on the heterogeneity of the modeling tools that produce artifacts of different formats. It is also a matter of heterogeneous domains. For instance, in cyber-physical systems, which require different domains of expertise to contribute to one system, which therefore require different modeling tools, notations, fields to coexist, heterogeneity is described as being a major challenge for the design and construction of trustworthy cyber-physical systems [13]. There are however situations where this assumption may not hold true. For instance, some modeling tools that support model-to-text transformations, such as Papyrus which can transform a UML diagram into an XML or Java code, may all be able to produce XML files which, from a syntactic point of view would be considered homogeneous. Therefore, it is not always easy to draw a line between heterogeneous and homogeneous artifacts.

11.8.3 Remaining Threats to our Taxonomy

Our trace links taxonomy was designed based on trace links classifications that we found in literature. There are many variations of trace links that either duplicated, inconsistent, or do not have specific usage. For instance, if two requirements are related to each other we can use either *Refine* or *Derive*; the two links have very close usage but not the same. There is no model exist in literature that attach a semantic to each link type that defines its usage. Our taxonomy consider capturing the semantic for trace links by providing a set of properties attached to every trace link that define its semantic. Therefore, we are concerned with modeling the infrastructure that facilitates the capturing of trace links semantics.

Our taxonomy integrates all trace links types into one place and eliminates the duplications and inconsistencies of trace links among all domains, however, there are situations in which we included two trace links that refer to the same meaning such as *Is-A* and *Generalization*, and *Contain* and *Has-A*, we consider these as remaining threats to the validity of our taxonomy. Our research is not concerned with specifying semantics for trace links, this requires an effort beyond our research scope. In addition, more work needs to be done in order to refine the trace links types, introduce new ones, or standardize them which requires also a devoted research effort. We consider these issues as remaining threats in our research.

11.8.4 Remaining Threats to the Literature Review

We conducted the traceability systematic literature review in order to ensure the coverage for important traceability articles. We searched are the IEEE Xplore, ACM Digital Library, Google Scholar, Science Direct, and Springer Link. Our search string is *Traceability AND (Heterogeneous OR Modeling OR Models OR MDE OR Model Driven OR Trace Link OR Requirement Engineering OR Systems Engineering OR Software Engineering)*. We believe there are two remaining threats with respect to the literature review: first, we could have included more terms that allow for more coverage of traceability articles. For instance, adding terms

such as: “Modelling”, “Requirement?”, and “RE”. Second, the search for articles in Google Scholar and Springer Link does not allow for a search in the *Abstract* which implies less accuracy in the results. We believe we should consider other digital libraries besides the one that we have listed.

11.8.5 Remaining Threats of the Case Study

Of course, one case study such as the one we describe in this thesis is not enough to be convincing about the applicability and validity of our solution. Although we made every effort to select a realistic case study, to exercise as many modeling elements of our solution as possible, we were not able, with one case study, to exercise everything (i.e., a Trace that has a sequence of trace elements) and one case study is not enough. More case studies must be used to model traceability in various contexts. Additionally, we only focused on the modeling aspects of traceability modeling, processing and management. More case studies are necessary to experiment with all aspects of traceability. Ideally, such case studies would come from the industry to lead to realistic results.

11.9 Summary

Considering threats to validity is very important when specifying the capabilities of our traceability framework. We consider the threats to validity throughout our research at different levels. Identifying the threats to validity at each step of our research can help us mitigate the threat to validity of the next step. Three types of threats to validity have been considered, the conclusion, internal and the external threats. The threats to conclusion validity are mitigated by making sure that the assumptions that we have in our experimentation are valid assumptions. We provide these assumptions based on the customer requirements, literature review, and the survey. The threats to internal validity can have impacts on our experimentation, while the external threats can have impacts on the traceability framework after its release. We identified the internal threats to the validity of our systematic literature review, traceability survey, traceability model, and trace links taxonomy. In addition, we talked about the threats to external

validity in terms of the use of the traceability model and the trace links taxonomy. The most important threat to the validity of our framework is how to cope with the future needs. We limited this threat by having a flexible design for the traceability model and the trace links taxonomy that should allow for adding any future functionalities without changing the design itself. Despite efforts to control or mitigate possible threats to validity, some threats remain, which we have discussed. Some of them should help pave the way for future research activities in the field.

12 Conclusion and Future Work

This research is concerned with finding a solution for the traceability needs during the engineering of systems that are realized through software and hardware solutions, and that include a wide range of disciplines. The solution should incorporate a traceability model that must be oblivious of the heterogeneity of the model elements to be traced. Moreover, we are concerned about building a trace links taxonomy, that can be configured with our traceability model, and which combines all types of trace links from the Requirements Engineering, Model Driven Engineering, and the Systems Engineering domains.

We argued that the traceability of heterogeneous artifacts is problematic for many reasons: first, traceable artifacts are generated by different modeling tools or have different formats; Second, traceable artifacts can exist at different phases or across different models or domains; Third, although some traceable artifacts can exist in one model, there are situations that require classifying the same artifact under different classifications. For instance, a state machine diagram can be classified as a UML diagram, a design artifact, or a state machine diagram. Fourth, as a result of artifacts heterogeneity various trace links classifications are required to serve this purpose and other purposes. For instance, stakeholders such as designers or analysts may have different interpretations for the same relation between the same two artifacts.

We established our research by proposing a set of research questions that can lead to an answer about our research problem. We followed a systematic approach in order to find answers for our research questions. First, we conducted a systematic literature review about traceability in order to a gain better understanding of available solutions. The review shows that research on modeling traceability of heterogeneous artifacts gains little attention, there are many useful articles that provide valuable information about traceability definitions and concepts, domain-specific traceability modeling, traceability tools, and trace links types and usage. These articles

are further investigated in order to gain some benefits from their proposed solutions. Furthermore, we analyzed these models by showing their benefits and the drawbacks. We found out that these models are either tailored to a specific platform or to a specific domain.

Second, we conducted a survey to get a feedback about the industrials practices with respect to traceability. The survey provides interesting results about traceability practices and traceability tools. We found that industry professional in traceability provide different opinions about their practices which is based on their special needs. However, the feedback provides us with insight about the different industrial traceability tools and their usages.

Fourth, we proposed a set of requirements for a generic traceability model. We based our requirements proposal on our need for a generic traceability model, and our finding from the systematic literature review and the survey feedback.

Fifth, we designed a generic traceability model based on the findings from the previous three steps. We benefit from the design of the existing traceability models that we found in literature, also we use the drawbacks of these models to improve our design.

Sixth, we validated our traceability model through many, complementary ways: a) validation by construction in which we justified the need for all classes, attributes, and associations, b) using small case studies from literature, c) introducing an industrial case study, and d) establish validation criteria that systematically illustrate the drawbacks of the existing traceability solutions. We showed that existing traceability models individually fail to satisfy almost 50% of the traceability requirements. The validation of our case study shows our model satisfies all the traceability needs for heterogeneous artifacts. The test results show that our traceability model can capture the traceability information of this case study. Moreover, introducing the *Characterization*, *Version*, and the *Constraint* classes in our design provides flexibility to our model to cope with future traceability needs without the need for changing the model design.

In order to mitigate threats to validity in our research, we devoted a complete chapter about threats to validity in which we consider all types of threats that our research might encounter beginning from the literature review, the survey, proposing the traceability model requirements, and the design of the traceability model and the trace links taxonomy.

Regarding the interoperability and portability of our model design, our models are not tied to a specific operating system or a tool. We used the USE case tool to build our model but it can be implemented by other tools. For instance, Papyrus modeling tool [150] can be used to model all the model classes and associations. The model classes can be transformed into Java or C++ classes. Moreover, the object model can be generated using the same tool, therefore, we avoid manual instantiation which is error prone. Our design for a trace links taxonomy complements the design of the traceability model by allowing users to reference any trace link based on trace links classifications. We implemented the taxonomy in plain English like text. We used the *Fluent editor* tool build the taxonomy by assigning a set of attributes for each trace link and specifying the relationships between the trace links. The tool allows for exporting and importing data with different formats such as RDF, XML, and OWL. We removed the duplications and inconstancies among trace links from the taxonomy, however, the taxonomy requires a review for the definitions and usages of many trace links.

Regarding the usability of our solution, as a future work, we are looking for integrating our model and the trace links taxonomy with other tools using the Lifecycle Collaboration (OSLC) service. In principle this is possible since OSLC specifications allow tools integration using linked data, and therefore seems to be an adequate technology to rely on for our solution. We have discussed linked data in chapter 8 during the discussion of building a trace links taxonomy. We will employ the same principle in integrating our traceability model with other development tools.

APPENDICES

APPENDIX A.
CUREB A&B: Research Ethics Protocol Form

Carleton UNIVERSITY
Canada's Capital University

RESEARCH ETHICS PROTOCOL SUBMISSION CHECKLIST

Please direct all questions regarding this checklist to the Research Compliance Office at: ethics@carleton.ca

Complete the checklist below to ensure important elements of your research ethics application are not overlooked. Please include the checklist as a cover page to your protocol. This will lead to a faster turnaround for ethics review and clearance.

☒	**Include all information requested in CUREB Submission form** (ensure track changes have been removed and spelling/grammar has been checked. Please combine the protocol and all attachments in a single PDF).
☒ ☐ **N.A.**	**For student researchers: ensure supervisor provides approval in one of the following ways:** • The Supervisor Signature Form is signed and attached with the application. • The supervisor is included as the PI on the application and submits in the system. • If a supervisor is unable to complete one of the above processes, an email to ethics@carleton.ca will be accepted as confirmation of supervisor approval.
☒ ☐ **N.A.**	**Copies of all written communications** (e.g. recruitment materials, information forms, informed consent forms, debriefing form) to participants must be on Department/Faculty letterhead
	Recruitment Materials
☐ ☒ **N.A.**	Script(s) – in-person, telephone, 3rd party, email, etc.
☐ ☒ **N.A.**	Invitation to participate
☐ ☒ **N.A.**	Advertisement, poster, flyer
☐ ☒ **N.A.**	None required – explanation provided
	Data Collection Methods Checklist
☐ ☒ **N.A.**	Standardized Instrument(s)
☒ ☐ **N.A.**	Survey(s), Questionnaire(s)
☐ ☒ **N.A.**	Interview and/or Focus Group Questions
	Confidentiality Agreement
☐ ☒ **N.A.**	Other (e.g. describe biomedical or prototype materials used in the experiment in detail and/or include a photo of these materials).
	Free and Informed Consent (instructions here)
☒ ☐ **N.A.**	Consent and Assent Form(s) – include forms for all participant groups and data

☐ ☐ **N.A.** ☐ ☒ **N.A.**	gathering methods. Letter(s) of Information for Implied Consent Verbal Consent and Assent Scripts
☐ ☐ **N.A.**	**Debriefing Materials**
☐ ☐ **N.A.** ☐ ☐ **N.A.**	**Are all recordings (videos, audio, photo) used in the study adequately explained in the protocol?** Permission obtained for each recording?
☐ ☒ **N.A.**	**Permission obtained to access confidential documents or materials?**
☐ ☒ **N.A.**	**If using deception, participants must be debriefed. They must also have the opportunity to withdraw their data (a follow-up consent form should be given so participants may consent to the use of data in cases of deception).**
☐ ☒ **N.A.**	**Other Approvals** **(e.g.** Biosafety committee approval, site permissions to recruit/conduct research on properties, at organizations, and at institutions, etc.).

Carleton UNIVERSITY
Canada's Capital University

1. Project Team

1A. Lead Researcher (Detailed instructions)

	Academic Staff
	Library or Other Staff
	Post-doctoral Fellow
x	Ph.D. Student
	Master's Student
	Undergraduate
	Other

Last name/First name

Response: Mustafa/ Nasser

Official university (or other institution) email address:

Response:nassermustafa@cmail.carleton.ca

Indicate your department, faculty and institution

Response: Systems and Computer Engineering, Faculty of Engineering and Design-Carleton University

1B. Academic Supervisor (Detailed instructions)

	Same as lead researcher

Academic supervisor(s) Last name/First name. (Note, the supervisor must be copied on all correspondence with CUREB.)

Response: Labiche/Yvan

Official university (or other institution) email address:

Response: yvan.labiche@carleton.ca

Indicate your department, faculty and institution

Response: Systems and Computer Engineering, Faculty of Engineering and Design-Carleton University

1C. Project Team Members (Detailed instructions)

x	Not applicable/No other team members

List the project team members here. For each team member, provide the following: 1) Last name/First name 2) Email address 3) Role in project 4) Department and institution (E.g. Master's student in Canadian Studies at Carleton)

Response: Not applicable/No other team members

1D. Researcher Training (Detailed instructions)

x	No training provided/Not applicable
	Researcher will be trained

Describe any additional training the researcher(s) have (or will receive) to work with the participants.

Response: No training is needed

	Researcher is trained

2. Study Overview

2A. Project Title (Detailed instructions)

Title of Research Project

Response: Traceability Modeling for the Engineering of Heterogeneous Systems

2B. Study Goal (Detailed instructions)

What question will your research answer (1-2 sentences)?

Response: Enhance and validate the design of a generic traceability model by identifying (a) traceability model requirements, (b) artifacts and trace links types, and (c) shortcoming of existing traceability models.

2C. Study Purpose and Benefits (Detailed instructions)

Study rationale: why should the research be pursued; what are the benefits, and to whom? (Benefits can be to research community, companies, or society in general.)

Response: Modeling traceability is a corner stone in Software and Systems Engineering especially in developing critical systems. It can prevent catastrophic loss by validating and verifying system's requirements. These benefits are reflected on researchers, software companies, and society.

2D. Dates of Recruitment/Participant Interaction (Detailed instructions)

	Not applicable/Secondary Data

When will you start recruiting participants? (DD/MM/YYYY)

Response:25/01/2018

When will you stop interacting with participants? (DD/MM/YYYY)

Response:25/03/2018

2E. New or Previously Recorded Data (Detailed instructions)

x	Primary Analysis (New Data are Collected)
	Secondary Analysis of Directly Identifying Data
	Secondary Analysis of

Does this research collect new data or analyse previously collected data (secondary analysis)? If the research involves secondary analysis, describe the coding of personal identifiers and indirect identifiers within data. Note: Tri-Council has defined anonymized data to be data irrevocably stripped of direct identifiers (a code is not kept to allow future re-linkage).

		Coded or De-identified Data	Response: Although some data are collected from literature review, this is the first time we collect data from participants.

2F.	Additional Reviews (Detailed instructions)		No additional review	Has this project been reviewed for academic merit (not required, but for the Board's information). By whom? As part of a tri-council grant application or student's thesis committee? Response: This is a Ph.D. project proposed by my supervisor.
		x	Departmental review	
			Grant council review	

3. Funding and Approvals

3A.	Project Funding (Detailed instructions)		Unfunded	Who is funding this project? If applicable, include the funding source/agency/company, program, award name, and number (from CUResearch) Response: This project was originally funded through an NSERC Collaborative and Research grant that has ended two years ago.
			Tri-Council Funded	
		x	Other Award/Grant	
			Contract Funded	
			Personal Consulting or Personal Work	
			Scholarship	

3B.	Researcher Funding (for research contracts and personal consulting only) (Detailed instructions)	X	Not applicable/Not contract funded research	For research contracts and personal consulting only: Is there a real or perceived conflict of interest and how will it be managed? How much funding (dollar amount and the percentage of the total) will the researcher(s) receive as income? Provide the title and date of any contracts. (The REB may review the contract.) Response: Not applicable/Not contract funded research
			No funds are paid directly to the researcher as personal income	
			The researcher will receive a portion of the funds as personal income	
			A copy of the contract/agreement has been submitted to the Research Compliance Office	

3C.	Minimal Risk Review Request		No	Would you like to request this protocol be considered for minimal risk review?

	(Detailed instructions)	X	Yes	If so, please briefly justify. If not requesting a minimal risk review, leave this section blank. (The REB will use this information to make a decision as to whether this application will be reviewed at full board or via a delegated process). Response: The survey will be published online and the responses will be gathered anonymously from software Engineers who works in this domain and are interested in improving traceability in their workplace. The Participants' demographic information will not be used or published in any medial.
3D.	Additional Approvals Required (Detailed instructions)		Not applicable/No other approvals required	Is organizational permission required to conduct research (e.g., schools, employers, other universities, correctional services, aboriginal communities, and other data collection locations)? If conducting research in another country, is local permission required? Indicate if permission has been secured and provide a copy of the permission. Research with biohazards or animals must also secure approval from the appropriate committee at Carleton University. Response: Carleton University ethic committee
		X	Organizational Permission	
			Visa/Travel Permits	
			Other REBs or Institutional Approvals	
			Biohazards	
			Animal Care Committee	
			Other (please specify)	
			Permission letters attached	
			Letters to follow	
4.	Methods: Participants			
4A.	Participant Interactions Overview (Detailed instructions)		Directly interacting with participants	Briefly list what will happen to, or will be required of, the participants during the course of the research. (Only a project overview is requested here; methodology details are required in the first question of each section). If the research involves secondary analysis of data that has already been collected, the REB needs information about the
		X	Interacting with participants online (e.g. online surveys)	

	Observing participants	original data collection to be confident data were collected ethically.
	Secondary Analysis of Data	Response: The survey will be published on the website: https://www.surveymonkey.com. This is a host website in which invitations are sent through the website. The method of sending invitations is not known to the researcher. Interested participants can view the survey online, answer the questions, and save their responses. Results are saved at a database belongs to the website, however, the researcher can access to the data belongs to his survey through a secure password. The expected time to fill the survey is 10 minutes, and the survey will be kept online for 2 months from the day of publishing it online. I am expecting to publish it on the 25th of January, 2018 until 25th of March 2018.
	Other	

4B. Description of Participants (Detailed instructions)		Describe the participants and any inclusion criteria. If applicable, describe any exclusion criteria. If using a separate sample of control participants, describe this group.
		Response: Interested participants who apply traceability in their workplace. Of course, the survey targets English speaking participants. Normally, they have minimum a college degree with experience in their field.
4C. Number of Participants (Detailed instructions)		What is the number of participants requested? If multiple groups of participants are involved, breakdown by participant type. Provide a justification including a statistical rationale if appropriate.
		Response: There is no limit on the number of participants

4D. Vulnerable Population (Detailed instructions)	x	Not Vulnerable Population	Describe any pre-existing vulnerabilities associated with the proposed participant group(s) that may cause additional risks. Describe the associated risks and mitigation strategy.
		Vulnerable Population	

			Response: Not Vulnerable Population, We believe there is no risk of conducting this survey
4E.	Participant Relationship to Researcher (Detailed instructions)	X	No previous relationship
			Instructor-Student
			Client
			Employee
			Friends/Family
			Other
			Describe any relationship that exists between the participants and the research team or any recruiting party or sponsor. Indicate how relationships will be managed so there is no undue pressure put on participants.
			Response: No relationship at all
4F.	Conflict of Interest (Detailed instructions)	X	No conflicts
			Financial
			Commercial Entity Benefits
			Other
			Describe any real or perceived conflicts of interest for any research team member that could affect participant welfare. If so, describe it here and indicate how it will be managed.
			Response: Participants are unknown to the researcher
5.	Methods: Recruitment		
5A.	Recruitment Methods (Detailed instructions)		Not applicable
			Posters
			Social Media
		x	Online Panels (e.g. Qualtrics)
			Student Participant Pool (e.g. SONA)
			Emails
			Letters
			Telephone
			Snowballing
			Other
			Describe each step of how participants will be recruited. This includes how contact information is obtained, how participants will be made aware of the study, where will recruitment materials be located, and how participants can express their interest. Provide a copy of the recruitment material(s) including any oral scripts, recruitment posters, recruitment emails, social media postings etc.
			Response: the survey will be published on https://www.surveymonkey.com. This is a known website for publishing surveys for different purposes. The internal details of recruitment is not known to the researcher. The researcher doesn't have any information about the participants. Survey Monkey Contribute, our proprietary online panel is dedicated solely to supporting customers seeking insights from respondents in the United States. We recruit people to Survey Monkey Contribute through a

variety of means, the primary method of recruitment being Survey Monkey survey respondents. Over 30 million unique respondents answer Survey Monkey surveys sent out by our subscribers each month.

We have also integrated a variety of third party sample partners into our Audience platform. Third party sample partners fulfill customers' needs that our internal sources cannot fulfill mainly international sample.

5B. Location of Recruitment (Detailed instructions)

x	Not applicable
	Carleton
	Other Canadian School/University
	Canada
	Online
	Other

List all recruitment locations. If some locations require permission prior to recruitment, indicate if permission has been secured.

Response: This an online Survey, it is not limited to a specific region or place

5C. Third Parties in Recruitment (Detailed instructions)

	Not applicable
X	Third Parties

If using third parties to recruit, indicate who is doing the recruitment and how it will be accomplished. Does the third party have contact information for the participants? If not, how will it be acquired?

Response: The researcher can choose a website among many who offer online survey publishing.

5D. Recruitment risks to Participants (Detailed instructions)

X	No risks / Not applicable
	Mild risks
	Moderate risks
	High risks

Describe any risks to participants during the recruitment phase.

Response: We believe no risk is involved

5E. Recruitment risks to Researcher (Detailed instructions)

x	No risks / Not applicable
	Mild risks
	Moderate risks
	High risks

Describe any risks to the research team during the recruitment phase.

Response: We believe no risk is involved

5F.	Benefits (Detailed instructions)	X	No Direct Benefits to Participants / Not applicable	Describe any direct benefits to the research participants as opposed to society or knowledge. Response: There is no direct benefit for the participants other than knowledge.
			Direct Benefits to Participants	
5G.	Compensation (Detailed instructions)	X	No Compensation/Not applicable	Describe all compensation/remuneration and indicate when participants will receive the compensation. What is the monetary value of the compensation/remuneration? What happens to the compensation if a participant withdraws? Response: No Compensation
			Money / Gift Card	
			Reimbursement of Travel Expenses	
			Refreshments	
			Course Credit	
			Other	
6.	Methods: Informed Consent			
6A.	Obtaining informed consent (Detailed instructions)		Signed consent	Describe the method for obtaining informed consent from the participants. If signed consent is not used, justify the alternative method chosen. Include a copy of the consent materials. Response: Online consent
		X	Online consent	
			Oral consent	
			Implied consent	
			Parent/Guardian consent	
			Assent	
			Other	
6B.	Deception (Detailed instructions)	X	Full Disclosure (i.e. no deception)	Describe the deception and/or partial disclosure (e.g. what information is withheld). Why must it be used and why not an alternative research method? Describe the magnitude and likelihood of harm. Deception requires debriefing and secondary consent forms. The secondary consent form allows the participant to consent to the use of their data when they have been informed of the true nature of the study. Partial disclosure requires a debriefing form. Response: The collected data will not be used other than improving the quality of the research.
			Partial Disclosure	
			Mild Deception	
			More than Mild Deception	

6C. Debriefing (Detailed instructions)

X	Not applicable/not required
	Participants will be debriefed

Will participants be debriefed? If this is the case, describe when and how participants will be debriefed. (Include a copy of any documents that will be provided to participants). Describe any risks during debriefing and how they will be mitigated. According to Tri-Council, debriefing is required in all cases of deception: http://www.pre.ethics.gc.ca/eng/policy-politique/initiatives/tcps2-eptc2/chapter3-chapitre3/#toc03-1b

Response: Not applicable/not required

6D. Withdrawal Procedures (Detailed instructions)

	Not applicable
X	Participants can withdraw
	Participants can only withdraw during the study session
	Special withdrawal procedures

Describe the procedures for a participant to withdraw. What will happen to data from participants who withdraw? Describe any deadlines and limitations on withdrawal.

Response: This is an online survey, unless a participant saves his/her work, s/he can withdraw at any time by logging out of his/her session. However, if a participant wishes to withdraw after submitting his/her answers, s/he can send an email to the website admin to be removed from the survey.
Please see the attached response letter for more details.

7. Methods: Data Collection

7A. Data Collection Methods (Detailed instructions)

X	Questionnaires / Surveys
	Interviews
	Focus Groups
	Oral and/or Visual Stimuli
	Equipment and/or software testing
	Other

Describe the method of data collection being used and provide details of any instruments used. Breakdown by phases, participant groups, or types if required. If data collection is being done online, visit the detailed instructions for full details on what information the REB requires. (CUREB requires a copy of any questionnaires, surveys, or interview guides).

Response: The website offers some tools that can provide the researcher with different statistics about the survey findings. However, the researcher can also perform other statistics and draw graphs based on his needs.

7B. Location of Data Collection (Detailed instructions)

	Carleton
	Canada (other than Carleton)
	Workplace
	Public venue
X	Online
	Other

Where will the participant be during data collection?

Response: Online, globally

7C. Photography or Recordings (Detailed instructions)

X	Not applicable
	Photographs
	Audio Recording
	Video Recording

If the participant will be photographed, video-recorded or audio-recorded, indicate how the data will be acquired and protected (if applicable).

Response: Not applicable

7D. Translation or Transcription (Detailed instructions)

X	Not applicable
	Translation
	Transcription
	Researcher will translate or transcribe

If you require the services of a translator or transcriber, describe what services you will use and how you will interact with the translator and/or transcriber. If a confidentiality agreement will be used, include a copy.

Response: the survey targets English speakers

7E. Online data collection (Detailed instructions)

	Not applicable
	Carleton-based server
	Commercial server (based in Canada)
X	Commercial server (outside Canada)
	Other

Describe the technology platform used to collect online data. Describe the security of the data. Will participant IP addresses be recorded? Are there any special limits to privacy?

Response: The survey will be published on https://www.surveymonkey.com. This This is supposed to be a secure host. The collected data will be emailed to the researcher. In addition, the website offers web based statistical tools for generating charts and reports.

7F. Bio-interactions (Detailed instructions)

X	Not applicable
	Biological specimens/fluids

Describe the apparatus and methods to acquire biological specimens or fluids. (e.g., blood, saliva, tissue samples.) How will specimens be safely stored and destroyed? If any will be kept, explain why, how and for how long.

Response: Not applicable

7G.	Bio-instruments (Detailed instructions)	X	Not applicable	Bio-instruments touch or send energy into the body. (e.g., electrodes, MRI/X-ray.) Describe the apparatus and its use. If applicable, explain any significant risks and compare the dose (e.g., electrical, radiation) to established safety standards. Response: Not applicable
			Bio-instruments	
7H.	Bio-interventions (Detailed instructions)	X	Not applicable	Describe the apparatus and methods associated with the bio-intervention. (e.g., drug, stress, medical devices.) Explain any risks to the participants and compare it to established safety standards. Response: Not applicable
			Bio-interventions	
7I.	Risk of Psychological Harm (Detailed instructions)	X	Not applicable/No risks	Explain the rationale for your selection, and, if applicable, explain the nature, magnitude and probability of the risks and how they will be mitigated. Response: Not applicable/No risks
			Mild risks	
			Moderate risks	
			High risks	
7J.	Risk of Physical Harm (Detailed instructions)	X	Not applicable/No risks	Explain the rationale for your selection, and, if applicable, explain the nature, magnitude and probability of the risks. Describe how they will be mitigated. Response: Not applicable/No risks
			Mild risks	
			Moderate risks	
			High risks	
7K.	Risk of Social and/or Economic Harm (Detailed instructions)	X	Not applicable/No risks	Explain the rationale for your selection, and, if applicable, explain the nature, magnitude and probability of the risks and how they will be mitigated. Response: Not applicable/No risks
			Mild risks	
			Moderate risks	
			High risks	
7L.	Incidental Findings (Detailed instructions)	X	Not applicable/No incidental findings anticipated	Describe possible incidental findings (unanticipated discoveries that relate to the welfare of participants or others) and how they will be managed. Examples are becoming aware of abuse of a child, or imminent harm to a participant or third party. Your approach to managing any findings should also be described in the informed consent.
			Low probability	
			High probability	

	Response: Not applicable/No incidental findings anticipated

8. Methods: Data Storage and Analysis

8A. Identifiability of stored data (Detailed instructions)

X	Anonymous
	Pseudonyms/Coded
	Real participant names with data attributable
	Real participant names with data non-attributable
	Different levels of anonymity for different groups of participants

Describe the identifiability of research data, including how pseudonyms will be assigned, if applicable. If there are different levels of anonymity for different groups, describe each level here.

Response: Participant's emails are optional in the survey. The researcher has no other contact information nor sustain a list for the participants. This is totally managed by the website admin.
Please be advised that the researcher is going to download the survey results on his own personal computer in order to perform further analysis

8B. Identifiability of published data (Detailed instructions)

X	Anonymous
	Aggregate data only
	Pseudonyms/Coded
	Real participant names with data attributable
	Real participant names with data non-attributable
	Different levels of anonymity for different groups of participants

Describe the identifiability of data that will appear in publications, including how pseudonyms will be assigned, if applicable. If there are different levels of anonymity for different groups, describe each level here.

Response: Anonymous

8C. Data Storage (during the project) (Detailed instructions)

	Encrypted
X	Password-protected
	Anonymous data
	Physical documents
	Other

How are data being stored and kept safe? Provide details for electronic data and hard copies if applicable.

Response: The data will be stored in the website database and access by the researcher by an assigned username and password.

8D. Data Disposition (after the project) (Detailed instructions)

	Retained by the researcher(s)
	Stored in a depository
	Archived
	Shared with research agreement
X	Shared publicly
	Returned
	Destroyed

After project completion, describe how the data (confidential and non-confidential aspects) will be stored for future use, made publicly available, archived, returned to participants, or destroyed. If shared, with whom? Describe any restrictions on access. If destroyed, how long will data be kept? Will personal identifiers and the actual data be destroyed at different phases? Will participant contact information be kept for future studies?

Response: A summary of the findings well be published.

8E. Data Breach Risks (Detailed instructions)

X	Mild risk to participants
	Moderate risk to participants
	High risk to participants

Describe how likely a data breach is to occur and how it could affect the participants. If risks are significant, how will they be mitigated?

Response: The survey will be published on https://www.surveymonkey.com. This is a known secure website for publishing surveys. It is unlikely to breach the data. As far as I know, the only way to breach the data is through hacking the website which cannot be guaranteed. However, breaching the data have no impact on the participants.

9. Declarations

9A. Supervisor Approval

	Not applicable
X	Supervisor Approved
	Supervisor has not approved

For student projects, please indicate the date that the supervisor approved the application to go forward for REB approval. (All CUREB-A applications must copy the supervisor when submitting an application to ethics@carleton.ca. CUREB-B applications are automatically submitted to the supervisor through the online application system; therefore, the student must inform his/her supervisor that the application has been submitted so the protocol may be approved by the supervisor and received by CUREB-B).

Response: this is a CUREB-B application

9B.	Declaration #1 (Detailed instructions)	X	I agree	This ethics application accurately describes the research project or scholarly activity that I plan to conduct.
9C.	Declaration #2 (Detailed instructions)	X	I agree	No recruitment or data collection for this protocol will commence before ethics clearance.
9D.	Declaration #3 (Detailed instructions)	X	I agree	No changes will be made to the research project as described in this protocol without receiving clearance from the Research Ethics Board.
9E.	Declaration #4 (Detailed instructions)	X	I agree	The Research Ethics Board will be notified immediately of any alleged or real ethical breaches or concerns, adverse events, or participant complaints that arise during or after the course of this research project.

10.	Comments	
10A.	Comments (optional)	Do you have any comments or suggestions on the form Response: no comments

Bottom of Form

Carleton UNIVERSITY
Canada's Capital University

Systems and Computer Engineering
4456 Mackenzie Engineering Building
1125 Colonel By Drive
Ottawa, Canada, K1S 5B6
Tel: (613) 520-5740

Consent Letter

Dear Participant,

I invite you to participate in a research study entitled: Traceability Modeling for the Engineering of Heterogeneous Systems. Please don't proceed to the survey if you never experienced projects that require traceability.
I am currently enrolled in the PhD program at the department of Systems and Computer Engineering at Carleton University in Ottawa, Canada. The purpose of my research is to determine whether the traceability model that I design can accommodate traceability of heterogeneous artifacts. Particularly, artifacts generated during Systems Engineering projects development.
The enclosed questionnaire has been designed to collect information about the use of traceability in Systems Engineering and Software Engineering projects.
Your participation in this survey is completely voluntary. There are no known risks to participation beyond those encountered in everyday life. Your responses will remain confidential and anonymous. Data from this survey will be kept secure, no one other than the researchers will know your individual answers to this questionnaire. Please be advised that your withdrawal from the survey is allowed in two cases: first, if you close the browser while you are in the first page of the survey, in this case your data will not be saved. Second, if you pass the first page of the survey, simply answer the last question with "NO" then submit your answers, in this case we will apply a filter to exclude the data of all participants who answered "NO" to that question. There is no need to email us in both cases.
If you agree to participate in this survey, please answer the questions on the questionnaire as best as you can. It should take approximately 10 minutes to complete your answers. Please submit the questionnaire through the host website.
If you have any questions about this project, feel free to contact me by email: Nasser.mustafa@carleton.ca, or contact my supervisor Dr. Yvan Labiche, Associate Professor in the department of Systems and Computer Engineering at Carleton University by email : Yvan.labiche@carleton.ca.
Information on the rights of human subjects in research is available through the Human Research ethics at Carleton University, 1125 Colonel By Drive Ottawa, ON, K1S 5B6 ; website: https://carleton.ca/humanresearchethics/; Gurban DuVAl, Phone: +1613-520-2600X4585
Thank you for your assistance in this important endeavor.

Sincerely yours,
Nasser Mustafa

Carleton UNIVERSITY
Canada's Capital University

Systems and Computer Engineering
4456 Mackenzie Engineering Building
1125 Colonel By Drive
Ottawa, Canada, K1S 5B6
Tel: (613) 520-5740

Traceability in Systems and Software Engineering

Purpose of the survey

Although traceability benefits are proven in Systems and Software Engineering, it is not widely adopted by developers. There are many reasons that prevent adopting traceability such as: the high cost of specifying and maintaining traceability information, the nature of the applications under development, and the complexity of applying traceability. However, traceability is mandated in safety critical systems such as aviation, automobile, and nuclear power plants.

The purpose of this survey is to collect data about the traceability of heterogeneous artifacts in Systems and Software Engineering in which many artifacts are generated during systems development. These artifacts are heterogeneous in nature since they are generated by different modeling tools and come from different domains of expertise.

The collected data will be analysed to answer the following research questions:

- What are the requirements of a generic traceability model?
- What are the possible types of traceable artifacts?
- How can we improve the identification of trace links between artifacts?
- What are the shortcomings of existing traceability tools?
- What features should a generic traceability model have?
- How to validate a generic traceability model?

A Questionnaire about Modeling Traceability in Systems and Software Engineering

1. Educational background?
   A. Bachelor
   B. Masters
   C. PhD
   D. Other
2. Industrial experience?
   A. Less than 5 years
   B. 5-10 years
   C. 11-20 years
   D. More than 20 years
3. Occupational role?
   A. Company CEO
   B. Project manager
   C. Supervisor
   D. Developer
   E. Other
4. Domain of expertise
   A. Analyst
   B. Designer
   C. Tester
   D. Programmer
   E. Other
5. Company domain
   A. Software development
   B. Hardware development
   C. A and B
   D. Other
6. Company location (country)
7. Company size
   A. Less than 10 employees
   B. Between 10-100
   C. Between 101-1000
   D. More than 1000
8. Why your company apply traceability?
   A. Comply Regulations
   B. Change impact analysis
   C. System validation and verification
   D. Other
9. In which domain traceability is adopted?
   A. Requirement Engineering
   B. Model Driven Engineering

C. Systems Engineering
D. All of the above

10. Please specify the type of artifacts that you trace?
11. Where do the traced artifacts exist?
    A. One model in a single phase/level
    B. One model different phases/levels
    C. Across different models and at the same phase/level
    D. Across different models and at the same phase/level
12. Please specify the direction in which you trace artifacts?
    A. One direction
    B. Bidirectional
13. Please specify the cardinality of the traced artifacts?
    A. One SOURCE artifact to one TARGET artifact
    B. One SOURCE artifact to many Target artifacts
    C. Many SOURE artifacts to one TARGET artifact
    D. Many Sources to many Targets artifacts
14. How do you identify the trace links between the traced artifacts
    A. Rely on logs
    B. Trace links taxonomy
    C. My Own Judgment
15. Please specify the name of the traceability tool that you use at work
16. Is the tool platform dependent (e.g., works only under certain operating systems)
    A. Yes
    B. No
17. Can the tool allow you to specify some constraints on a certain trace link or an artifact?
    A. Yes
    B. No
18. Can the tool allows you to specify some information about a trace link such as name, type, etc.?
    A. Yes
    B. No
19. Can the tool allow you to visualize traceability information in
    A. Textual Format
    B. Graphical format
    C. Textual and graphical format
    D. Tabular format
20. Can the tool allows you to specify information about an artifact such as type, version, etc.?
    A. Yes
    B. No

21. Can the tool allow you to specify which artifact is a SOURCE and which artifact is a TARGET
    A. Yes
    B. No
22. Can you specify a source and a target artifacts of different levels of granularity?
    A. Yes
    B. No
23. Can the tool be customized to add more traceability features if needed?
    A. Yes
    B. No
24. Would you be interested in participating of testing a new traceability model?
    A. Yes
    B. No
25. Please write any comment that you would like to add about this survey
26. Do you want to withdraw from the Survey?
    A. Yes
    B. No

APPENDIX B.
Traceability Model Specifications

-- Classes

```
abstract class TraceElement
attributes
uri: String
end

class TraceabilityRoot
 attributes
 name : String
end

class Constraint
attributes
  type : String
  value : String
  language: String
end

class Characterization
attributes
  type : String
  domain :String
  granularity : String
end

class Version
attributes
owner: String
modifiedBy: String
modifiedOn: String
time: String
baseline: String
end

class Trace < TraceElement
end

class Artifact < TraceElement
```

```
end
class TraceLink < TraceElement

end
```

-- Associations

```
composition traceElement between
  TraceabilityRoot [1]
  TraceElement[1..*]
end

aggregation orderedElements between
Trace[0..1]
TraceElement[0..*]
End

aggregation constraint between
  TraceabilityRoot[0..1]
  Constraint [0..*]
end

composition version between
  TraceElement[1..*]
  Version [0..*]
end

association target  between
TraceLink[0..*] role linkEnd
Artifact [1..*] role target

end
association source  between
 Artifact [1..*] role source
 TraceLink[0..*] role linkStart

end
composition description between
  TraceElement[1..*]
  Characterization [0..*]
```

end

```
constraints
context Trace inv:
---- Iterate through the sequence of all trace elements and check  if the trace element is of
----type artifact and the trace element at position i is an artifact
```

```
if self.orderedElements->forAll(e|e.oclIsKindOf(Artifact)) then
Sequence{1..self.orderedElements->size()-1}->iterate(i : Integer; b : Boolean = true | b and
self.orderedElements->at(i).oclAsType(Artifact).
------ then the target of the TraceLink at position (i+1) must be an artifact
sourceTraceLinks.targetArtifacts->includes(self.orderedElements->at(i+1)))
---- otherwise,  if the trace element in the sequence is of type  TraceLink then the source and -
-----target shouldn't be empty and must be of type artifacts.
else if self.orderedElements->forAll(e|e.oclIsKindOf(TraceLink)) then
Sequence{1..self.orderedElements->size()-1}->iterate(i : Integer; b : Boolean = true | b and
self.orderedElements->at(i).oclAsType(TraceLink).targetArtifacts->intersection(
self.orderedElements-> at(i+1).oclAsType(TraceLink).sourceArtifacts)->notEmpty())
endif
endif
```

Read Me –Traceability Model

In order to visualize the Traceability model in USE tool, follow the following steps:

1. Install the USE on your computer. For Installation and tutorial of USE, follow this Link: https://sourceforge.net/projects/useocl/
2. Create new file in Word Pad.
3. Copy and paste the specifications of the Traceability model to the newly created file.
4. Save the file in a computer directory.
5. Open the USE tool (the .exe file is inside the bin directory of the tool folder)
6. From the *File* menu in the Menu bar click on *Open specification* and choose the file that you created.
7. To view the Class diagram from the *View* menu bar click on Create *View and choose Class* Diagram

Specifications of the Trace Links Taxonomy

Namespace:
'http://www.ontorion.com/ontologies/Ontology92f6fe28b5854078a984b0607d68f51e'.

Every mde-link is a trace-link.
Every re-link is a trace-link.
Every se-link is a trace-link.
Every sysMl-link is a trace-link
Every process-related-link is an re-link.
Every product-related-link is an re-link.
Every satisfaction is a product-related-link.
Every dependency is a product-related-link.
Every rationale is a process-related-link.
Every evolution is a process-related-link.
Every contribution is an re-link.
Every evolution is an re-link.
Every dependency is a re-link.
Every generalization is an re-link.
Every satisfy is an re-link.
Every overlap is an re-link.
Every conflict is an re-link.
Every rationale is an re-link.
Refine is an re-link.
satisfy is a satisfaction.
Interface-With is a satisfaction.
Defined-By is a satisfaction.
Created-By is a satisfaction.
Allocated-To is a satisfaction.
Derived is a satisfaction.
Temporal is a dependency.
Depend-On is a dependency.
Trace-To is a dependency.
Import is a dependency.
Export is a dependency.
Usage is a dependency.
Contain is a dependency.
Used-by is a dependency.
Performed-by is a dependency.
Evolve-To is an evolution.
Modify is an evolution.
Define is an evolution.
Elaborate is an evolution.
Derive is an evolution.
Replace is an evolution.
Based-on is an evolution.
Frmalize is an evolution.

Elaborate is an evolution.
Contain is a generalization.
Every implicit is an mde-link.
Every explicit is an mde-link.
Every model-to-model is an explicit.
Every model-to-non-model-artifact is an explicit.
Every static is a model-to-model.
Every dynamic is a model-to-model.
consistent-with is a static.
dependency is a static.
Call is a dynamic.
Notify is a dynamic.
Generate is a dynamic.
Synch –with is a dynamic.
Satisfy is a model-to-non-model-artifact.
Allocated-To is a model-to-non-model-artifact.
Explain is a model-to-non-model-artifact.
Perform is a model-to-non-model-artifact.
Support is a model-to-non-model-artifact.
Update is an implicit.
Query is an implicit.
Creation is an implicit.
Composition is an implicit.
Deletion is an implicit.
Model-to-Model is an implicit.
Model-to-Text is an implicit.
Generated-By is a rationale.
Resolve is a rationale.
Supported-By is a rationale.
Affect is a rationale.
Based-On is a conflict.
Affect is a conflict.
Resolve is a conflict.
Generate is a conflict.
Inconsistency is a conflict.
Derive is a satisfy.
Refine is a satisfy.
Establish is a satisfy.
Contribute is a satisfy.
Adapt is an overlap.
Non-Causal-Conformance is an evolution.
Temporal is an evolution.
Refine is a dependency.
Require-Features-In is a dependency.
Developmental is a dependency.
Causal-Dependency-Conformance is a dependency.
Correspondence is a dependency.
temporal is an se-link.
directional is a se-link.
horizontal is a directional

vertical is a directional.
micro-Inter-Vertical is vertical.
macro-Intra-Vertical is vertical.
macro-Inter-Vertical is vertical.
micro-Intra-Vertical is vertical.

micro-Inter-Horizontal is horizontal.
micro-Intra-Horizontal is horizontal.
macro-Inter-Horizontal is horizontal.
macro-Intra-Horizontal is horizontal.
req-to-Req is a sysml.
req-Model-Element-To-Req is a sysml.
design-Element-To-Req is a sysml.
test-Element-To-Req is a sysml.
Derive is a req-To-Req.
Refine is a req-Model-Element-To-Req.
Satisfy is a design-Element-To-Req.
Verify is a test-Element-To-Req.
Every trace-link has-usage nothing-but (some string value).
Every trace-link has-definition nothing-but (some string value).
Every trace-link has-name nothing-but (some string value).
Dependency has-name equal-to 'depends-on'.
Dependency has-definition equal-to 'link between 2 artifacts where the existence of one depends on the existence of the other'.
Dependency has-usage equal-to 'links 2 co-existed artifacts'.

Read Me- Trace Links Taxonomy

In order to visualize the trace links taxonomy in Fluent Editor follow the following steps:

8. Install the Fluent Editor in your computer. For Installation and tutorial of Fluent Editor follow the Link: http://www.cognitum.eu/semantics/examples/
9. Create new file in Fluent Editor
10. Copy and paste the specifications of the trace links taxonomy to the newly created file.
11. Run the file to visualize the taxonomy.

13 References

[1] S. Winkler and J. Pilgrim, “A survey of traceability in requirements engineering and model-driven development,” *Softw. Syst. Model.*, vol. 9, no. 4, pp. 529–565, 2010.

[2] J. P. A. Almeida, M.-E. Iacob, and P. van Eck, “Requirements traceability in model-driven development: Applying model and transformation conformance,” *Inf. Syst. Front.*, vol. 9, no. 4, pp. 327–342, 2007.

[3] R. France, “Model-driven Development of Complex Software: A Research Roadmap.,” in *Future of Software Engineering*, 2007, pp. 37–54.

[4] J. Fraser and A. Gosavi, “What is systems engineering.,” *Am. Soc. Eng. Educ.* , 2010.

[5] O. Gotel *et al.*, “The quest for Ubiquity: A roadmap for software and systems traceability research,” in *2012 20th IEEE International Requirements Engineering Conference, RE 2012 - Proceedings*, 2012, pp. 71–80.

[6] J. C. Sampaio do Prado Leite and J. H. Doorn, Eds., “Requirements traceability,” in *Perspectives on software requirements*, no. 753, Berlin: Springer , 2004, pp. 91–113.

[7] R. Brcina and M. Riebisch, “Definig Traceability for decesion Support systems,” *European Conference on Model Driven Architecture*, Germany, Berlin, pp. 39–48, 2008.

[8] S. Hanes, *Universal Traceability: A Comprehensive, Generic, Technology-Independent, and Semantically Rich Approach*. Berlin: Logos-Verlag, 2012.

[9] B. Ramesh and M. Edwards, “Issues in the Development of a Requirements Traceability Model,” in *IEEE International Symposium on Requirements Engineering*, 1993, pp. 256–259.

[10] O. M. G. OMG, “Unified Modeling Language,” 2014. [Online]. Available: http://www.uml.org/.

[11] O. M. G. OMG, “Systems Modeling Language,” 2014. [Online]. Available: http://www.omgsysml.org/.

[12] ISO, “Information technology -- Meta Object Facility (MOF),” 2015. .

[13] F. I. Romanovsky, Alexander, *Trustworthy Cyber-Physical Systems Engineering*. 2017.

[14] E. Steve, S. Margaret-Anne, and D. Daniela, “Selecting Empirical Methods for Software Engineering Research,” in *Guide to Advanced Empirical Software Engineering.*, London: Springer, 2008.

[15] CMMI, “Capability Maturity Level Integration,” 2014. [Online]. Available: http://www.sei.cmu.edu/cmmi/.

[16] O. Gotel, J. Cleland-Huang, J. Huffman, A. Dekhtyar, G. Antoniol, and J. Maletic, “The Grand Challenges of Traceability (v1.0),” London, England, United Kingdom, 2011.

[17] IEEE, “IEEE Standard Glossary of Software Engineering Terminology,” *IEEE Standard Glossary of Software Engineering Terminology*. New York, 1990.

[18] J. Cleland-Huang, O. Gotel, and A. Zisman, “Software and Systems Traceability.”

Springer, 2014.

[19] O. Gotel and A. Finkelstein, “An Analysis of the Requirements Traceability Problem,” in *1st International Conference on Requirements Engineering*, 1994, pp. 94–101.

[20] G. Spanoudakis and A. Zisman, “Software Traceability: A road map,” in *Handbook of Software Engineering and Knowledge Engineering*, vol. 3, S. K. Chang, Ed. 2005, pp. 395–428.

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

[22] M. LindVall and kristian Sandahl, “Practical Implications of Traceability,” *Softw. Pract. Exp.*, vol. 26, no. 10, pp. 1161–1180, 1996.

[23] N. Aizenbud-Reshef, J. Rubin, and Y. Shaham-Gafni, “Model traceability,” *IBM Syst. J. - Model. Softw. Dev.*, vol. 45, no. 3, pp. 515–526, 2006.

[24] P. Letelier and V. Anaya, “Customizing Traceability in a Software Development Process,” in *Information Systems Development Springer* , 2005, pp. 137–148.

[25] F. A. C. Pinheiro, *Requirements traceability, in Perspectives on software requirements*. Berlin, Germany: Springer, 2004.

[26] M. Costa and A. Da Silva, “RT-MDD framework—a practical approach,” in *European Conference on Model Driven Architecture - Traceability Workshop* , 2007, pp. 17–26.

[27] N. Narayan, B. Bruegge, A. Delater, and P. Barbar, “Enhanced Traceability in Model-based CASE Tools using Ontologies and Information Retrieval,” in *MARK’11*, 2011, pp. 24–28.

[28] M. A. Javed and U. Zdun, “A Systematic Literature Review of Traceability Approaches Between Software Architecture and Source Code,” in *18th International Conference on Evaluation and Assessment in Software Engineering*, 2014, pp. 1–10.

[29] R. Torkar, T. Gorschek, U. Raja, and K. Kamran, “Requirements traceability: systematic review and industry case study,” *Int. J. Softw. Eng. Knowl. Eng.*, vol. 22, no. 3, pp. 1–49, 2012.

[30] P. Rempel, P. Mäder, T. Kuschke, and I. Philippow, “Requirements Traceability across Organizational Boundaries - A Survey and Taxonomy,” in *Requirements Engineering: Foundation for Software Quality*, vol. 7830, 2013, pp. 125–140.

[31] E. Bouillon, P. Mäder, and I. Philippow, “A Survey on Usage Scenarios for Requirements Traceability in Practice,” in *19th international conference on Requirements Engineering: Foundation for Software Quality*, 2013, pp. 158–173.

[32] G. Loniewski, E. Insfran, and S. Abrahão, “A Systematic Review of the Use of Requirements Engineering Techniques in Model-Driven Development,” in *Model Driven Engineering Languages and Systems-13th International Conference, MODELS* , 2010, vol. 6359, pp. 213–227.

[33] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, “Lessons from applying the systematic literature review process within the software engineering domain,” *J. Syst. Softw.*, vol. 80, no. 4, pp. 571–583, 2007.

[34] B. Kitchenham and S. Charters, “Guidelines for performing systematic literature reviews in software engineering,” 2007.

[35] J. Linaker, S. M. Sulaman, M. Host, and R. M. de Mello, “Guidelines for Conducting Surveys in Software Engineering,” Lund University, Sweden, 2015.

[36] Zotero, “Zotero for Firefox,” 2016. [Online]. Available: https://www.zotero.org/.

[37] J. Dick, “Rich Traceability ,” in *1st International Workshop on Traceability for Emerging forms of Software Engineering* , 2002.

[38] R. F. Paige, “Traceability in model-driven safety critical software engineering,” in *6th ECMFA Traceability Workshop*, 2010, p. 5.

[39] M. H. Ammar, M. Benaïssa, and H. Chabchoub, “Traceability management system: Literature review and proposal of a system integrating risk management for hazardous products transportation,” in *4th International Conference on Advanced Logistics and Transport*, 2015.

[40] A. Limon and J. Garbajosa, “The Need for a Unifying Traceability Scheme,” in *European Conference on Model Driven Architecture - Traceability Workshop* , 2005, pp. 47–56.

[41] J. Cleland-Huang and C. K. Chang, “Event-based traceability for managing evolutionary change,” *IEEE Trans. Softw. Eng.*, vol. 29, no. 9, pp. 796–810, 2003.

[42] A. Espinoza, G. Botterweck, and J. Garbajosa, “A Formal Approach to Reuse Successful Traceability Practices in SPL Projects,” *Symposium On Applied Computing* . Sierre, Switzerland, pp. 2353–2359, 2010.

[43] L. Bondé, P. Boulet, and J.-L. Dekeyser, “Traceability and Interoperability at Different Levels of Abstraction in Model Transformations,” in *Building Sustainable Information Systems*, L. Henry, F. Julie, B. Andrew, B. Chris, L. Michael, and S. Christoph, Eds. Springer, 2012, pp. 263–276.

[44] O. Gotel *et al.*, *Traceability Fundamentals*. 2012.

[45] P. Rempel, P. Mader, and T. Kuschke, “An empirical study on project-specific traceability strategies,” in *21st Intn’l Requirements Engineering Conference*, 2013, pp. 195–204.

[46] K. Pohl., “Adapting traceability environments to project-specific needs,” *Communications of the ACM*, vol. 41, ACM, New York, NY, USA , pp. 54–62, 1998.

[47] R. Noll, “Enhancing traceability using ontologies,” in *ACM symposium on Applied computing*, 2007, pp. 1496–1497.

[48] R. F. Paige *et al.*, “Rigorous identification and encoding of trace-links in model-driven engineering,” *Softw. Syst. Model.*, vol. 10, no. 4, pp. 469–487, 2011.

[49] S. Pavalkis, L. Nemuraite, and E. Milevičienė, “Towards Traceability Metamodel for Business Process Modeling Notation,” *IFIP Advances in Information and Communication Technology*. pp. 177–188, 2011.

[50] H. Le Dang, H. Dubois, and S. Gérard, “Towards a traceability model in a MARTE-based methodology for real-time embedded systems,” *Innov. Syst. Softw. Eng.*, vol. 4, no. 3, pp. 189–193, 2008.

[51] R. F. P. Taromirad, M., N. Matragkas, “Towards a Multi-Domain Model-Driven Traceability Approach,” in *7th International Workshop on Multi-Paradigm Modeling co-located with 2013 ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems (MODELS).*, 2013.

[52] E. s Yu, “Social modeling and I*,” in *Conceptual Modeling: Foundations and Applications*, A. T. C. Borgida V. K., P. Giorgini, and E. S. Yu, Eds. Berlin: Springer, 2009, pp. 99–121.

[53] C. Cavalcanti *et al.*, “Towards Metamodel Support for Variability and Traceability in Software Product Lines,” in *5th Workshop on Variability Modeling of Software-Intensive Systems*, 2011, pp. 49–57.

[54] L. Yilmaz and D. Kent, “ACART: An API Compliance and Analysis Report Tool for Discovering Reference Design Traceability,” in *49th Annual Southeast Regional Conference*, 2011, pp. 243–248.

[55] H. U. Asuncion and F. François, “An end-to-end industrial software traceability tool,” in *European software engineering conference and the ACM SIGSOFT*, 2007, pp. 115–124.

[56] M. Shahid and M. N. Mahrin, “An Evaluation of Requirements Management and Traceability Tools ,” *Int. J. Comput. Electr. Autom. Control Inf. Eng.*, vol. 5, no. 6, 2011.

[57] M. Gills and M. Bogdanovs, “Extended Use of a Traceability Tool Within Software Development Project,” Kluwer Academic Publishers, 2002, pp. 175–186.

[58] A. Goknil and I. Kurtev, “Tool support for generation and validation of traces between requirements and architecture,” in *6th ECMFA Traceability Workshop*, 2010, pp. 39–46.

[59] A. Yrjönen and J. Merilinna, “Tooling for the Full Traceability of Non-functional Requirements Within Model-driven Development,” *6th ECMFA Traceability Workshop*. ACM, Paris, France, pp. 15–22, 2010.

[60] R. Tsuchiya, T. Kato, H. Washizaki, M. Kawakami, Y. Fukazawa, and K. Yoshimura, “Recovering traceability links between requirements and source code in the same series of software products,” *International Software Product Line Conference*. ACM, Tokyo, Japan, pp. 121–130, 2013.

[61] C. Ziftci and I. Krüger, “Getting More from Requirements Traceability: Requirements Testing Progress,” in *7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, 2013, pp. 12–18.

[62] N. Alhindawi, O. Meqdadi, J. I. Maletic, and B. Bartman, “A tracelab-based solution for identifying traceability links using LSI,” in *TEFSE@ICSE*, 2013.

[63] J. I. Maletic and M. L. Collard, “An XML based approach to support the evolution of model-to-model traceability links,” in *3rd international workshop on Traceability in emerging forms of software engineering*, 2005, pp. 67–72.

[64] O. Armbrust, A. Ocampo, J. Münch, M. Katahira, and Y. Koishi, “Establishing and Maintaining Traceability Between Large Aerospace Process Standards ,” *7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*. Vancouver, BC, Canada, pp. 36–40, 2013.

[65] C. Trubiani, A. Ghabi, and A. Egyed, “Exploiting Traceability Uncertainty between Architectural Models and Code,” in *Software Architecture*, vol. 9278, Springer, 2015, pp. 305–321.

[66] J. Oh and S. Kang, “A hierarchical model for traceability between requirements and architecture,” in *29th Annual ACM Symposium on Applied Computing*, 2014, pp.

1035–1042.

[67] T. K. Satyananda, D. Lee, S. Kang, and S. I. Hashmi, “Identifying Traceability between Feature Model and Software Architecture in Software Product Line using Formal Concept Analysis,” *International Conference Computational Science and its Applications* . IEEE, pp. 380–386, 2007.

[68] N. Y. Jia and G. Z. Yang, “A Method for Verifying Traceability between Feature Model and Software Architecture,” *Adv. Mater. Res.*, vol. 998–999, pp. 1085–1091, 2014.

[69] L. Linsbauer, R. Lopez-Herrejon, and A. Egyed, “Recovering traceability between features and code in product variants,” in *17th International Software Product Line Conference*, 2013, pp. 131–140.

[70] A. Sureka, S. lal, and L. Agarwal, “Applying fellegi-sunter (fs) model for traceability link recovery between bug databases and version archives,” in *Asia-Pacific Software Engineering Conference*, 2011, pp. 146–153.

[71] J. Chanda, S. Sengupta, and A. Kanjilal, “Traceability between service component and class: a model based approach,” *SIGSOFT Softw. Eng. Notes*, vol. 37, no. 6, pp. 1–5, 2012.

[72] C. Xiaofan, “Automated documentation to code traceability link recovery and visualization,” The University of Auckland, 2012.

[73] H. U. Asuncion and R. N. Taylor, “Capturing custom link semantics among heterogeneous artifacts and tools,” in *Workshop on Traceability in Emerging Forms of Software Engineering*, 2009, pp. 1–5.

[74] M. Collin, P. Denys, and R. Meghan, “Combining Textual and Structural Analysis of Software Artifacts for Traceability Link Recovery,” in *Workshop on Traceability in Emerging Forms of Software Engineering*, 2009, pp. 41–48.

[75] D. Strasunskasa and S. E. Hakkarainenb, “Domain model-driven software engineering: A method for discovery of dependency links,” *Inf. Softw. Technol.*, vol. 54, no. 11, pp. 1239–1249, 2012.

[76] A. Sardinha, In. Niu., Y. Yu., and R. Awais, “EA-Tracer: Identifying Traceability Links between Code Aspects and Early Aspects,” in *27th Annual ACM Symposium on Applied Computing* , 2012, pp. 1035–1042.

[77] D. E. Perry, “Recovering and using use-case-diagram-to-source-code traceability links,” in *6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2007, pp. 95–104.

[78] L. Aversano, F. Marulli, and M. Tortorella, *Recovering Traceability Links between Business Process and Software System Components*. 2010.

[79] S. Nair, J. L. de la Vara, and S. Sen., “A Review of Traceability Research at the Requirements Engineering ,” in *21st IEEE International Requirements Engineering Conference (RE)*, 2013.

[80] F. Polack, “Detecting and Repairing Inconsistencies Across Heterogeneous Models,” in *International Conference on Software Testing, Verification, and Validation*, 2008, pp. 356–364.

[81] J.-R. Falleri, M. Huchard, and C. Nebut, “Towards a traceability framework for model

transformations in kermeta," in *European Conference on Model Driven Architecture - Traceability Workshop* , 2006.

[82] G. C. Filho, A. Zisman, and G. Spanoudakis, "Traceability approach for i* and UML models," in *International Workshop on Software Engineering for Large-Scale Multi-Agent Systems*, 2003.

[83] P. Patrick Maeder, I. Philippow, and M. Riebisch, "Customizing traceability links for the unified process," in *Quality of software architectures*, 2007, pp. 53–71.

[84] G. Antoniol, G. Canfora, A. De Lucia, and E. Merlo, "Recovering Traceability Links between Code and Documentation," *IEEE Trans. Softw. Eng.*, vol. 28, no. 10, pp. 970–983, 2002.

[85] P. A. J. Mason and A. Saeed, "MATrA : Meta-modelling Approach to Traceability for Avionics," University of Newcastle, 2002.

[86] S. Walderhaug, U. Johansen, E. Stav, and J. Aagedal, "Towards a generic solution for traceability in MDD," in *European Conference on Model Driven Architecture - Traceability Workshop (ECMDA-TW'06)*, 2006.

[87] A. Boronat, J. Á. Carsí, and I. Ramos, "Automatic Support for Traceability in a Generic Model Management Framework," in *Model Driven Architecture -- Foundations and Applications*, 2005, pp. 316–330.

[88] A. Maté and J. Trujillo, "A trace metamodel proposal based on the model driven architecture framework for the traceability of user requirements in data warehouses," in *International conference on Advanced information systems engineering*, 2011, pp. 123–137.

[89] N. Drivalos, D. S. Kolovos, R. F. Paige, and K. J. Fernandes, "Engineering a DSL for software traceability," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, vol. 5452, pp. 151–167.

[90] N. Anquetil, "A model-driven traceability framework for software product lines.," *Software. Syst. Model*, vol. 9, no. 4, pp. 427–451, 2010.

[91] J. A. Goguen, "An Object-Oriented Tool for Tracing Requirements," *IEEE Softw.* , vol. 13, no. 2, pp. 52–64, 1996.

[92] O. Gotel and A. Finkelstein, "Contribution Structures," in *2nd International Symposium on Requirements Engineering.*, 1995, pp. 100–107.

[93] J. M. Constantopoulos P Mylopoulos Y, Vassiliou Y, ", "The Software Information Base: A Server for Reuse," *Int. J. Very Large Data Bases*, vol. 4, no. 1, pp. 1–43, 1993.

[94] K. Mohan and B. Ramesh, "Managing Variability with Traceability in Product and Service Families," in *35th Annual Hawaii International Conference on System Sciences* , 2002, vol. 3, p. 76.

[95] A. Kozlenkov and A. Zisman, "Are their Design Specifications Consistent with our Requirements?," in *IEEE Joint International Conference on Requirements Engineering*, 2002.

[96] B. Ramesh and M. Jarke, "Toward Reference Models for Requirements Traceability," *IEEE Trans. Softw. Eng.*, vol. 27, no. 1, pp. 58–93, 2011.

[97] Paige Richard F, Olsen Gøran K, Kolovos Dimitrios S, and Z. Steffen, “Building Model-Driven Engineering Traceability Classifications,” in *ECMDA-Traceability Workshop Proceedings*, 2008, pp. 49–58.

[98] G. Spanoudakis, A. Zisman, and E. Pérez-Miñanab, “Rule-Based Generation of Requirements Traceability Relations,” *Syst. Softw.*, vol. 72, no. 2, pp. 105–127, 2004.

[99] P. Xu, “Supporting Workflow Management Systems with traceability,” in *35th Annual Hawaii International Conference on System Sciences*, 2002, vol. 3.

[100] I. Alexander, “Semi Automatic Tracing of Requirement Versions to Use Cases – Experience and Challenges,” in *2nd International Workshop on Traceability in Emerging Forms of Software Engineering* , 2003.

[101] M. Riebisch and I. Philippow, “Evolution of Product Lines Using Traceability,” in *Workshop on Engineering Complex Object-Oriented Systems for Evolution*, 2001.

[102] J. I. Maletic, E. V Munson, A. Marcus, and T. N. Nguyen, “Using a Hypertext Model for Traceability Link Conformance Analysis ,” in *2nd International Workshop on Traceability for Emerging Forms of Software Engineering* , 2003.

[103] A. von Knethen, “Automatic Change Support Based on a Trace Model,” in *1st International Workshop on Traceability in Emerging Forms of Software Engineering* ., 2002.

[104] K. Pohl., “PRO-ART: Enabling Requirements Pre-Traceability,” in *2nd IEEE International. Conference on Requirements Engineering*, 1996, p. 76.

[105] P. Letelier, “A Framework for Requirements Traceability in UML-based Projects,” in *1st Intl. Workshop on Traceability in Emerging Forms of Softw. Eng*, 2002, pp. 32–41.

[106] P. Mason, A. Saeed, P. Arkley, and S. Riddle, “Meta-Modelling Approach to Traceability for Avionics: A Framework for Managing the Engineering of Computer Based Aerospace Systems.,” in *10th IEEE International Conference on Engineering of Computer-Based Systems* , 2003, pp. 233–246.

[107] P. Roques, “Modeling Requirements with SysML,” *Requirement Engineering MAgazine*, no. 2015-02, IREB, Online, 2015.

[108] O. M. G. OMG, “Business process Model Notation,” 2014.

[109] O. M. Group, “Object Constraint Language (OCL),” 2014.

[110] No Magic Inc., “MagicDraw UML Profiling&DSL User Guide,” 2014.

[111] Z. Drey, C. Faucher, F. Fleurey, V. Mahé, and D. Vojtisek, “Kermeta language reference manual,” 2014.

[112] Graphviz, “Graph Vsulization Software,” 2017. [Online]. Available: http://www.graphviz.org/.

[113] W3C, “Extensible Markup Language (XML) 1.0 (Fifth Edition),” 2017. [Online]. Available: https://www.w3.org/TR/REC-xml/.

[114] D. S. Kolovos, L. Rose, A. Garcia-Dominguez, and R. Paige, “The Epsilon Validation Language,” in *The Epsilon Book*, 2014, pp. 57–76.

[115] O. M. G. OMG, “Modeling and Analysis for Real-time and Embedded Systems,” 2017. [Online]. Available: http://www.omg.org/omgmarte/Events.htm.

[116] A. Maté and T. Mondéjar, “Tracing conceptual models’ evolution in data warehouses

by using the model driven architecture," *Comput. Stand. Interfaces*, vol. 36, no. 5, pp. 831–843, 2014.

[117] N. Mustafa and Y. Labiche, "The Need for Traceability in Heterogeneous Systems: A systematic literature review," *IEEE International Computers, Software & Applications Conference*. Italy, 2017.

[118] C. Robson, *Real World Research: A Resource for Social Scientists and Practitioner-Researchers* . Oxford, UK : Blackwell Publishing , 2002.

[119] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Publishing Company , 2012.

[120] SurveyMonkey, "Traceability Survey," 2018. .

[121] ORCANOS, "Requirements traceability tool," 2018. [Online]. Available: https://www.orcanos.com/compliance/requirements-traceability-tool/.

[122] Attlassian, "TraceabilityX for Jira," 2018. [Online]. Available: https://marketplace.atlassian.com/plugins/es.excentia.jira.plugins.jira-traceabilityx-plugin/server/overview.

[123] Sparx, "Tools for traceability in enterprise architect," 2018. [Online]. Available: http://www.sparxsystems.com/resources/demos/traceabilitytools/webinar-tools-for-traceability.html.

[124] IBM, "Using Rational RequisitePro," 2018. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSSHCT_7.1.0/com.ibm.reqpro.help/get_start/c_product_overview.html.

[125] A. S. Foundation, "Suberversion," 2017. [Online]. Available: https://subversion.apache.org/.

[126] GitHub, "GitHub," 2018. [Online]. Available: https://github.com/.

[127] IBM, "IBM Rational DOORS Next Generation," 2018. [Online]. Available: https://jazz.net/products/rational-doors-next-generation/.

[128] O. M. G. OMG, "Meta Object Facility," 2018. [Online]. Available: https://www.omg.org/spec/MOF/. [Accessed: 22-Sep-2018].

[129] A. Rahman and D. Amyot, "A DSL for importing models in a requirements management system," in *4th IEEE International Model-Driven Requirements Engineering Workshop* , 2014, pp. 37–46.

[130] M. Gogolla and M. Richters, "USE: A UML-Based Specification Environment for Validating UML and OCL," *Sci. Comput. Program.*, vol. 69, no. 1–3, pp. 27–34, 2007.

[131] W3C, "Resource Description Framework," 2016. [Online]. Available: https://www.w3.org/RDF/.

[132] O. M. G. OMG, "Open Services for Lifecycle Collaboration," 2017. [Online]. Available: https://open-services.net/.

[133] A. Marques, F. Ramalho, and W. L. Andrade, "Towards a requirements traceability process centered on the traceability model," in *30th Annual ACM Symposium on Applied Computing*, 2015, pp. 1364–1369.

[134] G. Olsen and J. Oldevik, "Scenarios of Traceability in Model to Text Transformations," in *Third European Conference, ECMDA-FA 2007, J11-1une 5,*

2007, 2007, vol. 4530, pp. 144–156.

[135] B. Grammel and für Softwaretechnologie, "Automatic Generation of Trace Links in Model-driven Software Development," Technische Universität Dresden, 2014.

[136] A. de Lucia, F. Fasano, and R. Oliveto, "Recovering Traceability Links in Software Artifact Management Systems using Information Retrieval Methods," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 4, 2007.

[137] A. Rummler, B. Grammel, and C. Pohl, "Improving Traceability in Model-Driven Development of Business Applications ," in *European Conference on Model Driven Architecture - Traceability Workshop* , 2007, pp. 7–15.

[138] W3C, "Web Ontolgy Language," 2016. [Online]. Available: https://www.w3.org/TR/owl-features/.

[139] W3C, "SPARQL Query Language for RDF," 2016. [Online]. Available: https://www.w3.org/2001/sw/DataAccess/rq23/.

[140] L. Miller and D. Brickley, "FOAF," 2016. [Online]. Available: http://www.foaf-project.org/.

[141] E. Dumbill, "Description of a Project," 2016. [Online]. Available: http://lov.okfn.org/dataset/lov/vocabs/doap.

[142] Cognitum, "Fluent Editor 2015," 2017. [Online]. Available: http://www.cognitum.eu/semantics/FluentEditor/.

[143] R Foundation, "The R project for statistical computing," 2017. [Online]. Available: https://www.r-project.org/.

[144] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empir. Softw. Eng.*, vol. 14, no. 2, p. 131, 2008.

[145] E. Hutchins, "How a Cockpit Remembers Its Speeds," *Cogn. Sci.*, vol. 19, no. 33, pp. 265–288, 1995.

[146] M. Bretschneider, H.-J. Holberg, E. Bode, I. Bruckner, T. Peikenkamp, and H. Spenke, "Model-based Safety Analysis of a Flap Control System," *Int. Counc. Syst. Eng.*, vol. 14, no. 1, pp. 246–256, 2004.

[147] R. Feldt and A. Magazinius, *Validity Threats in Empirical Software Engineering Research - An Initial Survey*. 2010.

[148] J. Siegmund, N. Siegmund, and S. Apel, "Views on Internal and External Validity in Empirical Software Engineering," in *Proceedings of the 37th International Conference on Software Engineering* , 2015, vol. 1, pp. 9–19.

[149] CAE, "Canadian Aviation Electronics," 2017. [Online]. Available: http://www.cae.com/.

[150] Papyrus, "Papyrus Modeling Environment." [Online]. Available: http://www.eclipse.org/papyrus/. [Accessed: 07-Sep-2018].

[151] Profpilot, "Profpilot," 2016. [Online]. Available: https://profpilot.co.uk.