Towards Traceability Modeling for the Engineering of Heterogeneous Systems

Nasser Mustafa and Yvan Labiche
Carleton University, Department of Systems and Computer Engineering, Ottawa, ON, Canada

Keywords: Heterogeneous, Traceability, Generic, Characterization, Traceability Requirements.

Abstract: Capturing traceability information among artifacts allows for assuring product quality in many ways such as tracking functional and non-functional requirements, performing system validation and impact analysis. Although literature provides many techniques to model traceability, existing solutions are either tailored to specific domains (e.g., Ecore modeling languages), or not complete enough (e.g., lack support to specify traceability link semantics). This paper examines the current traceability models and identifies the drawbacks that prevent from capturing some traceability information of heterogeneous artifacts. In this context, heterogeneous artifacts refer to artifacts that come from widely different modelling notations (e.g., UML, Simulink, natural language text, source code). Additionally, the paper proposes traceability model requirements that are necessary to build a generic traceability model. We argue that the proposed requirements are sufficient to build a traceability model oblivious of the heterogeneity of the models which elements need to be traced. We also argue that our proposed requirements can be adopted to create a generic traceability model that provides flexibility and can accommodate new ways of characterizing and imposing constraints on trace links or systems artifacts. The proposed requirements incorporate the ideas from many existing solutions in literature, in an attempt to be as complete as possible.

1 INTRODUCTION

Traceability in its simplest form is the ability to describe and follow the life of software artifacts (Winkler and Pilgrim 2010). Traceability is often required for quality assurance (Pinheiro 2004), to certify or qualify system and software products. One important challenge when implementing traceability requirements is to relate multiple artifacts that can come from diverse disciplines, which are not necessarily software related, such as electronic, mechanical, and hydraulics. As a result, artifacts to be traced by traceability links are typically created using different modeling languages, different tools. 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 codes.

As an example, consider the engineering of a full flight simulator, which artificially re-creates an aircraft 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 a typical airport), 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). It can contain software systems simulating cockpit instruments or the same cockpit instruments (e.g., a flight management system) as the ones actually used in the aircraft. The traceability problem arise when a system encompassing widely different domains of expertise, such as the full flight simulator case in which many heterogeneous models need to be related to one another. 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 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 take-off and landing characteristics (e.g., visuals, air traffic data) of typical airports; one

Mustafa N. and Labiche Y..
Towards Traceability Modeling for the Engineering of Heterogeneous Systems.
DOI: 10.5220/0005246103210328
In *Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2015), pages 321-328
ISBN: 978-989-758-083-3

model would record the requirements for the whole system and the requirements for its many parts (hardware and software); one model would record data about verification and validation objectives and activities; one model would record faults and failures. These models are also heterogeneous because they are not typically specified using a common notation and semantic, i.e., a common metamodel. For instance, some software elements can be specified with the UML, some system levels characteristics can be modeled with SysML, some hardware elements can be modeled with Simulink models.

Another traceability problem arise in traceability management since it 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 additional artifacts need to be traced; one may discover down the road that artifacts from new models need to be traced.

The traceability problems mentioned above led us to investigate the existing traceability models for a generic traceability solution. Our search in the literature showed that the existing solutions lack some modeling elements that prevent from delivering a generic traceability solution oblivious of the heterogeneity of the models which elements need to be traced.

The contribution of this paper is to define requirements for a generic traceability model, and identify the drawbacks of the existed solutions based on the identified requirements.

The rest of the paper will be structured as follows: Section 2 introduces important traceability concepts necessary for our discussion; Section 3 identifies the requirements and characteristics of a generic traceability model; Section 4 reviews the literature and identifies the current traceability models; Section 1 provides our analysis of the current traceability models and their drawbacks, and section 6 concludes the paper.

2 TRACEABILITY CONCEPTS

Traceability has originated in Software Engineering particularly, in Requirement Engineering and has permeated Model Driven Development (MDD) and System Engineering. According to the IEEE dictionary (IEEE 1990) traceability is “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 Software and System Engineering. In Requirement engineering, Gotel and Finkelstein (Gotel and Finkelstein 1994) defined Requirement Traceability, or traceability for short, 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” (Gotel and Finkelstein 1994). They extended this definition to define other traceability types such as pre-requirement specification (pre-RS), which refers to “the aspects of a requirement's life prior to its inclusion in the requirement specification”, and post-requirement specification (post-RS), which refers to “the aspects of requirement’s life that result from its inclusion in the requirement specification”. Additionally, traceability between artifacts can be classified into horizontal or vertical traceability. Horizontal traceability implies tracing artifacts produced during different phases of development such as tracing artifacts from the analysis phase to the design phase, or tracing artifact between two different models such as a spread sheet produced during initial discussions with customers and a software requirements document produced during requirement elicitation (Spanoudakis and Zisman 2005). Vertical traceability, means tracing artifacts within a model or a phase (e.g., tracing two requirements during the analysis phase) (Spanoudakis and Zisman 2005).

In MDD, the notion of traceability is restricted to the situation where models are transformed from other models (Amar, Leblanc and Coulette 2008): elements of the input model trace to elements of the output model. Another definition used in MDD, which is closer to the Requirement Engineering definition, defines traceability as “any relationship that exists between artifacts involved in the software engineering life cycle” (Aizenbud-Reshef, Nolan, Rubin et al. 2006).

The definitions of horizontal and vertical traceability can be problematic when we include the type of artifacts being traced (i.e. have similar or different types), and the models boundaries (i.e., whether the artifacts are within the same phase or model or across different models). Mason (Mason 2002) argued that the definitions of vertical and horizontal traceability in software engineering do not fit the system engineering context. He extended the definition of vertical and horizontal traceability by

Table 1: Vertical/Horizontal traceability combinations.

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.

introducing the *micro* and *macro* terms to differentiate traceability *within* and *across decomposition levels* such as system, sub-system, and components. In addition, he introduced the *intra* and *inter* terms to differentiate traceability *within* and *across* system *descriptions* (i.e., system interacting with another system), respectively. Consequently, he introduced new combinations of the notions of vertical and horizontal traceability types (Table 1). For instance, Intra-Macro Vertical traceability means the ability to navigate or describe the relationship between artifacts of different types within system descriptions and across system levels of decomposition. These new terms introduced new subtypes that can be applied to vertical and horizontal trace links.

In order to implement traceability in software systems the concepts of *trace*, *trace artifact,* and *trace link* are typically defined. A *trace* is composed of a single source artifact, single target artifact, and single trace link (Cleland-Huang, Gotel and Zisman 2014). A *trace artifact* refers to "traceable unit of data" such as a class, requirement, or a document. The level of granularity of a trace artifact can be defined by its size: for instance, a requirement in a document has a fine-grained granularity, while a document that has as set of requirements has a coarse-grained granularity. Additionally, trace artifacts can be classified according to their types (e.g., test artifacts, design artifacts) (Cleland-Huang, Gotel et al. 2014). A *trace link* specifies a relationship between a source and target artifacts, which can be traversed from source to target artifact or from target to source artifact. Similar to trace artifacts, a *trace link* may have a type that can be identified based on the link's syntax or semantics. The semantics provides a purpose or a meaning to the relationship between source and target artifacts. Although trace link and trace relations may be used interchangeably, there is a difference between the two terms since the former is used to refer to "a specified *association* between a pair of *artifacts*, one comprising the *source artifact* and one comprising the *target artifact*" (Cleland-Huang, Gotel et al. 2014), whereas the latter refers to "all the *trace links* created between two sets of specified *trace artifact types"*. Trace links may be categorized based on different criteria such as model type (e.g., MDE, non-MDE models), purpose, usage, and functionality (Mason 2002; Costa and Da Silva 2007; Paige, Olsen, Kolovos et al. 2008; Cleland-Huang, Gotel et al. 2014)

From the discussion above it becomes clear that the notion of traceability is varied, that traceability information can be characterized in many different, sometimes complementary ways, which could be domain, organization or even project specific.

3 GENERIC TRACEABILITY MODEL REQUIREMENTS

Since we are interested in systems that are realized through software and hardware solutions, we extend the MDD traceability definition and consider traceability as any relationship that exists between artifacts involved in the system engineering life cycle.

We define a generic traceability model as a model that has the ability to capture traceability information of heterogeneous systems, as defined and illustrated in the Introduction, and provide flexibility to its users to model any required traceability information, with various taxonomies as presented in Section 2, and that can accommodate flexibility and evolution without having to change the model itself (only its instance would change).

We assume users to have three roles, similarly to other similar technologies, 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. 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; A domain expert who would use the technology and tool support to create traceability information between heterogeneous artifacts and reason about this information.

We are searching for a traceability model that must be oblivious of the heterogeneity of the models

Table 2: Traceability model requirements.

	Traceability Requirement	Model Characteristics
1	Model implementation	Independent of any language, tool, or framework.
2	Types of source and target artifacts types	Any type (i.e., heterogeneous or homogeneous).
3	Association cardinality between source and target artifacts	Allows for tracing source to target artifacts with cardinality 1-1, 1-many, and many-many.
4	Type of traced models	MDE and/or non-MDE models, i.e., any model, in the widest sense of the word, including, but not limited to, UML, SysML, Simulink models, electronic design, mechanical design, blueprint, plain language, source code, tests.
5	Characterization of trace links semantics	Allows for applying more than one characterizations to a given trace link.
6	Characterization of source and target artifacts types	Allows for applying more than one characterization to a given artifact.
7	Artifacts granularity	Allows for tracing artifacts at different levels of granularity.
8	Applying Constraints	Allows for applying more than one constraint to an artifact, a trace, or a trace link.
9	Support for model transformation	Allows for capturing traceability information during model to model transformation
10	Trace link direction	Indicate whether the trace link is outgoing from or incoming to with reference to the source artifact.
11	Artifacts linking	Prevents from establishing illegal links between certain artifacts
12	Model extensibility and flexibility	Allows for accommodating new types of trace links and artifacts without changing the model itself.

which elements are traced. For instance, modeling traceability links should not rely on the fact that artifacts are instances of a 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).

Based on the above assumptions, we put forward that a satisfactory generic traceability model should possess the following general characteristics, which are summarized in Table 2. The model implementation should be independent of any tool, language, or framework.

- It shall allow modeling traceability between artifacts of similar or different types (i.e., homogeneous and heterogeneous artifacts).
- It shall allow modeling traceability between source and target artifacts of one-to-one, one-to-many, and many-to-many cardinalities.
- It shall specify the direction of the trace link (i.e., is the link from source-to-target or from target-to-source)
- It shall allow capturing traceability information between artifact within one model or across different models.
- It shall allow applying more than one constraint to trace elements (i.e., trace link, artifact) using any constraint language.
- It shall allow apply more than one characterization to an artifact.
- It shall allow specifying various semantics for trace links between artifacts
- It shall allow modeling traceability between model elements at different levels of granularity (i.e., different conceptual or decomposition levels).
- It shall be able to receive different kinds of artifacts generated using different tools.

- It shall allow capturing traceability information between MDE and/or non-MDE model types.
- It shall allow capturing traceability information between models during model transformation.
- The model shall prohibit establishing illegal links between some artifacts.
- The model shall be flexible such that it allows for accommodating new trace links and artifacts without changing the model itself.

4 RELATED WORK ON TRACEABILITY MODELS

Paige and colleagues (Paige, Drivalos, Kolovos et al. 2011) defined a taxonomy of semantically rich trace link between MDE models that may be constructed using diverse modeling languages. Trace links are said to be semantically rich because their types conform to a project-specific traceability metamodel, accompanied by a set of project-specific correctness constraints. For validation purposes, the authors applied their method for identifying trace links to the Requirement Engineering phase, which they split into early activities, modeling with I* (Yu 2009) and later activities, modeling with the UML (specifically the class diagram, i.e., a domain model). It turns out that this solution satisfies partially requirements number 2, 3, 4, 5, 8, 10 (Table 2). One drawback of their approach, using the *I*/UML* example is that the types of traceability links between I* and class diagram artifacts need to be in the metamodel itself, making the metamodel difficult to evolve to accommodate new artifacts, new types of models (recall our requirements). Specifically, 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. As a result, this solution fails to satisfy requirements 1, 6, 7, 9, 11, 12 (Table 2).

Pavalkis and colleagues (Pavalkis, Nemuraite and Milevičienė 2011) defined a traceability metamodel for relating artifacts in an instance of the Business Process Model Notation (BPMN) (Object Management Group 2014a): e.g., between resources and their process, between participants involved in messages (message sender and receiver). Although their approach is extensible and customizable since new rules can be defined for BPMN traceability links, their solution is specific to BPMN, and more generally to MOF-based modeling techniques, and is therefore not adequate for our purpose. In summary, this solution satisfies partially requirements 3, 4, 8, 10, 12 but not requirements 1, 2, 5, 6, 7, 9, 11 (Table 2).

Drivalos and colleagues (Drivalos, Kolovos, Paige et al. 2008) presented the Traceability Metamodeling Language (TML) for defining the syntax and semantic of traceability metamodels. With TML, one models traceability links between Ecore-based model elements while providing some context-specific information. Constraints can be expressed in the Epsilon Validation Language (EVL) (Kolovos, Rose, Garcia-Dominguez et al. 2014), an extension of the OCL (Object Management Group 2014b). 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). We note that the solution is specific to Ecore-based models, does not provide enough information about the direction of the trace link (i.e., does not specify which of the artifacts is a source or a target), does not provide a mechanism for capturing traceability information during model transformation (transitivity of links), and doesn't accommodate various characterizations for trace links or artifacts. Therefore, this solution satisfies partially the requirements 2, 3, 4, 5, 6, 8, 12 but fails to satisfy requirements 1, 7, 9, 10, 11 (Table 2).

Falleri and colleagues (Falleri, Huchard and Nebut 2006) 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. This solution is good for capturing traceability links during model transformation as it can capture a sequence or a chain of links between source and target artifacts. However, it is domain (transformation) specific and can only do that, the source and target artifacts must conform to the same metamodel, and it doesn't provide any semantics or constraints on the type of the trace links. They

validated their solution by first writing a simple transformation code in Kermeta (Drey, Faucher, Fleurey et al. 2014) that maps each UML class (resp. attribute) into a database table (resp. column), and then visualize the traceability links as a graph (using graphviz). In summary, this solution satisfies partially the requirements 3, 4, 9, 10, 12 but not requirements 1, 2, 5, 6, 7, 8, 11 (Table 2).

Cysneiros and colleagues (Cysneiros, Zisman and Spanoudakis 2003) propose a light-weight XML-based approach to generate bidirectional traceability relations between UML use case and class diagrams and I* models: e.g., actors in I* are linked to actors in the use-case diagram. Although they provide an approach to generate traceability links between two heterogeneous models (i.e., I* and UML), there is no indication that it can be extended to other model types, for instance models not conforming to MOF-based metamodels. This solution satisfies partially requirements 2, 3, 4, 7, 8 but does not satisfy requirements 1, 5, 6, 9, 10, 11, 12 (Table 2).

Anquetil and colleagues (Anquetil, Kulesza, Moreira et al. 2010) introduced a traceability reference metamodel that supports general traceability for aspect-oriented and model-driven software product line. A trace link is bidirectional, multivalued, between artifacts uniquely identified with a Universal Resource Identifier (URI), thereby accommodating for model heterogeneity. Their model has metaclasses that allow a trace link or an artifact to have subtypes. These metaclasses can be used to specify (il)legal links between certain artifacts, e.g., specifying which types of artifacts can (or cannot) be linked, or to provide justifications for linked artifacts. This solution is therefore the one in the literature that satisfies the largest number of our requirements (Table 2), specifically, requirements 2, 3, 4, 7, 8, 10, 11, 12. However, we note a few issues with this solution. It does not explicitly model traceability links between artifacts due to model transformations. More seriously the specification of link types or artifact types assumes a subtype relationship whereas literature suggests one would likely be interested in orthogonal taxonomies of such types. For instance, one would like to combine the (vertical, horizontal) taxonomy (Ramesh and Edwards 1993) with the (refine) taxonomy (Ramesh and Edwards 1993). The solution is therefore not as generic and extensible as claimed by the authors, and does not fit our needs. Specifically, the solution does not satisfy requirements 1, 5, 6, 9.

Table 3: Summary of traceability models features.

Reference	Traceable Models	Metamodel Technology	Tool Support	Validation	Extension to new link types without changing metamodel	Important Design Features	Satisfied Requirements from Table 2
Paige, 2011	Ecore based	UML class diagram	Eclipse possible	Partial instantiation, couple of case studies	No	trace links classifications, linking heterogeneous models	2, 3, 4, 5, 8, 10 (yes/partially). 1,6,7,9, 11, 12 (no)
Pavalkis, 2011	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	3, 4, 8, 10, 12 (yes/ partially). 1, 2, 5, 6, 7, 9, 11 (no)
Drivalos, 2008	MOF-based models	UML class diagram	Eclipse	Partial instantiation, one case study	Yes, but limited to MOF	modeling link types	2, 3, 4, 5, 6, 8, 12 (yes/ partially). 1, 7, 9, 10, 11 (no)
Falleri, 2006	MOF-based models	UML class diagram	Kermeta	Partial instantiation, one case study	Yes, but limited to MOF	sequence of links in model transformation	3, 4, 9, 10, 12 (yes/partially). 1, 2, 5, 6, 7 ,8, 12 (no)
Cysneiros, 2003	Heterogeneous	XML	Prototype tool.	Partial instantiation, one case study	No	linking of heterogeneous models	2, 3, 4, 7, 8 (yes/ partially). 1, 5, 6, 9, 10, 11, 12 (no)
Anquetil, 2010	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	2, 3, 4, 7, 8, 10, 11, 12 (yes). 1, 5, 6, 9 (no)

5 ANALYSIS OF TRACEABILITY MODELS AND DRAWBACKS

The search in the literature for a solution to the problem discussed in the Introduction was not successful. The main reason is that, each existing solution is tailored to a specific domain: e.g., some solutions can only trace artifacts from MOF-based models, some solutions can only trace during model transformation. As we conducted our search we also noticed, putting aside the abovementioned issue, 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 current existing solutions at once.

To summarize, existing solutions fail to solve our problem, as defined in the introduction, and summarized in terms of requirements of Table 3 for one or more of the following reasons. They target specific domains such as model transformation, Ecore models, or BPMN (Cysneiros, Zisman et al. 2003; Falleri, Huchard et al. 2006; Amar, Leblanc et al. 2008; Drivalos, Kolovos et al. 2008; Pavalkis, Nemuraite et al. 2011) as opposed to heterogeneous models(Cysneiros, Zisman et al. 2003; Anquetil, Kulesza et al. 2010). As a result, some cannot accommodate the definition of new traceability types between new types of artifacts, or cannot easily do so(Cysneiros, Zisman et al. 2003; Paige, Drivalos et al. 2011). They lack the ability to specify the semantics of trace links (Falleri, Huchard et al. 2006; Pavalkis, Nemuraite et al. 2011) although work exist specifically on that topic (Drivalos, Kolovos et al. 2008; Kolovos, Paige and Polack 2008; Anquetil, Kulesza et al. 2010; Paige, Drivalos et al. 2011).

In addition to the drawbacks discussed in section 4, we set up comparison criteria among the existing traceability models based on the requirement we stated in section 3. The results, summarized in Table 3 indicate that no one model can comprise all the requirements we stated in section 3. This suggests a need for a traceability model to accommodate such requirements.

6 CONCLUSIONS

Traceability in its simplest form is the ability to describe and follow the life of software artifacts (Winkler and Pilgrim 2010). Collecting traceability information plays an important role in ensuring quality, and is also mandated by many agencies for instance to qualify or certify software and systems. In our work we consider traceability needs during the engineering of systems that are realized through software and hardware solutions, and that include a wide range of disciplines and therefore heterogeneous modeling notations. We argued that, as a result, the solution to model traceability information between artifacts in the many models that specify a system must be oblivious of the solutions being used to model those artifacts. In other words, the traceability model must be oblivious of the heterogeneity of the models which elements are traced. Additionally, we argued that the solution to model traceability should accommodate situations where new artifacts, possibly in new models, need to be traced, where new ways of characterizing artifacts and traceability links need to be used. In other words, the traceability modeling language being devised should not change when new artifacts, coming from models created with new modeling notations, possibly characterized in new ways, need to be traced. The solution to model traceability information should be flexible to accommodate many different situations and flexibility should come at the model instance level instead of at the model level to facilitate updates. In other words, we conclude that there is a need for yet another traceability model.

ACKNOWLEDGEMENTS

This work was performed under the umbrella of a NSERC-CRD grant. The authors would like to thank NSERC, CRIAQ, CAE, CMC Electronics, and Mannarino Systems & Software for their financial support.

REFERENCES

Aizenbud-Reshef, N., B. T. Nolan, J. Rubin, et al. (2006). "Model traceability " *IBM Systems Journal 45*(3): pp. 515–526.

Amar, B., H. Leblanc and B. Coulette (2008). A Traceability Engine Dedicated to Model Transformation for Software Engineering. *European Conference on Model Driven Architecture - Traceability Workshop* Berlin.

Anquetil, N., U. Kulesza, A. Moreira, et al. (2010). "A model-driven traceability framework for software product lines. ." *Softw. Syst. Model 9*(4): pp. 427-451.

Cleland-Huang, J., O. Gotel and A. Zisman, Eds. (2014). Software and Systems Traceability, Springer.

Costa, M. and A. Da Silva (2007). RT-MDD framework—

a practical approach. *European Conference on Model Driven Architecture - Traceability Workshop.*

Cysneiros, F., A. Zisman and G. Spanoudakis (2003). Traceability approach for I* and UML models. *International Workshop on Software Engineering for Large-Scale Multi-Agent Systems*, Portland.

Drey, Z., C. Faucher, F. Fleurey, et al. (2014). Kermeta language reference manual.

Drivalos, N., D. S. Kolovos, R. F. Paige, et al. (2008). Engineering a DSL for software traceability. *Software Language Engineering*.

Falleri, J., M. Huchard and C. Nebut (2006). Towards a traceability framework for model transformations in kermeta. *European Conference on Model Driven Architecture - Traceability Workshop.*

Gotel, O. and A. Finkelstein (1994). An Analysis of the Requirements Traceability Problem. *Proceedings of the First International Conference on Requirements Engineering*, Utrecht, The Netherlands.

IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology. *IEEE Standard Glossary of Software Engineering Terminology*. I. s. board. New York.

Kolovos, D., R. Paige and F. Polack (2008). Detecting and Repairing Inconsistencies Across Heterogeneous Models. *International Conference on Software Testing, Verification, and Validation*.

Kolovos, D., L. Rose, A. Garcia-Dominguez, et al. (2014). 'The Epsilon Validation Language', in (eds.) *The Epsilon Book*. pp. 57-76.

Mason, P. (2002). MATrA : Meta-modelling Approach to Traceability for Avionics, University of Newcastle.

Object Management Group (2014a). Business process Model Notation (BPMN).

Object Management Group. (2014b). "Object Constraint Language (OCL)." Available at: http://www.omg.org/spec/OCL. (Accessed 20 July, 2014).

Paige, F., N. Drivalos, D. S. Kolovos, et al. (2011). "Rigorous identification and encoding of trace-links in model-driven engineering." *Software & Systems Modeling 10*(4): pp. 469-487.

Paige, F., G. K. Olsen, D. Kolovos, et al. (2008). Building Model-Driven Engineering Traceability Classifications. *European Conference on Model Driven Architecture - Traceability Workshop* Berlin, Germany.

Pavalkis, S., L. Nemuraite and E. Milevičienė (2011). "Towards Traceability Metamodel for Business Process Modeling Notation." *IFIP Advances in Information and Communication Technology*: pp. 177-188.

Pinheiro, F. A. C. (2004). 'Requirements traceability', in J. C. Sampaio do Prado Leite and J. H. Doorn (eds.) *Perspectives on software requirements*. Berlin: Springer pp. 91-113.

Ramesh, B. and M. Edwards (1993). Issues in the Development of a Requirements Traceability Model. *Proceedings of the IEEE International Symposium on Requirements Engineering*.

Spanoudakis, G. and A. Zisman (2005). 'Software Traceability: A road map', in S. K. Chang (eds.) *Handbook of Software Engineering and Knowledge Engineering*. pp. 395-428.

Winkler, S. and J. Pilgrim (2010). "A survey of traceability in requirements engineering and model-driven development." *Software and Systems Modeling 9*(4): pp. 529-565.

Yu, E. (2009). 'Social modeling and I*', in A. T. C. Borgida, V. K., P. Giorgini and E. S. Yu (eds.) *Conceptual Modeling: Foundations and Applications*: Springer. pp. 99-121.