Modeling Traceability for Heterogeneous Systems

Nasser Mustafa and Yvan Labiche
Carleton University, 1125 Colone by Drive, Ottawa, Ontario, Canada

Keywords: Traceability, Modeling, Heterogeneity, Characterization, Generic.

Abstract: In System Engineering, many systems encompass widely different domains of expertise; there are several challenges in relating these domains due to their heterogeneity and complexity. Although, literature provides many techniques to model traceability among heterogeneous domains, 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 proposes a generic traceability model that is not domain specific; it provides a solution for modeling traceability links among heterogeneous models, that is, systems for which traceability links need to be established between artifacts in widely different modeling languages (e.g., UML, block diagrams, informal documents). Our solution tackles the drawbacks of existing solutions, and incorporates some of their ideas in an attempt to be as complete as possible. We argue that our solution is extensible in the sense that it can adapt to new modeling languages, new ways of characterizing traceability information for instance, without the need to change the model itself.

1 INTRODUCTION

Traceability refers to the ability of following the life of software artifacts (Winkler and Pilgrim, 2010). It has gained more attention in the past 20 years and is mandated by many industries such as aviation, automobile, and nuclear power. It is required to certify or qualify systems and software products (Pinheiro, 2004). Traceability needs arise due to many problems during system development. For example, during system development in the System Engineering field there is a need to relate many heterogeneous artifacts. These artifacts are not necessarily software related; they can be also mechanical or electrical. Moreover, these artifacts can be modeled by different languages and 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.

Another problem arises due to the fluidity of activities since not all traceability requirements are known to the system engineer upfront. For example, the granularity and the type of traced artifacts are not easy to discover upfront. In several cases a system engineer might need to obtain traceability information of new artifacts, or he might want to link two models, or a requirement to a model that refines it. Therefore, the heterogeneity and fluidity of artifacts require a traceability model that can accommodate capturing the traceability information of such artifacts. We have demonstrated the need of such model in our previous work using the example of full flight simulator.

Our search in the literature for a solution to the problems discussed above was not successful (Mustafa, 2015). The main reason is that, each solution we found is tailored to a specific domain: e.g., some solutions can only trace artifacts from MOF-based models; and some solutions can only trace during model transformation.

This paper contribution is manifold. It involves the design of a generic traceability model oblivious of the heterogeneity of the model's elements that need to be traced. We argue that our solution is extensible in the sense that it can adapt to new modeling languages, new ways of characterizing traceability information for instance, without the need to change the model itself. Our traceability model should be able to receive different kinds of artifacts in different kinds of models that are generated from different tools. Our trace model instance can then be used to support traditional traceability management tasks such as querying to identify broken traceability links (e.g., a requirement is traced to a component, which is traced to a class

Mustafa N. and Labiche Y..
Modeling Traceability for Heterogeneous Systems.
DOI: 10.5220/0005520303580366
In *Proceedings of the 10th International Conference on Software Engineering and Applications* (ICSOFT-EA-2015), pages 358-366
ISBN: 978-989-758-114-4
Copyright © 2015 SCITEPRESS (Science and Technology Publications, Lda.)

operation, which does not trace to an implementation). Note that the design of the dedicated interpreters to feed the traceability model instance from various modeling tools is outside the scope of this paper, but others showed this is possible in the limited case of I* and UML (Cysneiros et al., 2003). Additionally, the use of traceability information through query, visualization, analysis, is outside the scope of this paper, although there is no reason to believe this would be much different than with other solutions. Our solution is based on our previous work, where we identified the requirements of a generic traceability model (Mustafa, 2015).

The rest of the paper is structured as follows: we discuss our solution, a traceability model, in section 2, justify our decisions, and argue that it brings generality (heterogeneity of traces) and extensibility (adapt to new models, new ways of characterizing traceability information) without requiring changes to the model itself, only its instantiation. Discussion of related work can be found in our previous publication (Mustafa, 2015) and is omitted from this paper due to space constraints. We then validate our solution in section 3, and conclude the paper in section 4.

2 THE TRACEABILITY MODEL

In this section we discuss our traceability model (section 2.2). Prior to this, we discuss the general context of use of our solution: section 2.1.

2.1 Context of Use

We are proposing a new traceability modeling solution to model traceability links between artifacts, while accounting for requirements we stated in the Introduction and in our previous work (Mustafa, 2015). We typically need to link artifacts that come from widely different sources. We also want to accommodate any taxonomy of traceability links engineers may want to use. The solution 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 traceability model itself. Only the instantiation (i.e., existing traceability links) would have to be changed (updated).

2.2 The Traceability Model Design

Our traceability model comprises classes that can accommodate the capturing of traceability information among heterogeneous artifacts. It includes: the *TraceabilityRoot* class (Figure 1), which acts as the root of the traceability model. Its purpose is to hold the traceability information about the artifacts under study. These include artifacts created during a specific software/system development. The *name* attribute is required to retrieve or store all the traceability information that the *TraceabilityRoot* instance holds (e.g., the attribute *name* can be the name of the software/system development which artifacts are being traced to one another). A *TraceabilityRoot* contains *TraceElements* (abstract class), which are either instances of *TraceLink*, *Trace*, or *Artifact*. The *TraceElement* class is associated to class *Characterization* to allow the characterization of any given trace element according to any taxonomy the

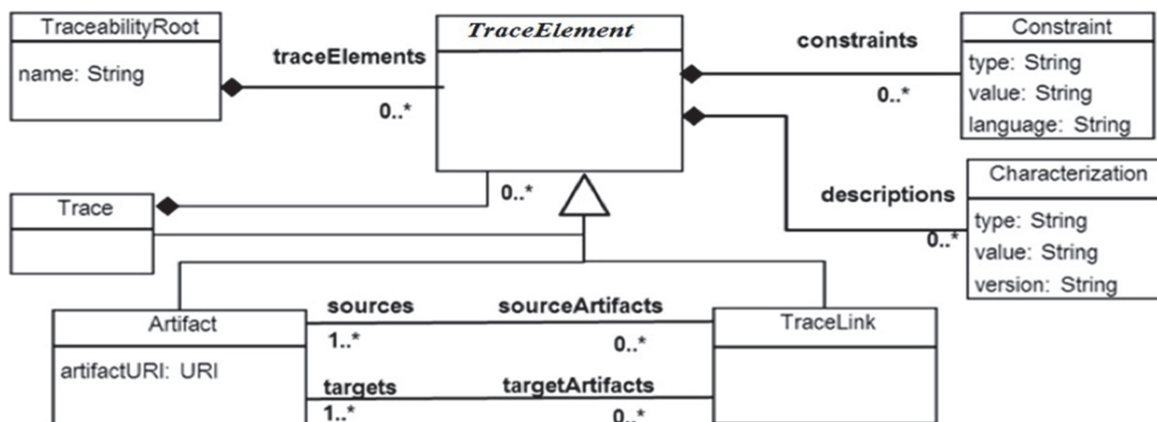

Figure 1: Generic Traceability Model.

user wishes to employ. These ideas are borrowed, merged, and integrated from previous works, such as (Ramesh and Edwards 1993; Drivalos et al., 2008; Anquetil et al., 2010; Paige et al., 2011), though not all come from a single of these works (we integrate previous solutions). A *Trace* represents a sequence (constraint *{ordered}* in the model) of chained trace elements, generated during a sequence of model transformations (e.g., as modeled by Falleri et al., (2006)), 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, under the form of an OCL constraint (not shown in the paper) that 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, the OCL constraint specifies that in case the *Trace* instance is a sequence of *TraceLink* instances, the target *Artifact* of the ith *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 ith and $(i+1)^{th}$ *Artifact* instances are the source and target of a *TraceLink* instance.

A *TraceLink* instance represents a traceability link between artifact(s): one or many source artifacts and one or many target artifacts. Note that some previous works limit those multiplicities to be strictly 1, though we believe 1..* brings more flexibility. Its purpose, thanks to (inherited) associations to *Characterization* and *Constraint*, is 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 (Ramesh and Edwards 1993), categories of traceability types specific to MDE software development (Paige et al., 2011).

We decided to exclude the 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 user can decide which taxonomies are important to their context. The drawback is that our solution may appear too generic or permissive: i.e., one can provide meaningless characterization. This can however be solved by requiring that *Characterization* attribute only take allowed values (in a specific set of taxonomies), which can be enforced by means of constraints (class *Constraint*).

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 document. The *resourceURI* attribute specifies the exact location of a traceable artifact, whether it is within a model, a file, or a 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 (Drivalos et al., 2008).

Artifact and *TraceLink* instances can be linked to zero or several *Constraint* instances in order to enforce some structural integrity of the model instance, i.e., of the traceability information (Drivalos et al., 2008; Paige et al., 2011). 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 specific, they cannot be all specified in the model and must be specified by the Engineer. 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 description is written to then allow an algorithm to trigger automatically the right constraints evaluation engine: e.g., the type value could equal "OCL" and the description could be an OCL expression. Our traceability model is more generic than previously published solutions since it can accommodate, by design, tracing artifacts that come from widely varying model types (i.e., class *Artifact* is not tied to any other metamodel), tracing new kinds of artifacts (because *Artifact* is not tied to any other metamodel), from possibly new kinds of models, and that those artifacts can be characterized in any

possible way the engineer sees fit (thanks to classes *Constraints* and *Characterization*). 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 structural integrity of the model instance. It is interesting to note that the level of complexity of our solution, for instance in terms of number of classes, associations and attributes, is similar to that of other solutions (e.g., (Drivalos et al., 2008; Anquetil et al., 2010)), though it is more generic and addresses our needs, contrary to those other solutions.

3 MODEL VALIDATION

Since our model is generic, its validation is not trivial because we cannot define a threshold for the required number of case studies to prove its generality. Therefore, we envisioned the validation in terms of four different criteria: (1) validity by construction which means justifying the reasons for the need of all our model elements (i.e., classes, associations, cardinalities) as we have already done in section 2.2; (2) showing that the existing traceability models fail to accommodate all the traceability requirements for our problem (Mustafa, 2015); (3) showing that our solution can model some representative examples collected from related works; and (4) showing that our solution can be applied in a realistic industry context.

With respect to (2), we propose several validation scenarios based on our design requirements and show that all existing models fail to satisfy some requirements (see requirements 1, 6, 7, 9, 10, 13, 14, 15, 19 in Table 1). The table shows that each existing solution only satisfies at most six of the 19 validation scenarios, and that together, all solutions only satisfy 10 of the 19 validation

SCIENCE AND TECHNOLOGY PUBLICATIONS

Table 1: Traceability model test cases and validation.

	Traceability Model Characteristics	***Test Case***	***Satisfied by***
1	Independent of languages, tools, frameworks	Check the characteristics of the existing models	None
2	Horizontal traceability between artifacts of heterogeneous models	Trace artifacts of heterogeneous models	(Paige, 2011); (Anquetil, 2010).
3	Horizontal traceability across phases within the same model.	Trace one requirement in one model to another requirement in another model.	(Paige, 2011); (Anquetil, 2010); (Pavalkis, 2008); (Drivalos, 2011); (Falleri,2006); (Cysneiros, 2003).
4	Vertical traceability (i.e., tracing artifacts within the same model or phase)	Trace one requirement in analysis phase to another requirement at the same phase.	(Pavalkis, 2008); (Drivalos, 2011); (Falleri,2006); (Cysneiros, 2003); (Paige, 2011); (Anquetil, 2010).
5	Trace cardinality between artifacts: 1-1	Trace a source requirement to a target test case	(Pavalkis, 2008); (Drivalos, 2011) (Falleri,2006); (Cysneiros, 2003); (Paige, 2011); (Anquetil, 2010).
6	Trace cardinality between artifacts: 1-M	Trace one source requirement to two target requirements that refine it.	None
7	Trace cardinality between artifacts: M-N	Tracing many to many artifacts	None
8	Bidirectional traceability	Trace a requirement to a use case and vice versa.	(Pavalkis, 2008); (Cysneiros, 2003); (Anquetil, 2010).
9	More than one characterization to a trace link	Use two orthogonal characterizations	None
10	More than one characterization to an artifact	Characterize an artifact by its type and location	None
11	Tracing artifacts of different granularities	Trace a requirement in a word file to a use case.	(Anquetil, 2010).
12	Constraint to a trace, artifact, or trace link	Apply an OCL constraint to a trace link	(Pavalkis, 2008); (Cysneiros, 2003); (Anquetil, 2010).
13	More than one constraint to a trace.	Apply two OCL constraints to a trace	None
14	More than one constraint to an artifact	Apply two OCL constraints to a requirement	None
15	More than one constraint to a trace link.	Apply two OCL constraints to a trace link.	None
16	Traceability information during model to model transformation.	Identify a trace (chained trace links or artifacts) in a sequence of model transformations.	(Falleri,2006).
17	Specifying the direction of the trace link.	Trace two artifacts A and B, where A is the source and B is the target or vice versa	(Pavalkis, 2008); (Falleri, 2006); (Paige, 2011); (Anquetil, 2010)
18	Prevents illegal links between certain artifacts.	Define a constraint to prevent specific links.	(Paige, 2011); (Anquetil, 2010).
19	Extensibility	Apply new characterization to a trace link by creating a new instance of the characterization class.	None

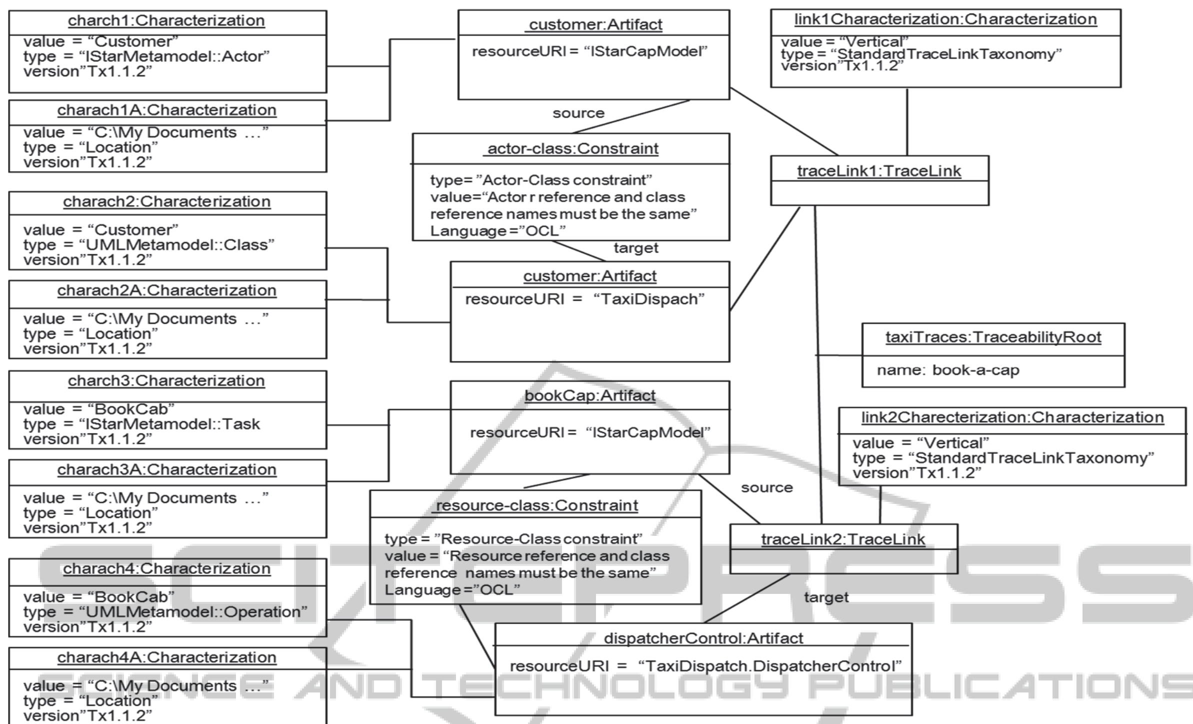

Figure 2: Traceability model instance for the example in Figure 3.

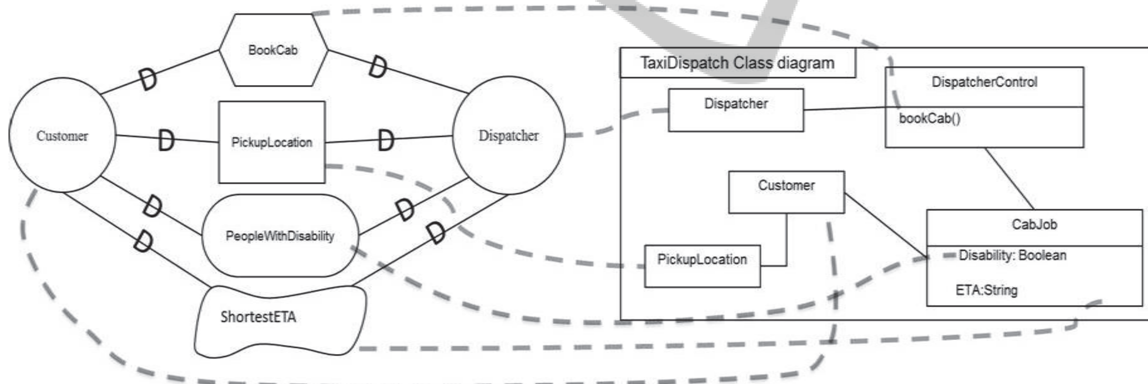

Figure 3: Traceability Example: I* (excerpt) model (left), UML (excerpt) class diagram (right), traceability links (grayed dashed lines).

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

scenarios. Since, as discussed later, our solution does satisfy all those scenarios, our solution is more than the mere combination of existing ideas. For the validation of our model against those unsatisfied requirements, we instantiated all the examples provided in the related work, sometimes adding to the original examples to illustrates more aspects of our solution. We used two examples from literature, and validated most of those scenarios (section 3).

The instantiations of the two examples of this paper show that our traceability model can accommodate the traceability requirements (1, 9, 10, 14, 19) of Table 1, which cannot be satisfied by the existing traceability models. It is not necessary to validate requirements 6, 7, 13, and 15 with an example since they are satisfied by construction: multiplicities in our model; and they are not satisfied by existing solutions (again by construction). Step (4) is under way, and there would anyway not be enough room in this paper due to size constraints to do that in addition to step (3). Note that "*None"* in Table 1 means that the existing models fail to satisfy the required scenario.

3.1 Case Study 1: Traceability between I* Model and UML Class

We use this example to validate the extensibility of our model by demonstrating how an artifact or a trace link can have different characterizations as wished by the engineer, and how different constraints can be applied to a source or target artifacts without changing the model. When tracing between an I* model and a UML class model (Paige et al., 2011), I* actors and resources trace to UML classes, I* goals, either hard or soft ones trace to UML class attributes, and I* tasks trace to UML class operations. We illustrate this using a cab dispatching system, whereby a customer wants a dispatcher to book a cab for a specific pick up location, possibly requesting a cab that can accommodate people with disabilities, and is interested in the expected time of arrival (ETA): see the I* excerpt diagram in Figure 3 (left), with I* actors *Customer* and *Dispatcher*, I* resource *PickupLocation*, I* goals *PeopleWithDisability* and *ShortestETA* (hard and soft goals, respectively).

Figure 3 (right) shows a UML class diagram with five different classes and some attributes and operations. The figure shows, as greyed out dashed lines between I* artifacts and UML class diagrams artifacts, the traceability links that need to be established according to Paige and colleagues (Paige et al., 2011). In addition, we defined the following constraints to be enforced on the artifacts and links of I* and UML models:

- The instance of *Attribute* (*Disability*) that is linked to an instance of *HardGoal* (*PeopleWithDisability*) must be of *Boolean* type to verify whether a hard goal is fulfilled or not.
- The name of an *Actor* in the I* model and the name of the linked *Class* in the UML model must be identical to ensure models are consistent.
- The name of a *Resource* in the I* model and the name of the linked *Class* in the UML model must be identical to ensure models are consistent.

(Note that, as per Table 1 such constraints cannot be specified with the solution of Paige and colleagues).

Figure 2 shows the instantiation of our traceability model for the traceability information in Figure 3. This is only an excerpt, because of size constraints, showing the traceability link of I* *Customer* actor to the UML class *Customer*, and the I* *BookCab* task to the UML *BookCab()*. The instantiation illustrates many important aspects of our traceability model. First, it can capture traceability information of heterogeneous models (i.e., I* and UML models). Second, it provides flexibility to the model's user to apply more than one characterization to an artifact, for instance, *customer* instance in Figure 3 is instantiated with two characterizations (see test case 10, Table 1). Third, the traceability model stayed unchanged, although the traceability requirements can vary based on application needs.

3.2 Case Study 2: Traceability in Model Transformation

This example is borrowed from Falleri and colleagues (Falleri et al., 2006) and demonstrates how our model handles traceability in case of model

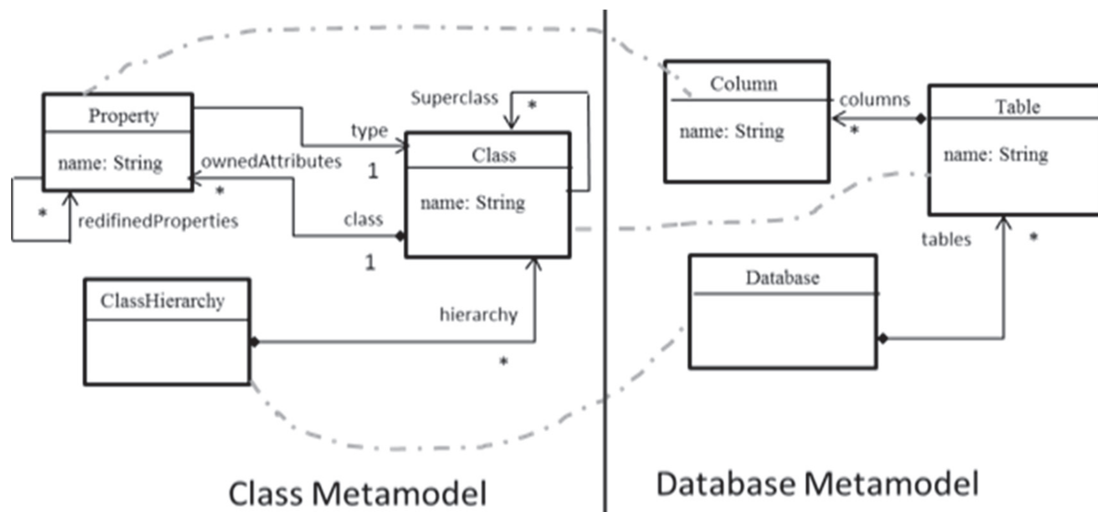

Figure 4: Metmodels of UML Class to Database mapping.

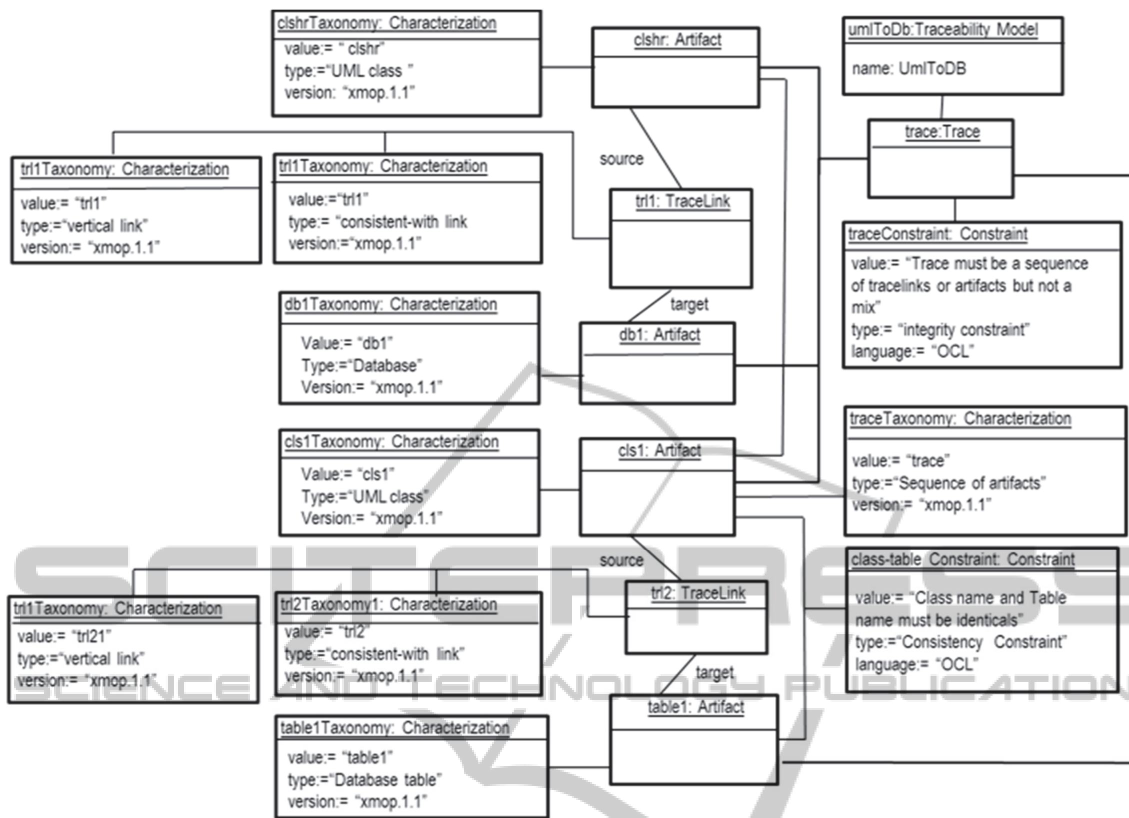

Figure 5: Traceability instance for the example in Figure 4.

transformation while applying constraints on the traces, specifically, from a UML class diagram to a database schema: Figure 4. The *Trace* class in our model is utilized to handle such a transformation. The greyed dashed lines in Figure 4 show a mapping between a UML metamodel and database metamodel. A class hierarchy in the UML model must transform into a database, a class in the UML model must transform into a database table, and a property in the UML model must transform into a database column. The following constraints must be enforced during the transformation in addition to the constraints on the *Trace* class mentioned in section 2: The name of a table that is transformed from a class should be the same as that class' name; The name of a table column that is transformed from a UML Property should be the same as this Property's name. In addition we identified the following trace links with characterizations that belong to orthogonal taxonomies: all the links are Vertical links since they link artifacts at two levels of abstraction; all the links are *Consistent-with* links since they ensure consistency between the two models.

Figure 5 shows an excerpt instantiation of our traceability model that corresponds to Figure 4. Note that the constraints are written in pure English for demonstration purposes. Also, the instance of the Trace class (trace) in this example represents an ordered set of artifacts. This example demonstrates different aspects about our model than the previous case study. First, it demonstrates how our model can accommodate model-to-model transformations using the *Trace* class (test case 16 in Table 1). Second, it allows the user to add constraints to an artifact (test case 14, Table 1). Third, it allows the user to identify multiple levels of granularity for a trace link characterization: e.g., the *Tracelink* instance *trl1* in Figure 5 can have coarse grained characterization (*vertical*) or fine grained characterization (*consistent-with*), (test case 9, Table 1).

4 CONCLUSIONS

Traceability in its simplest form is the ability to describe and follow the life of software artifacts (Winkler and Pilgrim 2010). 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 many models that specify a system must be oblivious of the solutions being used to model those artifacts. Additionally, we argued that the solution to model traceability should accommodate the situation where new artifacts, possibly in new models, need to be traced, where new ways of characterizing artifacts and traceability links need to be used. The solution to model traceability information should be flexible to accommodate many different situations.

We have proposed a traceability model that can address all those general issues, while at the same time incorporating ideas from many previous works on traceability modeling. We showed that this novel fusion of previous ideas, to satisfy specific requirements, is more than their sum. Indeed, we derived validation scenarios from the requirements and showed that each existing solution only satisfies at most six of the 19 validation scenarios, and that together, all solutions only satisfy 10 of the 19 validation scenarios.

Given the level of artifacts generality, validating our solution is a challenge. On the one hand, we argued that our traceability model addresses the requirements by design: we described and justified the classes, associations, and attributes in our model based on our knowledge of the literature and the problem we had to solve. One the other hand, we proceeded with respect to validation of our solution similarly to all the other traceability metamodeling techniques we have reviewed, that is, we used (two) instantiation examples we collected from the literature and enriched. In doing so we showed which aspect of our validation requirements are illustrated with which case study to cover as many cases as possible given space constraints. Future work will necessarily involve additional validation activities.

We have started is to systematically illustrate how our solution can accommodate the examples found in the literature (similarly to what we did in section 2.2). We will also have access to realistic traceability requirements from our industry partners.

REFERENCES

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

Amar, B., H. Leblanc and B. Coulette (2008). A Traceability Engine Dedicated to Model Transformation for Software Engineering. EC-MDA - *Traceability Workshop*.

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.

Cysneiros, F., A. Zisman and G. Spanoudakis (2003). Traceability approach for I* and UML models. *Int. Workshop on Soft. Eng. for Large-Scale Multi-Agent Systems*.

Drey, Z., C. Faucher, F. Fleurey, et al. (2014). "Kermeta language reference manual." Available at: http://www.kermeta.org/docs/fr.irisa.triskell.kermeta.documentation/build/pdf.fop/KerMeta-Manual/KerMeta-Manual.pdf. (Accessed 10 Sep 2014).

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

Espinoza, A., P. Alarcon and J. Garbajosa (2006). "Analyzing and Systematizing Current Traceability Schemas." *Software Engineering Workshop, Annual IEEE/NASA* pp. 21-32.

Falleri, J., M. Huchard and C. Nebut (2006). Towards a traceability framework for model transformations in kermeta. ECMDA - *Traceability Workshop*.

Gotel, O. and A. Finkelstein (1994). An Analysis of the Requirements Traceability Problem. *Proceedings of the Int. Conf. on Requirements Eng.*.

Kolovos, D., R. Paige and F. Polack (2008). Detecting and Repairing Inconsistencies Across Heterogeneous Models. *IEEE ICST*.

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

Mustafa, N. and Labiche, Y. (2015). 'Toward Traceability Modeling for the Engineering of Heterogeneous Systems', *Modelsward.*

Object Management Group. (2014a). "Business process Model Notation (BPMN)." Available at: http://www.omg.org/bpmn/Documents/BPMN_1-1_Specification.pdf. (Accessed 20 July, 2014).

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." SoSyM ***10***(4): pp. 469-487.

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*. Springer pp. 91-113.

Ramesh, B. and M. Edwards (1993). Issues in the Development of a Requirements Traceability Model. *IEEE Int. Symp. on Requirements Eng*.

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." *SoSyM* ***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.