Employing Linked Data in Building a Trace Links Taxonomy

Nasser Mustafa and Yvan Labiche
Department of Systems and Computer Engineering, Carleton University, 1125 Colonel By Dr, Ottawa, Canada

Keywords: Traceability, Trace Links, Semantics, Taxonomy, Requirement Engineering, Systems Engineering, Model Driven Engineering, Linked Data, Resource Description Factor, Open Service for Lifecycle Collaboration.

Abstract: Software traceability provides a means for capturing the relationship between artifacts at all phases of software and systems development. The relationships between the artifacts that are generated during systems development can provide valuable information for software and systems Engineers. It can be used for change impact analysis, systems verification and validation, among other things. However, there is no consensus among researchers about the syntax or semantics of trace links across multiple domains. Moreover, existing trace links classifications do not consider a unified method for combining all trace links types in one taxonomy that can be utilized in Requirement Engineering, Model Driven Engineering and Systems Engineering. This paper is one step towards solving this issue. We first present requirements that a trace links taxonomy should satisfy. Second, we present a technique to build a trace links taxonomy that has well-defined semantics. We implemented the taxonomy by employing the Link data and the Resource Description Framework (RDF). The taxonomy can be configured with traceability models using Open Service for Lifecycle Collaboration (OSLC) in order to capture traceability information among different artifacts and at different levels of granularity. In addition, the taxonomy offers reasoning and quantitative and qualitative analysis about trace links. We presented validation criteria for validating the taxonomy requirements and validate the solution through an example.

1 INTRODUCTION

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 Requirement Engineering (RE), Model Driven Engineering (MDE), and Systems Engineering (SE). They are heterogeneous in nature since they are produced by different tools, and for different system disciplines. Establishing relationships between these artifacts requires different types of trace links with 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 adverse effect on product quality (Ramesh, B. and M. Jarke, 2011).

Our aim is to build a trace links taxonomy which has well-defined semantics and that encompasses various types of trace links in the RE, MDE, and SE disciplines. This is important for many reasons. First, in RE, many artifacts are produced during requirements elicitation, analysis, and validation, hence, require different types of trace links with different semantics. Second, in MDE, which permits model transformations, a large number of trace links is 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, which are slightly different from what one can define in RE or SE. Third, in SE, 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

Mustafa, N. and Labiche, Y.
Employing Linked Data in Building a Trace Links Taxonomy.
DOI: 10.5220/0006471701860198
In *Proceedings of the 12th International Conference on Software Technologies (ICSOFT 2017)*, pages 186-198
ISBN: 978-989-758-262-2

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 (Ramesh, B. and M. Jarke, 2011). Finally, with the various types of modeling tools across different domains, it is a necessity to have a trace links taxonomy that can be integrated with other API's. In other words, we need a portable taxonomy that can be integrated easily with other tools.

In an effort to have more insight about trace links and their classifications we conducted a systematic literature review about traceability aspect in which trace links is among them (Nasser Mustafa and Yvan Labiche, 2017). The review covers the published papers between the years 2000-2016 in five major computing libraries (i.e., IEEE Xplore, ACM, Google Scholar, Science Direct, and Springer). We specified the following search string in order to extract the traceability publications in RE, MDE, and Systems Engineering: Traceability AND (Heterogeneous OR Modeling OR Models OR MDE OR Model Driven OR Trace Link OR Requirement Engineering OR Systems Engineering OR Software Engineering). Based on our review, we identified some research papers that define traceability and traceability relations (Ramesh, B. and M. Jarke, 2011; Spanoudakis, G. and A. Zisman, 2005; Gotel, O. and A. Finkelstein, 1994; Nasser Mustafa, Yvan Labiche, 2015; Mason, P., et al., 2003; IEEE, 1990; Cleland-Huang, et al., 2014; Gotel, O., et al., 2012; Ramesh, B. and M. Edwards, 1993; Aizenbud-Reshef, N., et al., 2006; Nasser Mustafa, Yvan Labiche, 2015), other papers that classify or identify some types of trace links (Ramesh, B. and M. Jarke, 2011; Spanoudakis, G. and A. Zisman, 2005; Gotel, O. and A. Finkelstein, 1994; Spanoudakis, G., et al., 2004; Xu, P. and B. Ramesh, 2002; Pohl, K., 1996; Alexander, I., 2003; Riebisch, M. and I. Philippow, 2001; Mason, P., et al., 2003; Cleland-Huang, J., et al., 2014; Gotel, O., et al., , 2012; Paige, F., et al., 2008; Mohan, K. and B. Ramesh, 2002; Maletic, J. I., et al., 2003; Gotel, O. and A. Finkelstein, 1995; Constantopoulos P., et al., 1993; Pinheiro, F. A. C. and J. A. Goguen, 1996; Grammel, B., 2014; Olsen, G. K. and J. Oldevik, 2007; Paige, R. F., et al., 2011), and some papers that discuss the need for trace links semantics (Paige, R. F., et al., 2011; Letelier, P., 2002; Dick, J., 2002; Lucia, A. D., et al., 2007; Rummler, A., 2007). 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 trace links from all domains. 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 RE (Ramesh, B. and M. Jarke, 2011; Spanoudakis, G. and A. Zisman, 2005), though classifications only apply to RE.

The contribution of this paper includes the followings. First, we propose requirements for trace links taxonomy. Second, we offer a technique to build a trace links taxonomy which has well-defined semantics and that can accommodate the classification of trace links in RE, MDE, and SE. The taxonomy employs the Open Service for Lifecycle Collaboration (OSLC), and the Resource Description Framework (RDF) (W3C, 2016(a)) for defining a set of properties and their values for each trace link. Third, we validate the taxonomy through a case study that requires heterogeneous artifacts from multiple domains.

This paper is structured as follows. Section 0 discusses an example that will help us illustrate the motivation behind this work. Section 0 presents related work on trace links and their limitations. Section 0 highlights the requirements for trace links taxonomy and introduces the RDF technique. Section 0 shows our proposed taxonomy requirements. Section 0 describes the benefit of using RDF in building the trace links taxonomy. Section 0 shows our design decisions and the taxonomy implementation using the RDF technique on a case study. Section 7 concludes the paper.

2 A 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 RE, MDE, and SE 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 example for relating the i* metamodel artifacts, which capture *early-phase requirements,* and the *UML* Class metamodel which

captures *late-phase requirements* (Filho, G. C., et al., 2003). It involves different types of artifacts that require certain types of trace links. 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 to the *Class* metaclass in the *UML* metamodel. Also, the *SoftGoal* and *HardGoal* in the *i** metamodel are mapped to the *Attribute* in the *UML* metamodel. Finally, a *Task* in the *i** metamodel is mapped to the *Operation* in the *UML* metamodel. Determining the relationship between instances from the two metamodels depends on users' needs: two different users might need two different types of trace links. For instance, a user might use the *Consistent-with* trace link between instances of the *Actor* class and the *Class* class since each *Actor* instance in the *i** model must have a corresponding *Class* instance in the UML model. A second user might be interested in a high level view for the two metamodel instances, so the generic *model-to-model* or *static* trace links can be used. In this example, different users require trace links at different levels of granularity. This scenario and others encouraged us to build a trace links taxonomy that combines all trace links across different domains and shows their relationships.

3 RELATED WORK

This section elucidates important aspects about our review of traceability in RE, MDE, and SE. Section 3.1 discusses traceability and trace links definitions while section 0 discusses existing trace links classifications in RE, MDE, and SE. The review in this section is extensive since it will be used as a core for our work in order to collect all trace links types for building the taxonomy.

3.1 Traceability Definitions

Traceability is defined by the IEEE (IEEE, 1990) 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 RE, MDE, and SE as well. The IEEE definition is extended to include other types and subtypes of relationships. Cleland-Huang and colleagues (2014) 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*". The *derived* represents the meaning of the relation between such artifacts. Therefore, we might have similar or extended types of trace links among the RE, MDE, and SE 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 (Cleland-Huang, J., 2014).

In RE, several types of trace links are introduced as a result of traceability definitions. Gotel and colleagues (1994) defined traceability as the ability to describe and follow the life of a requirement in both forward and backward directions. In this context, a pre-requirement specification refers to the aspects of a requirement's life prior to its inclusion in the requirement specification, and a post-requirement specification refers to the aspects of a requirement's life that result from its inclusion in the requirement specification (Gotel, O., et al., 2012). Also, there are the notions of vertical and horizontal traceability (Spanoudakis, G. and A. Zisman, 2005; Cleland-Huang, J., 2014; Ramesh, B. and M. Edwards, 1993; LindVall, M. and K. Sandahl, 1996). 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 MDE, Aizenbud-Reshef and colleagues (2006) defined traceability as "any relationship that exists between artifacts involved in the software-engineering life cycle". This definition is broader than the RE definition since it assumes other types of trace links such as explicit links which can be generated during model transformation, implicit links which are computed based on existing information, and statistical links that can be inferred based on history.

In SE, Mason (Mason, P., et al., 2003) extended the notions of vertical and horizontal traceability by introducing the terms: Micro, Macro, Inter, and Intra. The Micro and Macro terms are introduced to differentiate traceability within and across decomposition levels. The Intra and Inter terms are introduced to 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.

3.2 Traceability Classifications

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 RE, MDE, and SE 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 requirement and source code), or target one domain only (e.g., RE or MDE). This section summarizes our effort in collecting and organizing these classifications in order to build a trace links taxonomy.

In RE, relationship between artifacts are discussed extensively (Ramesh, B. and M. Jarke, 2011; Spanoudakis, G. and A. Zisman, 2005; Gotel, O. and A. Finkelstein, 1994; Spanoudakis, G., et al., 2004; Xu, P. and B. Ramesh, 2002; Pohl, K., 1996; Alexander, I., 2003; Riebisch, M. and I. Philippow, 2001; Cleland-Huang, J., et al., 2014; Gotel, O., et al., , 2012; Paige, F., et al., 2008; Mohan, K. and B. Ramesh, 2002; Maletic, J. I., et al., 2003; Gotel, O. and A. Finkelstein, 1995; Constantopoulos P., et al., 1993; Pinheiro, F. A. C. and J. A. Goguen, 1996; Kozlenkov, A. and A. Zisman, 2002); among these papers, we found two papers that discuss trace links classifications (Ramesh, B. and M. Jarke, 2011; Spanoudakis, G. and A. Zisman, 2005). Spanoudakis and Zisman 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. These links include the following types:

- The dependency links which 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) such as decision objects. Dependency relations are one of the most widely used in RE and have different uses and forms (Spanoudakis, G. and A. Zisman, 2005). For instance, Xu and Ramesh (Xu, P. and B. Ramesh, 2002) use dependency relations in workflow management systems between business process objects, decision objects, and workflow system objects. Dependency relations are used in product and service families (Mohan, K. and B. Ramesh, 2002) to support the management of variability, i.e., ensuring that the changed artifacts reflect the intended system functionality. Knethen and colleagues (Knethen, 2002) 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 (Pohl, K., 1996; Alexander, I., 2003) use them to link requirement scenarios and code, and Riebisch and Philippow (2001) 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 (Spanoudakis, G., et al., 2004) 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. (Maletic, J. I., et al., 2003) who use them to link documents that represent an implied ordering (e.g., bug reports cannot be produced before implementation report). Gotel and colleagues (Gotel, O. and A. Finkelstein, 1995) 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. (1993) as correspondence relations which link requirements, design, and code artifacts.
- The evolutionary relations are used to link requirements in which one requirement replaces another. This category contains the replace, based-on, formalize, and elaborate trace links. Pinheiro (Pinheiro, F. A. C. and J. A. Goguen, 1996) showed the use of replace and abandon trace links during requirements evolution. A requirement is replaced by another if a mistake is discovered, the original requirement will be abandoned. Gotel (Gotel, O. and A. Finkelstein, 1995) called the evolution relations temporal relations which

Table 1: Trace links classifications in RE, MDE, and SE.

Ref.	Requirement Engineering Classifications									
Ramesh, B. and M. Jarke, 2011	Product- related			Process –related						
	Evolution	Rationale	Dependency	Satisfaction						
	Derive, Elaborate, Depend-on	Select, Affect	Is-a, Part-of, Contain, Used-by, Performed-by	Define, Allocate-to, Depend-on, Created-by, Verify, Generate						
Spanoudakis, G. and A. Zisman, 2005			Dependency	Evolution: Replace, Based-on, Formalize, Elaborate	Generalize/ Refine	Satisfaction	Overlap	Conflict: Based-on, Affect, Resolve, Generate	Rationale	Contribution
Other RE References (using the same name or a different names)										
Gotel, O. and A. Finkelstein, 1994										X
Spanoudakis, G., et al., 2004			Requires-feature-in				X			
Xu, P. and B. Ramesh, 2002			X						X	
Pohl, K., 1996			X							
Alexander, I., 2003			X							
Riebisch, M. and I. Philippow, 2001			X							
Maletic, J. I., et al., 2003			Causal-dependency conformance	Non-causal conformance						
Pinheiro, F. A. C. and J. A. Goguen, 1996						Satisfy / Derive / Refine				
Gotel, O. and A. Finkelstein, 1995			Developmental	Temporal	Containment		Adopt			
Constanto-poulos P., et al., 1996			Correspondence							
Letelier, P., 2002									X	
Dick, J., 2002						Satisfy / Establish / Contribute				
Knethen, 2002								Inconsistency		
Filho, G. C., et al., 2003							X			

Table 1: Trace links classifications in RE, MDE, and SE (cont.).

<table>
<tr><th>Ref.</th><th colspan="8">Model Driven Engineering Classifications</th></tr>
<tr><td rowspan="3">Paige, F., et al., 2008</td><td>Implicit</td><td colspan="7">Explicit</td></tr>
<tr><td rowspan="2"></td><td colspan="5">Model-to-model</td><td colspan="2">Model-to-artifact</td></tr>
<tr><td colspan="3">Static</td><td>Dynamic</td><td colspan="2" rowspan="2">Satisfy, Allocated-to, Explain, Perform, Support</td></tr>
<tr><td colspan="2">Consistent-with</td><td>Dependency</td><td>Call, Notify, Generate</td></tr>
<tr><td colspan="2"></td><td colspan="2">Export, Usage, Is-a, has-a, Part-of, Import Refine</td><td colspan="3"></td></tr>
<tr><td colspan="9">Systems Engineering Classifications</td></tr>
<tr><td rowspan="3">Mason, P., et al., 2003</td><td rowspan="3">Temporal</td><td colspan="8">Directional</td></tr>
<tr><td colspan="4">Vertical</td><td colspan="4">Horizontal</td></tr>
<tr><td colspan="2">Micro</td><td colspan="2">Macro</td><td colspan="2">Micro</td><td colspan="2">Macro</td></tr>
<tr><td>Intra</td><td>Inter</td><td>Intra</td><td>Inter</td><td>Intra</td><td>Inter</td><td>Intra</td><td>Inter</td></tr>
</table>

which refer to linking requirements in terms of their historical order. Maleic and colleagues (Maletic, J. I., et al., 2003) 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 (Ramesh, B. and M. Jarke, 2011) generalization/refinement is considered a dependency abstraction link. Gotel (Gotel, O. and A. Finkelstein, 1995) 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 (Ramesh, B. and M. Jarke, 2011). Satisfiability has sub-types such as the establish (cardinality 1-1 between two artifacts) and contribute (cardinality 1-m between artifacts) relations (Dick, J., 2002). Pinheiro (1996) 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. between artifacts of common features (e.g., linking a goal specification in an i* model and a use case in a UML model) (Filho, G. C., et al., 2003). Spanoudakis et al., (Spanoudakis, G., et al., 2004) use the overlap relations in an analysis model between use cases and classes. Gotel and Finkelstein (1995) 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 (Ramesh, B. and M. Jarke, 2011). 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 (2002) referred to conflict relations as inconsistency relations. For instance, inconsistency relations are established when two similar goals in a specification or different specifications cannot be achieved.
- The rationalization relations link two artifacts in which one of them captures the rationale behind the creation or evolution of the other. Letelier (2002) 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 (Xu, P. and B. Ramesh, 2002).
- The Contribution relations relate requirements and their stakeholders (Gotel, O. and A. Finkelstein, 1994), for instance to link requirements to the stakeholders who contributed them.

Another classification for trace links in RE is introduced by Ramesh and Jarke (2011). Their

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 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 of operations performed in a process. The product-related links describe the relationships between artifacts independent 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 described earlier. The authors deduced other types of relations from the abovementioned categories based on the use of low-end and high-end users. For instance, with respect to low-end users', the original or derived requirements can be allocated-to system components that interface-with external systems. Also, requirements can be developed-for compliance-verification-procedures (e.g., test, simulation, and prototype). Compliance-verification-procedures generate change proposal or used-by resources. With respect to high-end users, traceable artifacts (e.g., requirements, components, designs) are based-on a rationale. Decisions depends-on assumptions, or select or evaluate alternatives. Also, decisions may affect requirements, and arguments oppose or support alternatives.

In MDE trace links are generated explicitly by adding additional code into the transformation, or implicitly through the transformation tool (Olsen, G. K. and J. Oldevik, 2007). Paige and colleagues (2008) classified MDE trace links into implicit and explicit trace links. The implicit trace links are classified based on query, transformation, composition (merging), update, deletion and creation, model-to-text, and sequences operations. The explicit trace links are classified as model-to-model links which relate MDE artifacts with each other, and model-to-artifact which relate MDE artifacts with non-MDE 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 models 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 calls (e.g., a model calls the behaviors provided by another), notifies (e.g., changed artifacts that need intervention), and generates (e.g., links two models where one model produces the other).

The model-to-artifact category contains the satisfies trace link which indicate 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. In SE, Mason et al. (2003) 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 the links between synchronized artifacts, for instance, linking an artifact and its subsequent revised one in a model that based on an event. We summarize the trace links classifications in Table 1.

As evidenced by the above discussion, existing classifications of trace links have the following drawbacks:

- Each classification is either problem specific or domain specific.
- 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 (2005) classified is-a as an evolution link, while Paige and colleagues (2008) classify it as dependency link.
- Classifications are redundant, which we conjecture is also a side effect of the first drawback. For instance, the rationale trace

links appear in RE, MDE, and SE classifications.
- Classifications don't 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 all classifications to be used in all 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.

4 TAXONOMY REQUIREMENTS

The abovementioned limitations encouraged us to think about a new method for integrating all trace links classifications into a taxonomy that would provide the relationship between different trace links. In the light of the previous discussion, the new taxonomy shall have the following characteristics, or taxonomy requirements (TRQ):

TRQ 1: the taxonomy shall provide semantic specifications for trace links that relate various artifact 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), and therefore different domains.

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 properties of trace links without changing the existing structure of the taxonomy.

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 tool integration.)

TRQ 6: the taxonomy shall have a universal format that is not tailored to a specific environment or application.

5 EMPLOYING THE RDF IN BUILDING THE TAXONOMY

We propose a trace link taxonomy that integrates all existing trace link classifications into a structure to satisfy the requirements proposed in Section 0. The taxonomy utilizes the OSLC and the RDF (W3C, 2016(a)) 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 diagram, 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 turns can be a subject that has other properties, which form nested subjects. RDF files are written using the RDF/XML format which is a common format on the web.

The rationale for employing RDF in creating a trace links 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.
- 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 an 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 Software Engineering and Systems Engineering.

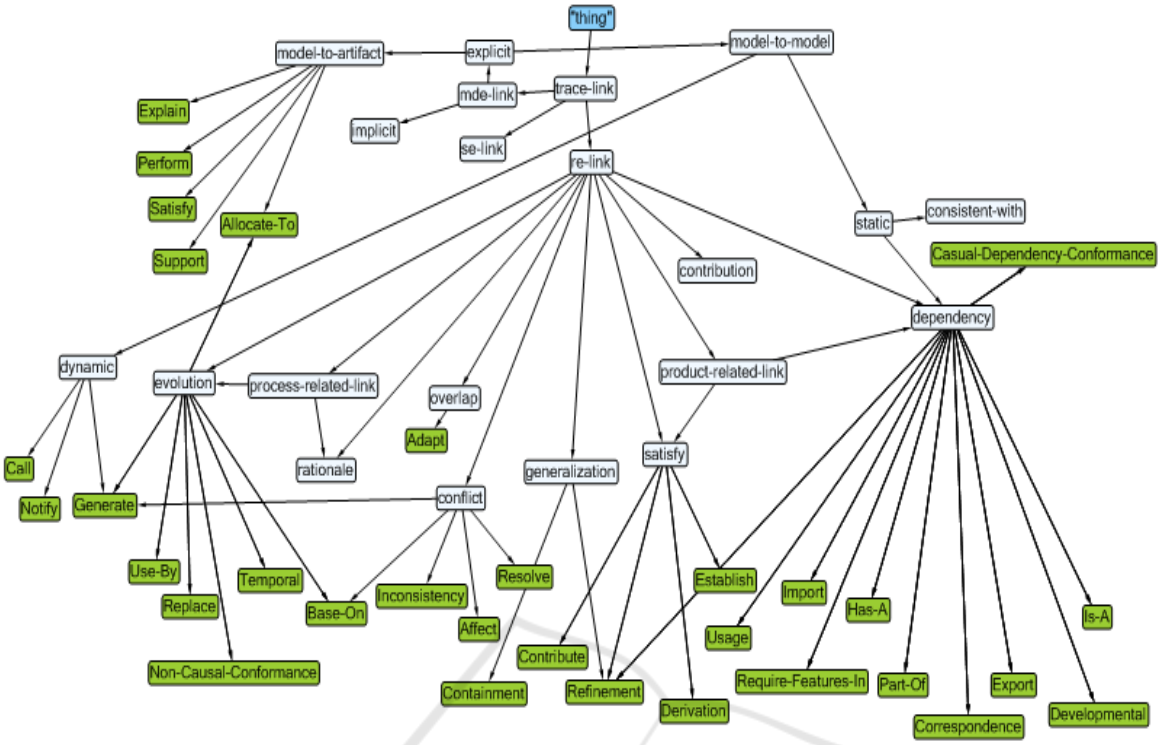

Figure 1: The trace links taxonomy (excerpt).

- 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 (W3C, 2016(c)); the query can be customized based on a user's needs.

6 TAXONOMY DESIGN AND IMPLEMENTATION

There are two essential components that should exist in order to build our trace links taxonomy: (a) provide a set of controlled vocabulary (Metadata), the controlled vocabulary 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. An Ontology can be defined using the Web Ontology Language (OWL) (W3C, 2016(b)) which is an extension of RDF. Many organizations standardized their controlled vocabulary and made it available freely for use on the net such as the Friend of a Friend (FOAF) (Miller, L. and D. Brickley, 2016) which has standard vocabulary/Ontology for social networks across the web, and the Description of a Project (DOAP) (Dumbill, E., 2016) for describing open source software projects.

6.1 Taxonomy Design

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

- Identify all articles that discuss trace links classification in RE, MDE, and SE.
- Identify the terms that describe general types of trace links. Usually, these are nouns or adjectives that describe the relationship between artifacts such as dependency, evolution, and vertical.
- Identify all terms that describe the relationship between specific types of artifacts. These relationships are identified by the role name of the association between artifacts. They are usually represented as verbs such as perform, generate, and depend-on.

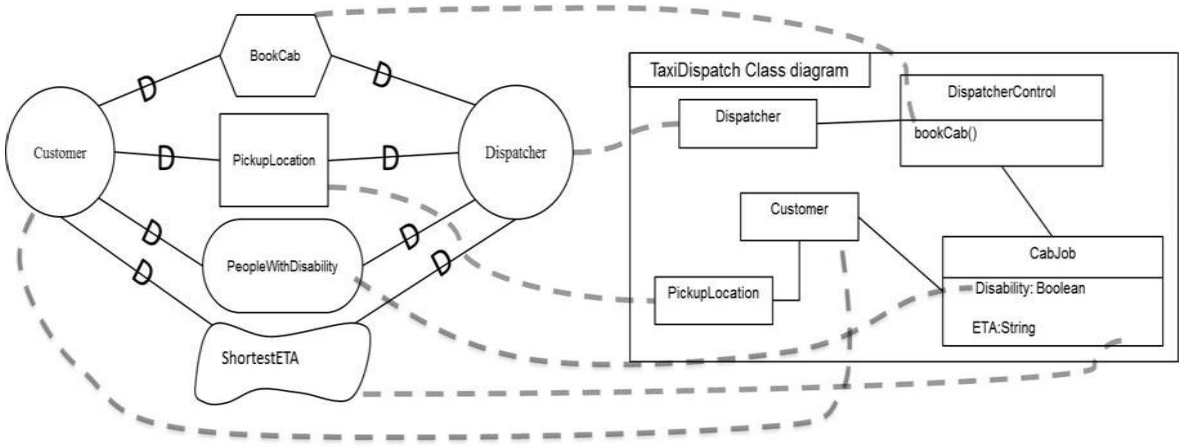

Figure 2: Traceability Example: I* (excerpt) model (left), UML (excerpt) class diagram (right), traceability links (greyed dashed lines) (Ramesh, B. and M. Jarke, , 2001).

- We consider the terms that represent general types as classes, and the terms that represent relationships between specific artifacts as instances or leaves in our taxonomy. For instance, evolution is a class that represents a general type of a relationship; derive, elaborate, and depend-on are instances of this class.
- Provide a naming convention for the general types and the relationships. We have done that by screening all the conjugations 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), we choose to call it evolution. Moreover, we considered the terms perform, performs, and performed as identical terms that refer to a specific relationship (i.e., instance) and we call it to perform.
- We also provide a set of properties for *instances*. Each i*nstance* must have unique values that differentiate it from other instances. We limit the properties to include: Id, *name*, *usage*, *type*, and *definition,* however, other properties can be added. The *name* represents the *instance name*, and the type represents the *Class name*.

6.2 Taxonomy Implementation

We build the taxonomy by following the previous steps and employing the RDF. We used the Fluent editor application (Cognitum, 2015) 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 formats such as RDF, XML, and OWL. In addition, it supports building and visualizing ontologies as interactive diagrams or trees. Finally, the application allows for integrating ontologies with the R Language (R. Foundation, 2017), in which quantitative and qualitative analysis can be performed.

The taxonomy provides a global view for all trace links across the RE, MDE, and SE domains. It shows the relationships between all trace links in these domains. The diagram in Figure 1 depicts an excerpt of the taxonomy, it shows partial classification of the RE and MDE trace links; we could not provide the complete taxonomy here since it occupies a big space and it is very hard to visualize all the trace links connections. On the top centre of the diagram, we can see the root of all elements in the taxonomy which is represented by the word "*thing"* which in turn is connected to *a trace-link* Class. A *trace-link* is a general type that has three sub-types i.e., re-*link, mde-link, and se-link* for the respective RE, MDE and SE domains. Following the path of any type, we reach the leaves which represent the links between artifacts. 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 the diagram, the *Allocate-to* trace link belongs to *model-to-artifact and to evolution* categories. This design eliminates trace links redundancy.

6.3 Taxonomy Validation

The taxonomy can be validated by (a) ensuring that it satisfies the taxonomy requirements that we proposed in section 0 in order to resolve the issues about existing classifications, and (b) it can accommodate any traceability problem.

Regarding part (a) we proposed the following validation criteria in order to ensure that all requirements can be met:

- TaxCr 1. Trace Links redundancy. This criterion validates whether a trace link exists more than once in the taxonomy. It should only appear once.
- TaxCr 2. The capability of the taxonomy to accommodate the classification of trace links related to RE, MDE, and SE.
- TaxCr 3. Consistency. This Criterion validates whether a trace link has only one definition. This can be validated also by
- TaxCr 1.
- TaxCr 4. Extensibility and maintainability. This criterion validates how easy a trace link or a property can be added to the taxonomy without changing the existing design.
- TaxCr 5. Tool Support. This criterion validates whether the taxonomy data can be saved, exported to other formats or applications, or performing queries about it. We found out that all requirements are satisfied.

With respect to part (b), we validate the taxonomy using the example stated in section 0 and depicted in Figure 2. However, the validation will not be complete since the purpose of the taxonomy is to be configured with a traceability model, for instance (Nasser Mustafa and Yvan Labiche, 2015). The links between the instances of *i** and *UML-Class* metamodels can have different semantics based on the type of the traced artifacts, consequently, the trace links can have different classifications based on the following scenarios:

1) Identify the instances (artifacts) in the two metamodels that relate early-phase requirements to late-phase requirements.
2) Identify the instances in the i* metamodel that capture the (why) and relate them to the corresponding instances in the *UML-Class* metamodel.
3) Identify the instances of the two metamodels that must have identical names.
4) Identify the instances of the i* metamodel that must be mapped to *Classes* in the *UML-Class* metamodel.
5) Identify the *Tasks* in the i* metamodel that must have a correspondence method in the *UML-Class* metamodel.
6) Identify the instances of the i* metamodel that are satisfied by the instances of the *UML-Class* metamodel.

We have shown in Table 2 some instances from both metamodels. Based on the generated artifacts, we have shown the trace link type as configured by our taxonomy.

Table 2: Configuring trace links between instances of the i* Actor and UML Class Metaclasses.

No	i* artifacts (source)	UML artifacts (target)	Trace link
1	i* model	UML-Class model	re-link
2	Actor :Dispatcher	Class: Dispatcher	rationale
3	Actor: Customer	Class: Customer	Consistent-with
4	Resource: PickupLocation	Class: PickupLocation	Consistent-with
5	Task: BookCab	Method: bookCab	Realize
6	HardGoal: PeopleWithDisability	Attribute: Disability	Satisfy

7 CONCLUSION AND FUTURE WORK

We have 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 external resources (taxonomy elements) to form a flat hierarchical structure. We implemented the taxonomy using the RDF technique, which allows for 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.

As a future work, this taxonomy can be extended by adding more trace link properties and trace links 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 links data then can be filtered and edited. We believe this taxonomy can be used as a base for standardizing trace links semantics. Future work should, of course, consider using our taxonomy so increase our confidence that it addresses needs as planned.

REFERENCES

Ramesh, B. and M. Jarke, *Toward Reference Models for Requirements Traceability*. IEEE Trans. Softw. Eng., 2011. 27(1): p. 58-93.

Spanoudakis, G. and A. Zisman, *Software Traceability: A road map*, in *Handbook of Software Engineering and Knowledge Engineering*, S.K. Chang, Editor. 2005. p. 395-428.

Gotel, O. and A. Finkelstein. *An Analysis of the Requirements Traceability Problem*. in *1st International Conference on Requirements Engineering*. 1994. Utrecht, The Netherlands.

Spanoudakis, G., et al., *Rule-Based Generation of Requirements Traceability Relations* Systems and Software, 2004. 72(2): p. 105-127.

Xu, P. and B. Ramesh, *Supporting Workflow Management Systems with traceability*. in *35th Annual Hawaii International Conference on System Sciences*. 2002. Hawaii: IEEE.

Pohl, K. *PRO-ART: Enabling Requirements Pre-Traceability*. in *2nd IEEE International. Conference on Requirements Engineering* 1996: IEEE Computer Society.

Alexander, I., *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. Canada.

Nasser Mustafa, Yvan Labiche, "The Need for Traceability in Heterogeneous Systems: A systematic literature review". (Accepted), IEEE COMPSAC, Italy, 2017.

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

Nasser Mustafa, Yvan Labiche, "Modeling Traceability for Heterogeneous Systems", ICSOFT. 10 international joint conference on Software Technologies. Colmar, Alsace, France, 2015.

Mason, P., et al., *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. Huntsville, AL, USA: IEEE.

IEEE, *IEEE Standard Glossary of Software Engineering Terminology*, in *IEEE Standard Glossary of Software Engineering Terminology*, I.s. board, Editor. 1990: New York.

Cleland-Huang, J., O. Gotel, and A. Zisman, eds. *Software and Systems Traceability*. ed. Z. A. 2014, Springer.

Gotel, O., et al., *Traceability Fundamentals*. Software and Systems Traceability 2012.

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

Aizenbud-Reshef, N., et al., *Model traceability.* IBM Systems Journal - Model-driven software development, 2006. 45(3): p. 515-526.

Nasser Mustafa, Yvan Labiche, "Towards Traceability Modeling for the Engineering of Heterogeneous Systems", Model-Driven Engineering and Software Development (MODELSWARD), France, 2015.

Paige, F., et al., *Building Model-Driven Engineering Traceability Classifications*. in *European Conference on Model Driven Architecture - Traceability Workshop* 2008. Berlin, Germany.

Mohan, K. and B. Ramesh, *Managing Variability with Traceability in Product and Service Families*. in *35th Annual Hawaii International Conference on System Sciences* 2002. Hawaii: IEEE.

Maletic, J. I., et al. *Using a Hypertext Model for Traceability Link Conformance Analysis* in *2nd International Workshop on Traceability for Emerging Forms of Software Engineering* 2003. Canada.

Gotel, O. and A. Finkelstein, *Contribution Structures*. in *2nd International Symposium on Requirements Engineering*. 1995: IEEE.

Constantopoulos P, J. M., Mylopoulos Y., Vassiliou Y., " *The Software Information Base: A Server for Reuse.* The International Journal on Very Large Data Bases, 1993. 4(1): p. 1-43.

Pinheiro, F. A. C. and J. A. Goguen, *An Object-Oriented Tool for Tracing Requirements.* IEEE Software 1996. 13(2): p. 52-64.

Grammel, B., *Automatic Generation of Trace Links in Model-driven Software Development*, in *Fakultät Informatik*. 2014, Technische Universität Dresden.

Olsen, G. K. and J. Oldevik, *Scenarios of Traceability in Model to Text Transformations*. in *3rd European conference on Model driven architecture-foundations and applications*. 2007. Haifa-Israel: Springer-Verlag.

Paige, R. F., et al., *Rigorous identification and encoding of trace-links in model-driven engineering.* Software & Systems Modeling, 2011. 10(4): p. 469-487.

Letelier, P., *A Framework for Requirements Traceability in UML-based Projects*. in *1st Intl. Workshop on Traceability in Emerging Forms of Softw. Eng*. 2002.

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

Lucia, A. D., F. Fasano, and R. Oliveto, *Recovering Traceability Links in Software Artifact Management Systems using Information Retrieval Methods.* ACM

Transactions on Software Engineering and Methodology, 2007. 16(4).

Rummler, A., B. Grammel, and C. Pohl, *Improving Traceability in Model-Driven Development of Business Applications* in *European Conference on Model Driven Architecture - Traceability Workshop* 2007.

OMG, O.M.G. *Open Services for Lifecycle Collaboration*. 2017 [cited 2017 Feb 20]; Available from: https://open-services.net/.

W3C, *Resource Description Framework*. 2016 (a) [cited 2016 Oct 15]; Available from: https://www.w3.org/RDF/.

Knethen, A.v. *Automatic Change Support Based on a Trace Model*. in *1st International Workshop on Traceability in Emerging Forms of Software Engineering* . 2002. Edinburgh.

Filho, G. C., 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. Portland.

LindVall, M. and K. Sandahl, *Practical Implications of Traceability.* Software Practice & Experience, 1996. 26(10): p. 1161-1180.

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

W3C, *Web Ontology Language*. 2016 (b) [cited 2016 Nov 3rd]; Available from: https://www.w3.org/TR/owl-features/.

W3C, *SPARQL Query Language for RDF*. 2016 (c) [cited 2016 Nov 3rd]; Available from: https://www.w3.org/2001/sw/DataAccess/rq23/.

Miller, L. and D. Brickley, *FOAF*. 2016 [cited 2016 Nov 3rd]; Available from: http://www.foaf-project.org/.

Dumbill, E., *Description of a Project*. 2016 [cited 2016 Nov 3rd]; Available from: http://lov.okfn.org/dataset/lov/vocabs/doap.

Cognitum, *Fluent Editor 2015*. 2017 [cited 2017 Feb 2]; Available from: http://www.cognitum.eu/semantics FluentEditor/.

R. Foundation, *The R project for statistical computing*. 2017 [cited 2017 Feb 2]; Available from: https://www.r-project.org/.