Enhanced Traceability in Model-based CASE Tools using Ontologies and Information Retrieval

Nitesh Narayan, Bernd Bruegge
Institut für Informatik
Technische Universität München
Boltzmannstrasse 3, 85748 Garching, Germany
{narayan, bruegge}@in.tum.de

Alexander Delater, Barbara Paech
Institute of Computer Science
University of Heidelberg
Im Neuenheimer Feld 326, 69120 Heidelberg, Germany
{delater, paech}@informatik.uni-heidelberg.de

***Abstract*—Model-based CASE tools provide mechanisms to capture and store heterogeneous artifacts produced during the software development process. These tools incorporate a meta-model describing artifact types and traceability links. Although model-based CASE tools provide required means to create and link different artifact types, still the process of linking artifacts is primarily manual resulting in missing or broken traceability links. This paper proposes a novel approach to create and utilize a project-specific ontology derived from the textual and structural information available in the development artifacts to assist the traceability link creation process. We discuss the benefits and challenges of incorporating the proposed approach in a model-based CASE tool.**

***Keywords*-artifacts, CASE tool, information retrieval, ontology, traceability link**

I. Introduction

Traceability between development artifacts plays an important role throughout the software development process. Traceability facilitates in program comprehension, maintenance, impact analysis, software reuse and prevention of misunderstandings [1] by utilizing the relationship information available in the software development artifacts.

Model-based CASE tools (Computer-Aided Software Engineering) aim at providing a centralized repository to capture and work with heterogeneous artifacts. As an example, UNICASE is a model-based CASE tool [2] merging two models: project model and system model. The possibility to incorporate model elements based on the context eases the creation and deployment of domain-specific models for special needs. The underlying meta-model defines the artifact types, e.g. requirements, use cases, UML diagrams, along with the traceability links artifacts can have between each other [3].

The links defined in the meta-model are called explicit traceability links. Explicit traceability links are directed references between two artifacts [4]. Each link has a specific type (e.g. refine, realize, verify etc.). For example, an artifact of type *requirement* can have a link of type ”realize” with an artifact of type *source code*. So the meta-model provides a predefined standard set of relationship types that can be supported by a model-based CASE tool [5]. However, the links which are frequently observed in the artifact instances in a project-specific scenario can not be foreseen, thus not all possible explicit links are included in the meta-model [6].

In addition to explicit traceability links, implicit traceability links can exist between artifacts. In contrast to explicit links, these links are not defined in the meta-model. The identification of implicit traceability links requires the analysis of existing explicit links to identify new traceability links not yet considered, but which are of importance, e.g. when performing change impact analysis to existing systems [7]. For example, artifact *A* is involved in realizing requirements *R1* and *R2* and explicit traceability links exist between them. Furthermore, code section *C* is based on artifact *A* and they are connected by an explicit traceability link, as well. Based on this knowledge, we can conclude that code section C is involved in realizing requirements *R1* and *R2* and that implicit traceability links exist between these artifacts. [7] Thus, the available information, for instance, can be used to verify if all requirements are implemented in the code. Similarly, the knowledge that two artifacts reference the same requirement could be used to derive a traceability link between these two artifacts.

Although model-based CASE tools provide required means to create and link different artifact types, still the process of linking artifacts is primarily manual resulting in missing or broken traceability links. Apart from explicitly defined traceability links, it is difficult to identify and incorporate new explicit link types not included in the meta-model. Additionally, identifying implicit links is only feasible if explicit links exist.

There are several approaches assisting the traceability link identification process, e.g. using ontologies [8] [9] and information retrieval techniques [10] [11]. These approaches either consider the structural information available in the model or the textual information available in the artifacts. Moreover, there is no clear separation between approaches considering explicit links and implicit links.

In this paper, we present a novel approach, which creates and utilizes a project-specific ontology over a model-based artifact repository to assist the traceability link creation process. The proposed approach considers the textual information available in the artifacts, along with the structural information in the meta-model. By considering the explicitly identified traceability links in the meta-model for a given artifact type, the approach assists in creating missing or broken links between the artifact instances. Therefore, the manual effort is reduced. Furthermore, the proposed approach eases the process of identifying new explicit link types frequently occurring within the artifacts.

The paper is structured as follows: In section 2, a motivation describes the problem in more detail and discusses related work regarding model-based traceability, ontologies and information retrieval. In section 3, we present our novel approach and describe it theoretically. In section 4, we apply our approach to a running example. The approach is discussed in section 5 and section 6 provides a conclusion.

II. Motivation

There are several approaches trying to deal with the problem of identifying traceability links. We highlight the benefits and limitations of the approaches while also presenting how they can be combined together to improve traceability link identification and creation.

A. Model-based Traceability

In model-based traceability, a meta-model describes the artifacts types, their attributes, as well as all possible explicit traceability links between the artifacts. A model-based traceability approach helps organizations to gain full benefit from the traces they develop and to allow project stakeholders to plan, generate, and execute trace strategies [3].

Model-based traceability supports creating pre-defined traceability links between artifacts. However, the process of traceability link creation is primarily manual. Moreover, only pre-defined link types are possible.

B. Ontologies and Traceability

Ontologies are used to define the terms that describe and represent a knowledge domain. They include the definition of concepts and relationships among them, allowing knowledge to be shared and reused [8]. Ontologies have been considered in several research work to support the traceability between artifacts, e.g. integrating ontologies into the Unified Process to provide concept-based traceability throughout the software lifecycle [8], or for recovering traceability links between existing source code and documentation to support reverse engineering [9]. Additionally, ontologies can substantially help in automating the link recommendation process using its built-in logical reasoning capabilities [9].

However, ontology-based approaches currently miss the possibility to utilize the available textual information within the artifacts. They only consider the structural information and rely on other mechanisms for ontology population. Usually, the focus is on a particular artifact type with predefined link types identified to be included in the ontology. Thus, the extensibility of the approach requires the base ontology to be modified for every newly introduced artifact and relationship type.

C. Information Retrieval

Information Retrieval (IR) techniques have been shown to assist with the automated generation of traceability links by reducing the time it takes to generate the traceability mapping [10]. Researchers have applied techniques such as Latent Semantic Indexing [11], vector space retrieval, and probabilistic IR. These approaches use natural language processing (NLP) to analyze the textual information available within the artifacts to identify possible traceability links.

IR techniques rely on textual analysis (lexical or semantic). Compared to ontologies, they do not consider the structural information of the artifacts. Artifact type, applicable traceability link type and existing knowledge of traceability links is not considered, as well.

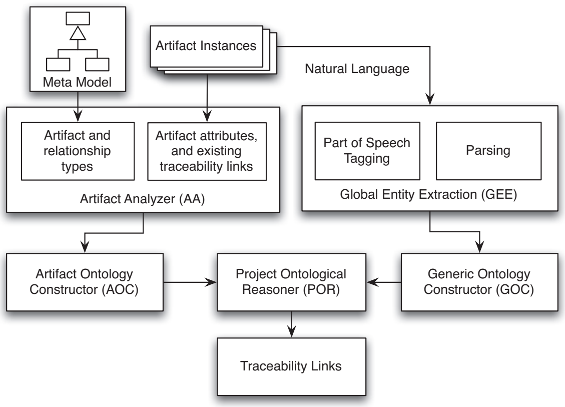
D. Combining Approaches

Until now, all these three approaches for traceability link creation are applied independently from one another. The combination of model-based traceability, ontology-based approaches and IR techniques would bring a more comprehensive traceability between artifacts by identifying additional traceability links. As model-based technologies have matured enough, it is possible to provide easy and effective tool support for a model-based CASE tool with an extensible meta-model for domain-specific needs. Ontology-based approaches help to identify missing, broken or even new traceability links between the artifact instances. At the same time, IR techniques can help during the ontology population by analyzing the textual information of the artifacts.

III. Approach

The process of the transition from artifacts to a project-specific ontology and the creation of traceability links is shown in Figure 1. The *Artifact Analyzer* (AA) module extracts the artifact types and explicitly defined traceability links from the *Meta-model*. This information provides insight into predefined sets of traceability links. Artifact types defined in the meta-model provide the possibility to focus on certain identified artifact types and the applicable properties.

Additionally, the AA module extracts the information from the *artifact instances*. Existing traceability links and various attributes, which if changed, can implicitly invoke changes in various artifacts. A simple example can be a priority attribute associated with a requirement, which if changed, causes the priority change of all tasks which are necessary to realize it. Existing traceability links between artifacts can assist in the link recommendation process. For example, suppose a task *T* needs an assignee and is linked to a requirement *R* (existing traceability link). *T* can be assigned to the developer who has worked on the similar task (similarity analysis using structural or textual analysis) within the same requirement as *T*. Model-based tools support this using structural information, but the process is manual and specific to a given scenario.



Meta Model
Artifact Instances
Natural Language
Artifact and relationship types
Artifact attributes, and existing traceability links
Artifact Analyzer (AA)
Part of Speech Tagging
Parsing
Global Entity Extraction (GEE)
Artifact Ontology Constructor (AOC)
Project Ontological Reasoner (POR)
Generic Ontology Constructor (GOC)
Traceability Links

Figure 1. Process flow

The *Global Entity Extraction* (GEE) module utilizes two toolkits from the Stanford NLP group [12]. The Stanford Part-Of-Speech Tagger is used to extract named entities from the natural language text in the artifacts. It considers every noun term as an important entity irrespective of it representing the name of an artifact completely or partially (composite name). Next, the Stanford Parser within the GEE module extracts the typed dependencies for the identified named entities. Stanford Dependencies provide simple and effective representation of grammatical relationships within a sentence. Thus, named entities along with the relationships they have semantically with other entities in the sentences are extracted.

The *Artifact Ontology Constructor* (AOC) and *Generic Ontology Constructor* (GOC) get the input from the AA and GEE module, respectively, and create an ontological representation of the information automatically. Class hierarchies identified for the ontology have two major elements: artifact (see Figure 2) and traceability link (see Figure 3). Analyzing the meta-model along with the instances derived from it gives a clear understanding of the artifact types and project-specific terminology to represent them.

The GOC module takes the input from the GEE module and populates the ontological class *NamedEntity*. Semantically named entities can have relationships as captured by the Stanford Dependencies. A simple sentence such as ”Eclipse crashes on Mac”, can be processed to identify subject as ”Eclipse” with traceability link ”crashesON” to a named entity ”Mac”.

The *Project Ontological Reasoner* (POR) combines the ontologies created by the AOC and GOC modules to derive traceability links. It derives the missing or broken explicitly

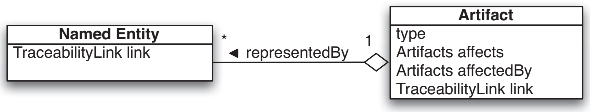


Figure 2. Excerpt showing Artifact Classes (UML notation)

identified link types in *Meta-model*, based on the artifact type extracted by AOC. This requires considering the ontology created by linguistic analysis of natural language text by GOC module. Identified named entities, which as a whole represent an artifact, are among the first candidates for a possible target of an explicit link type. While in scenarios where a named entity is not self-sufficient to represent an artifact completely, POR looks for semantic relationships between every known named entity in the source artifact. For example, if there is a requirement *R* which can be linked to a task *T* using the explicitly defined traceability link of type *realize* from the meta-model, POR takes all the named entities within the textual description of *R* and maps them to all the tasks *T* with named entities semantically related.

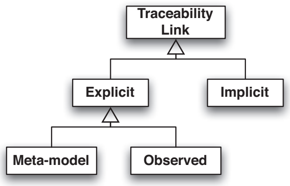


Figure 3. Excerpt showing Traceability Link Classes (UML notation)

Explicit traceability links *Observed* are the links which appear frequently within the artifacts and are not part of the meta-model. These traceability links are derived based on the semantic relationship between named entities in various artifacts within a software development project. Based on the frequency and importance, these links are prime candidates to become part of the meta-model. Traceability links between artifacts and additionally ”observed” explicit links supports reasoning to derive new knowledge of ”implicit” traceability links.

IV. Example

We apply our approach to an example project specified in the model-based CASE-tool UNICASE [2]. It describes a small supermarket and documents the requirements, classes, stakeholders as well as all tasks to realize them. The entire project contains 61 artifacts and 63 traceability links between them. To present the entire project structure and all artifacts within the project is beyond the purpose of this paper. Therefore, we only present a small subset of the artifacts, their structural and textual information as well as their existing and newly identified traceability links, to demonstrate the presented approach.

Suppose in our meta-model, we have the artifact types use case, initiating actor, class and bug report. In Figure 4, a graph of a subset of the artifacts is shown. The use case *Walk through automatic door* has an initiating actor *Shopper* and both are connected with an explicit traceability link. The class *AutomaticDoor* is linked to a bug report *DoorBug* with textual description *The door should only open if a human approaches*. The bug report was created because the door also opened on the detection of small animals (e.g. dogs). The bug report is assigned to a developer who shall fix the problem. The objective is now to identify new explicit and implicit links between the artifacts.

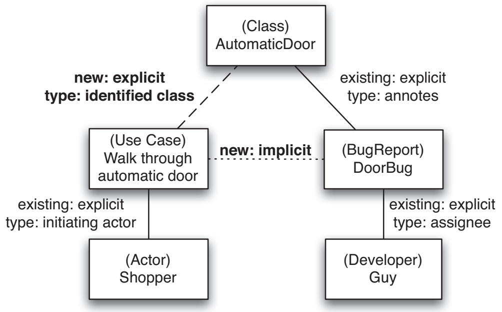


Figure 4. Identified traceability links using approach

The AA module extracts information from the meta-model about what artifact types and traceability links the artifact can have, along with any constraint on the artifact type as a target. It identifies that the class *AutomaticDoor* has a missing explicit link to the use cases it participates in. At the same time, there is an existing use case *Walk through automatic door* with initiating actor *Shopper*. To derive the target of type "use case" for explicit link type "participatesIN" from source class *AutomaticDoor*, our approach considers the relationship between the extracted named entities and the existing structural information.

The GEE module is used to parse the description of the use case (This use case describes how a user enters the supermarket through the automatic door) and identifies the named entity "automatic door". This named entity represents another artifact completely, which is of the required type to create the missing link as specified in the meta-model. Furthermore, parsing the textual description of the bug report *DoorBug* (see Figure 5) which is linked to class "automatic door" by traceability link type "annotate" reveals the named entity "door", which is also an existing entity in the project representing the super class of "automatic door".

Thus, based on the reasoned output of GEE and AA, we create a new explicit link between use case *Walk through automatic door* and class *AutomaticDoor* (see Figure 4). Additionally, parsing the description of bug report "DoorBug" identifies a semantic constraint on door functionality, i.e. open only $prep\_if$ a human approaches (see Figure 5).

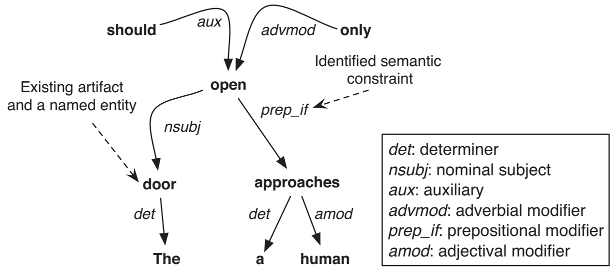

should
aux
advmod
only
open
Existing artifact and a named entity
Identified semantic constraint
nsubj
prep_if
door
approaches
det
det
amod
The
a
human
det: determiner
nsubj: nominal subject
aux: auxiliary
advmod: adverbial modifier
prep_if: prepositional modifier
amod: adjectival modifier

Figure 5. Linguistic processing of description of DoorBug

Now that explicit links between use case *Walk through automatic door*, class *AutomaticDoor* and bug report *DoorBug* exist, a new implicit link can be derived between them. "Door" is already identified as a named entity after parsing description of use case. The same entity is identified in the description of the bug report as these entities have syntactically the same term and are semantically related through the structural information of explicit links. Because both artifact types do not have an explicit specified relationship in the meta-model, our approach has identified a new implicit traceability link (see Figure 4). It is clear that if a use case is changed or adapted, a linked bug report can become obsolete. Moreover, the developer has now the advantage that the bug report is already linked to the related use case that needs to be revised. Our approach has improved the overall traceability between the considered artifacts by introducing new and useful traceability links.

V. Discussion

There exist different terms for explicit and implicit links. For example, Spanoudakis et al. [13] are using the terms direct/indirect links or dependent links, while Maeder et al. [4] use explicit/implicit links, as well. Implicit links are sometimes called *transient* of *multi-hop* links because they represent a connection between two elements in a transitive closure. Although in our work we additionally consider the relationships entailed by semantics and not only by literally present *hops* as implicit links.

Furthermore, a main difference is with respect to the syntactic matching and consideration of the semantic aspects of the artifacts. Spanoudakis et al. [14] use rules to syntactically match the terms to find traceability links, while the proposed approach uses IR by semantic similarity, instead. Moreover, they only consider standard a set of relationships between artifacts, while our proposed approach additionally leaves enough room to identify project-specific traceability links by logical reasoning.

In our approach, an ontology of named entities and the relationships between them extracted using linguistic processing allows us to identify missing, broken or new traceability links. We believe that having similar terms in two different artifacts does not sufficiently represent a link between them. Even if these artifacts have any relationship, it needs to be identified carefully with the proper semantics. Most of the approaches identify generic relationships, i.e. association. In our proposed approach, we consider a standard set of relationships, while leaving sufficient room for newly identified relationships.

In the approach, the artifacts and standard set of explicit relationships derived from the meta-model are separate from the logic used to derive the traceability links. This enables the applicability of the proposed approach to wide range of model-based artifact repositories.

From the implementation point of view, the extraction of the named entities as well as the ontology population from the meta-model and artifacts is straight-forward. However, extracting the named entities and recovering the semantic relationships may require to include linguistic processing techniques to identify semantic similarity and relatedness between concepts.

VI. Conclusion

In this paper, we presented a novel approach, which creates and utilizes a project-specific ontology over a model-based artifact repository to assist the traceability link creation process. Our proposed approach extracts the explicit information from a meta-model apart from extracting structural and textual information from artifact instances to populate the ontology. We create two sets of ontologies, i.e. one which considers meta-model, artifact instances and the available structural information, while the other one is a generic ontology created by linguistic processing of natural language text available in the artifacts. These two ontologies combined assist in creating missing and broken traceability links along with the possibility to identify additional traceability links.

In the future, we plan to implement and evaluate the presented approach for the existing model-based CASE tool UNICASE. In this work, we considered traceability link creation and identification separately from utilization. However, the proposed approach will create a large number of explicit and implicit links, especially in the context of large projects. Thus, the resulting volume of implicit links is likely to be more of a problem than a benefit. We are aware of this problem and plan to deal with this issue by considering the work context and expertise of the traceability user. Apart from standard heuristics used for evaluation of traceability links, we plan to do an extensive evaluation considering working context in which explicit and implicit links are useful, e.g. during software development and its corresponding project management.

References

[1] Egyed, A., and Grunbacher, P. ”Supporting software understanding with automated requirements traceability”. International Journal of Software Engineering and Knowledge Engineering, 2005, vol.15, no.5, pp. 783-810.

[2] Bruegge, B., Creighton, O., Helming, J., and Koegel, M. ”Unicase - an ecosystem for unified software engineering research tools”. Third IEEE International Conference on Global Software Engineering, ICGSE, 2007, vol. 2008.

[3] Cleland-Huang, J., Hayes, J. H., and Domel, J. M. ”Model-based traceability”. In TEFSE 09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering, Washington, DC, USA, 2009, IEEE Computer Society, pp. 6-10.

[4] Maeder, P., Philippow, I., and Riebisch, M.; , ”A Traceability Link Model for the Unified Process”. Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007, vol.3, no.1, pp. 700-705.

[5] Aizenbud-Reshef, N., Nolan, B. T., Rubin, J., and Shaham-Gafni, Y. ”Model traceability”. IBM Syst. J., vol.45, no.3, 2006, pp. 515-526.

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

[7] Aleksy, M., Hildenbrand, T., Obergfell, C., and Schwind, M. ”A Pragmatic Approach to Traceability in Model-Driven Development”, In Proceedings of the PRIMIUM Subconference at the Multikonferenz Wirtschaftsinformatik (MKWI) 2008, Garching, Germany, February, 2008, pp. 26-28.

[8] Noll, R. P., and Ribeiro, M.B., ”Ontological Traceability over the Unified Process”. Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS07), March 26-29, 2007, pp. 249-255.

[9] Y. Zhang et al., ”An Ontology-Based Approach for Traceability Recovery”. Proceedings of the 3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering (ATEM 2006), Genoa, October 1, 2006, pp. 36-43.

[10] Hayes, J. H., Dekhtyar, A., and Osborne, J. ”Improving requirements tracing via information retrieval”. Requirements Engineering, IEEE International Conference, 2003, pp. 138-147.

[11] Marcus, A., and Maletic, J. I. ”Recovering documentation-to-source-code traceability links using latent semantic indexing”. In ICSE 03: Proceedings of the 25th International Conference on Software Engineering, Washington, DC, USA, 2003, IEEE Computer Society, pp. 125-135.

[12] Stanford Log-linear Part-Of-Speech Tagger (version 3.0.1), Stanford Parser (1.6.6), Stanford Dependencies, Stanford University, Available at http://nlp.stanford.edu/software/index.shtml, April 2011.

[13] Spanoudakis, G., and Zisman, A. ”Software traceability: A roadmap”. In Handbook of Software Engineering and Knowledge Engineering, World Scientific Publishing, 2004, pp.395-428.

[14] Spanoudakis, G., Zisman, A., Perez-Minana, E., and Krause, P. ”Rule-based Generation of Requirements Traceability Relations”, Journal of Systems and Software, vol.72, no.2, 2004, pp. 105-127.