# Towards Querying of Traceability Information in the Context of Software Evolution

**Hannes Schwarz**
**Jürgen Ebert**
**Volker Riediger**

Institute for Software Technology
University of Koblenz-Landau
Koblenz, Germany
{hschwarz, ebert, riediger}@uni-koblenz.de

**Andreas Winter**

Institute for Computer Science
Johannes Gutenberg University, Mainz
Mainz, Germany
winter@uni-mainz.de

**Abstract:** Traceability of various artifacts created during the development of software systems plays an important role in software evolution. Subsequent changes to artifacts must be traced to other artifacts potentially affected by the change, thus ensuring the system's consistency or enabling to estimate the impact of changes. Using a querying approach, this paper shows how to extract traceability information on the basis of an integrated metamodel. The metamodel allows for the representation of artifacts such as requirements, design models, or code. It may also be customized in order to accommodate for specific needs.

## 1 Introduction

*Traceability* plays a significant role within the field of software development and evolution. Among others, the areas of application are reuse, change management, and maintenance. Closely coupled to change management and maintenance is *impact analysis*, dealing with the effect of changing one particular artifact to other requirements, code fragments, etc. [ACC+02]. For example, a changed requirement calls for finding out its implementing code fragments in order to allow for the realization of the change. Conversely, a modified code fragment must be traced back to the requirements it implements, to ensure proper implementation of relevant, but unchanged requirements. Next to this *horizontal traceability*, representing interdependencies of different types of artifacts, like requirements, test cases, and code, traceability in software evolution also addresses *vertical traceability*, showing interdependencies between artifacts of the same kind [RE93]. Vertical traceability originating from source code includes e. g. accessing external programs and databases, which are subject of program comprehension activities in multi language systems [KWDE98].

This paper shows how traceability information can be extracted using a graph-based querying approach. Graphs constitute an abstract representation of artifacts and their interconnecting traceability relationships. They conform to an integrated requirements metamodel based on the *Requirements Reference Metamodel* (*RRM*) [Weh06]. The RRM is mainly designed for requirements management, but also facilitates the representation of other artifact types such as components, design models, code fragments, or test cases. Thus it

alleviates the shortcomings of existing approaches, such as those of the popular requirements management tools *DOORS*[1] and *RequisitePro*[2]. The models inherent in these tools show deficiencies especially concerning the representation and traceability of artifacts related to activities subsequent to requirements engineering, e. g. design and implementation [Weh06]. According to the aim of reference models [Win00], the RRM is adaptable to specific areas of application.

The RRM is strongly influenced by experience gained in the EU-funded *ReDSeeDS* project [Śmi06] where the same graph-based querying approach is applied in the context of reuse: Based on a given set of requirements, potentially reusable artifacts connected to them are identified. ReDSeeDS employs a very detailed metamodel to represent the artifacts and their interrelationships produced in the course of a system's development [KSC+07]. In fact, the metamodel can be regarded as an adaption of the RRM, but is too "heavy-weight" to be of practical relevance for the purposes of the approach presented here.

The metamodel introduced in this paper, which is also an application of the RRM, introduces a refinement of the elements and links intended for code modeling. Consequently, it enables traceability between requirements and code fragments as well as between code fragments themselves. In order to more fully comprise the breadth of different artifact types occurring in a development project, it is also possible to integrate architectural and design models in the RRM. ReDSeeDS demonstrates how this can be achieved by incorporating the UML into its metamodel.

The issue of obtaining the graph representation of artifacts and their interconnections can be tackled by two means. Either artifacts are immediately recorded as graphs or the graph structure has to be *extracted* from the artifacts' native formats, e. g. XML files [BERS08]. In order to avoid inconsistencies, the second approach involves the drawback that the extraction step has to be repeated once changes are made to the artifacts. ReDSeeDS pursues both approaches: While the structure of architecture, design, and code is extracted using special tools, requirements are immediately recorded as graphs.

Following this introduction, section 2 presents an excerpt of an adapted version of the RRM, enabling query-based traceability over all kinds of software artifacts in section 3. Finally, section 4 sums up the paper and gives a short conclusion.

## 2 The Requirements Metamodel

The requirements metamodel is divided into five packages, each focusing on different aspects of the model, thus fostering its comprehensibility. The packages are elements, links, artifacts, representations, and containers.

The classes contained in the elements package, situated at the top of the model's generalization hierarchy, act as superclasses for all the other classes in the model. A Link connects any two LinkedElements, defined in the artifacts package, and permits their traceability (cf. Figure 1). Representations and containers serve to constitute different representations for the same artifact and to group model elements, respectively.

[1] http://www.telelogic.com/products/doors/index.cfm.
[2] http://www-306.ibm.com/software/awdtools/reqpro/.

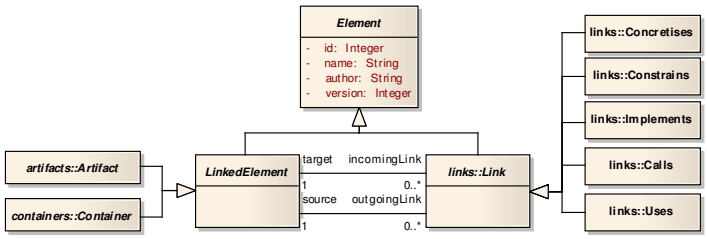


Figure 1: The contents of the elements and links packages of the requirements metamodel

The artifacts and links packages are of particular importance to the querying approach. Therefore they are discussed in more detail below.

### The Artifacts Package

Artifacts (cf. Figure 2) are the main entities of the RRM. Besides Components, DesignModels and CodeFragments, an Artifact may also constitute ChangeRequests, TestCases, or Requirements. ChangeRequests are proposals or demands to change one or more Elements. TestCases serve to verify requirements.

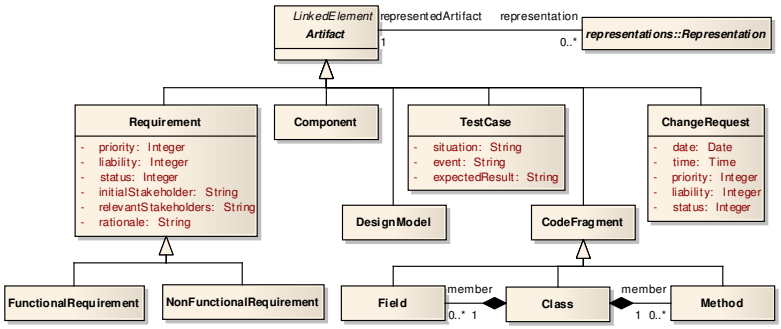


Figure 2: The contents of the artifacts package of the requirements model

Note that Artifacts do not possess attributes containing their, e.g. textual, representation. This task is performed by so-called Representations. In contrast to the RRM, this model adds subclasses of CodeFragment in order to permit fine-granular representation and traceability of such artifacts. Further refinements of the RRM, for example regarding architecture and design elements, are possible by subclassing Component, DesignModel, or even by adding other direct specializations of Artifact.

### The Links Package

The links package contains the Link class and its specializations. The concrete link classes depicted in Figure 1 serve as examples only and shall be defined as required. Consider Implements links: they are intended to connect CodeFragments to Requirements they are implementing. Uses and Calls links relate Classes and Methods to each other, respectively.

## 3 Querying

Provided that artifacts created during the development of software systems, and the traceability relationships between them are represented according to a precisely defined metamodel, querying can be employed to extract information needed for a specific purpose. The querying approach makes use of the *Graph Repository Query Language* (*GReQL*) [EKRW02]. GReQL is designed for posing queries to *TGraphs*, a very general kind of graphs. Besides being *directed*, *ordered*, and *attributed*, TGraphs feature *typed* vertices and edges [EF95]. The overall structure of a particular graph is defined by a *schema*, which is the equivalent to a metamodel in TGraph terminology. Consequently, it is possible to use the RRM as TGraph schema and to represent its instances as TGraphs.

GReQL itself supports a declarative formulation of queries in a SQL-like syntax. The queries rely on trivalent logics and regular path expressions for expressing constraints. The query's `from`-part binds variables to *domains*. The variables are used in the `with`-part to impose constraints. The `report`-part defines the structure of the query's result.

```
from class:V{Class}, req:V{Requirement}
with class.name = "Customer" and
     class --> {Implements} req
report req.name end
```

```
from caller:V{Method}, callee:V{Method}
with caller.name = "delete" and
     caller --> {Calls}* callee
report callee.name end
```

Figure 3: Sample queries for obtaining requirements potentially affected by a code modification (left query) and methods called by another method (right query).

Figure 3 displays two sample queries demonstrating horizontal and vertical traceability [RE93]. The left query could be applied if one wants to find out the possible impact of a code change to a exemplary "Customer" class: It retrieves all requirements directly related to this class. Changing the predicate in the `with`-part of the left query to test for a given requirement calculates the code fragments, affected by changing this requirement. The predicate depicting the relationship between requirement and code stays unchanged. The right query determines all methods directly *and* indirectly called by the "delete" method.

## 4 Conclusion

This paper gives an overview of how querying can be used to extract traceability information for supporting software evolution. The prerequisite is that artifacts and traceability relationships associated with a software system are abstractly represented according to a metamodel, integrating at least requirements and code views on software systems. A candidate for such a metamodel, based on the *Requirements Reference Model*, is presented. Though originally intended for requirements management, the RRM's customizability actually permits it to act as a basis for recording arbitrary artifacts and relationships.

Since the *ReDSeeDS* project, with its similar modeling and querying approach, involves extensive validation activities on the basis of real-world projects, it is expected to yield valuable input on the validity of the approach pursued in this paper.

Future work will be to further refine the querying technology, e. g. by employing path systems [Mar06] to retrieve every vertex reachable by a given path expression, thus relieving the user of the need to specify the types of artifacts to be returned. Consequently, rather simple queries would suffice to extract artifacts of many different types.

Concerning traceability, it is of interest to find out how relationships can be automatically identified and, if artifacts are changed, maintained to ensure consistency. It is expected that inference of traceability information requires more formal semantics than the notion associated with the name of the link type in natural language. Furthermore, the usefulness of forms of relationships beyond binary links, e.g. clusters and paths, has to be examined.

## References


[ACC+02] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering Traceability Links between Code and Documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.

[BERS08] D. Bildhauer, J. Ebert, V. Riediger, and H. Schwarz. Using the TGraph Approach for Model Fact Repositories. To appear in Proceedings of the 2nd International Workshop on Model Reuse Strategies (MoRSE'08), published by Fraunhofer IRB Verlag, 2008.

[EF95] J. Ebert and A. Franzke. A Declarative Approach to Graph Based Modeling. In *Graph-theoretic Concepts in Computer Science*, pages 38–50. Springer Verlag, 1995.

[EKRW02] J. Ebert, B. Kullbach, V. Riediger, and A. Winter. GUPRO. Generic Understanding of Programs - An Overview. *Electronic Notes in Theoretical Computer Science*, 72(2), 2002.

[KSC+07] A. Kalnins, A. Sostaks, E. Celms, E. Kalnina, A. Ambroziewicz, J. Bojarski, W. Nowakowski, T. Straszak, V. Riediger, H. Schwarz, D. Bildhauer, S. Kavaldjian, R. Popp, and J. Falb. Reuse-Oriented Modelling and Transformation Language Definition. Project Deliverable D3.2.1, ReDSeeDS Project, 2007. www.redseeds.eu.

[KWDE98] B. Kullbach, A. Winter, P. Dahm, and J. Ebert. Program Comprehension in Multi-Language Systems. In *Proceedings of the 5th Working Conference on Reverse Engineering 1998 (WCRE '98)*, pages 135–143. IEEE Computer Society, 1998.

[Mar06] K. Marchewka. Entwurf und Definition der Graphanfragesprache GReQL 2. Diplomarbeit, Universität Koblenz-Landau, 2006.

[RE93] B. Ramesh and M. Edwards. Issues in the Development of a Requirements Traceability Model. In *Proceedings of IEEE International Symposium on Requirements Engineering*, pages 256–259, 1993.

[Śmi06] M. Śmiałek. Towards a requirements driven software development system. Poster presentation at MoDELS, Genova, Italy, 2006.

[Weh06] T. Wehner. Entwicklung eines Referenzmodells zur Erfassung und Verwaltung von Anforderungen. Master's thesis, Universität Koblenz-Landau, 2006.

[Win00] A. Winter. *Referenz-Metaschema für visuelle Modellierungssprachen*. DUV Informatik. Deutscher Universitätsverlag, 2000.