This is a repository copy of *Building Model-Driven Engineering Traceability*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/109242/

Version: Published Version

Proceedings Paper:
Paige, Richard Freeman orcid.org/0000-0002-1978-9852, Olsen, Gøran K, Kolovos, Dimitris orcid.org/0000-0002-1724-6563 et al. (2 more authors) (2010) Building Model-Driven Engineering Traceability. In: ECMDA Traceability Workshop (ECMDA-TW). Sintef , p. 49.

Reuse
Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

Building Model-Driven Engineering Traceability Classifications

Richard F. Paige, Gøran K. Olsen, Dimitrios S. Kolovos, Steffen Zschaler and Christopher Power

Department of Computer Science, University of York, UK.
{paige, dkolovos, cpower}@cs.york.ac.uk
SINTEF, Oslo, Norway. Goran.K.Olsen@sintef.no
TU Dresden, D-01062 Dresden, Germany. szschaler@acm.org

Abstract. Model-Driven Engineering involves the application of many different model management operations, some automated, some manual. For developers to stay in control of their models and codebase, trace information must be maintained by all model management operations. This leads to a large number of trace links, which themselves need to be managed, queried, and evaluated. Classifications of traceability and trace links are an essential capability required for understanding and managing trace links. We present a process for building traceability classifications for a variety of widely used and accepted operations (both automated and manual) and show the results of applying the process to a rich traceability context.

1. Introduction

Traceability is the ability to chronologically interrelate uniquely identifiable entities in a way that matters. The IEEE Standard Glossary of Software Engineering Terminology [IEEE 2004] defines traceability 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; for example, the degree to which the requirements and design of a given software component match."* Thus, traceability refers to the capability for tracing artefacts along a set of chained operations, where these operations may be performed manually (e.g., crafting a software design for a set of software requirements) or with automated assistance (e.g., generating code from a set of abstract descriptions). In the context of Model Driven Engineering (MDE), many of the artefacts of interest are *models*, conforming to a metamodel, and are constructed using a set of modelling tools. Traceability in MDE is therefore predominantly concerned with chronologically defined relationships involving models and elements of models. The relationships between models are often called *trace links* [Olsen 2007]. However, when applying MDE, we start development of models from other kinds of artefacts: informal, natural language descriptions of requirements, spreadsheets, etc. Traceability needs to consider these artefacts as well, in terms of how models can be traced to other (non-model) artefacts and how (non-model) artefacts can be traced to models.

Generating and maintaining traceability information is important in order to help control the wealth of different artefacts in the development process: as systems become more complex, and as the application of MDE techniques within a process becomes more in depth, the need for better management of MDE artefacts increases. Traceability helps us to understand the many *dependencies* that exist between MDE artefacts. If we are able to support *end-to-end* traceability—that is, between all artefacts developed and generated in an MDE systems development process—then we can support a variety of different kinds of analysis; for example, showing that a requirement is fulfilled in implementation or showing that artefacts are up to date.

In a realistic MDE context, it is likely that a large amount of traceability information will be generated or created; understanding and managing this information will therefore be challenging, and will require structure to be imposed in order to understand the most appropriate ways to manage it. Traceability information can be better understood and managed through the help of a *traceability classification.* Several classifications have been published (e.g., for requirements engineering) [Ramesh 2001, Limon 2005, Walderhaug 2006]; these vary from abstract, conceptual classifications that help to systematise our understanding of the traceability problem domain, to concrete classifications (or traceability *metamodels*) that help to manage trace information in an implementation. Our focus in this paper is on the *process of building* traceability classifications, and on using this to classify both manual and automated MDE operations, thus helping to enable the full vision of end-to-end traceability.

The structure of the rest of the paper is as follows. In Section 2 we briefly review related work. In Section 3 we sketch a simple process for building traceability classifications. In Section 4 we outline how we have applied the process to build a traceability classification for the MODELPLEX[1] project, encompassing both manual and automated trace links.

2. Related Work

Different styles of traceability classifications have been presented in the literature. In particular, classifications given in terms of *scenarios of use* of traceability are postulated by [Olsen 2007, Walderhaug 2006]. Classifications in terms of specific *domains* have been produced by [Ramesh 2001] for requirements engineering, and for business applications [Rummler 2007].

Traceability classifications in MDE have been developed that emphasise different *attributes* or *characteristics* of traceability. In particular, two categories of classifications can be identified in the literature: classifications that focus on *explicit trace links* (which are captured directly in models themselves using a suitable concrete syntax), and *implicit trace links* where trace information is generated or arises due to application of one or more model management operations. The classification we build in Section 4 includes both explicit and implicit trace links.

More generally, traceability has been identified as an important research issue. The European project AMPLE, focusing on Aspect-Oriented and Model-Driven product

1 http://www.modelplex.org/

line engineering aims to support traceability across software product lines [Rummler 2007]. The Grand Challenges in Traceability report [GCT 2006] identifies a number of challenges for managing and maintaining trace information, including evolution of trace information, trace link semantics, and eliciting trace knowledge.

Many trace tools have been developed for managing trace information. Some of the most well-known and widely used include Reqtify [ChiasTek 2007], RequisitePro [IBM 2007], and Acceleo Pro [Obeo 2007]. An approach to trace link generation has been presented by Egyed [Egyed 2003], who presents a trace tool that automatically derives traces from code through requirements. [Kolovos 2006] presents support for trace links where trace information is stored separate from the model; [Jouault 2005] outlines a loosely coupled trace scheme for model transformations.

3. A Traceability Classification Process

As suggested by the related work presented in Section 2, a number of traceability classifications have been presented, but there is little guidance yet on how to systematically build and maintain them (ranging from conceptual models to concrete metamodels). A demonstrated *process* for building traceability classifications is useful for this, not only for building classifications in the first place, but for maintaining classifications as new MDE operations, new relationships between MDE artefacts, and new stakeholder requirements, arise. In order to support this, we first describe a very simple process for building and maintaining traceability classifications, and then in the next section we use it to develop a classification for the stakeholder requirements of MODELPLEX.

The simple process is called the *Traceability Elicitation and Analysis Process (TEAP)*. It is derived from a process developed in [Chan 2005] for elicitation and understanding different forms of model-based contracts. The aim of TEAP is to elicit and analyse traceability relationships in order to determine how they fit into a traceability classification. While eliciting new traceability relationships, we improve our understanding of the *key attributes* of these traceability relationships: the artefacts they involve, their semantics, and their domain of applicability.

When applying TEAP, we typically bootstrap from a simple traceability classification or metamodel, and iteratively and incrementally refine the classification through a number of TEAP *cycles.* Each cycle in the TEAP enriches the existing classification in terms of one or more key attributes of interest.

TEAP is a *triggered* process, in which there are three main activities in each iteration: Elicitation, Analysis, and Classification. In elicitation, we identify basic trace links and relationships. In the second phase, we extrapolate from these the key characteristics of traceability (based on our current understanding) and as a result of this analysis, identify constraints on relationships and any generalisations of relationships and trace links. In the third phase, we build a classification. From this, we can iteratively enrich, refactor, and improve the classification, for example when customer requirements dictate.

TEAP is intended for use in building new classifications *and* for maintaining classifications. The classification we describe for MODELPLEX is generic, and may be

useful in a variety of settings – however, it can and will be extended over the duration of MODELPLEX, and TEAP can be used to support this.

TEAP is derived from the spiral software development model, due to Boehm, and the spiral model in requirements engineering. The main difference is that TEAP produces metamodels and classifications, rather than software or requirements. The advantage of defining and using TEAP is that it gives extensibility to its product. This is essential in providing a generic, flexible framework for classifying and managing traceability. In other words, the traceability classification can be kept up-to-date by carrying out additional TEAP cycles.

As mentioned earlier, TEAP is a *triggered* process; we can identify when TEAP cycles should be executed. The triggers for executing TEAP cycles include:

- a new model management operation has been defined, in which case cycles should be executed in order to refine the classification
- the system development process has changed; thus cycles should be executed in order to refine how to handle sequences of model management operations.
- one or more modelling languages have changed to include new model relationships, artefacts, or changed model relationships, in which case cycles should be executed to refine and extend the explicit link classification.
- The existing classification does not capture all requirements for traceability inherent in a domain or project context.

The TEAP process is meant to provide guidance, not to dictate the way in which the traceability classification must be extended and refined.

4. Example: Building the MODELPLEX Classification

In this section we outline how we used TEAP (from Section 3) to build a conceptual traceability classification for use in the MODELPLEX European project. We start with a very short overview of the traceability requirements for MODELPLEX, then summarise a few iterations of TEAP applied to these requirements.

4.1 MODELPLEX traceability requirements

MODELPLEX is a three-year integrated project funded by the European Commission, with a mandate to improve productivity in the development of complex systems through use of MDE. The project is case-study driven, with four industrial partners - SAP, Telefonica, Western Geco, and Thales Information Systems - providing real complex system scenarios, to which MDE technology (e.g., architectural modelling, model transformation, performance analysis, simulation, model composition) is to be applied. Each case study has traceability requirements. These can be summarised as:

- the ability to record traceability information that results from applying model management operations: model-to-model (M2M) transformations, model-to-text (M2T) transformations, compositions, simulations, and refinements;
- the ability to manually create trace links between MDE artefacts, e.g., between an architectural model and a use case model, between a weaving model and a design

model. Manual creation of trace links can involve modelling tools, or the use of a textual domain-specific language tailored for one or more of the case studies;

- the ability to (typically manually) create trace links between MDE artefacts (e.g., models) and non-MDE artefacts (e.g., requirements stored in a MANTIS repository, PDF documents). This is a necessary requirement as some of the MODELPLEX partners do not currently use MDE technologies in their everyday practice; moreover, non-MDE artefacts will always play a substantial role in the MDE process, e.g., for early requirements elicitation and description.
- the ability to store and retrieve trace links and trace metadata from a repository.

We decided to initially produce a conceptual traceability classification that addressed the basic information that needed to be recorded for the first three sets of requirements above. This would then be refined and implemented in a traceability tool, which also provided repository features.

4.2 Basis for TEAP iterations

To apply TEAP, we initially constructed simple traceability infrastructure that would evolve over the TEAP iterations. This infrastructure is depicted in Fig. 1. It is, effectively, a very simple traceability metamodel that expresses the fundamental concepts of *artefacts, trace links,* and *operations.* We specialise this model in the following subsections. A*rtefacts* may be both MDE artefacts (e.g., domain-specific models) and non-MDE artefacts (e.g., spreadsheets), and operations (either manual or automated) elaborate the traceability information to be recorded. Finally, the different kinds of trace links will be discussed in the following sections.

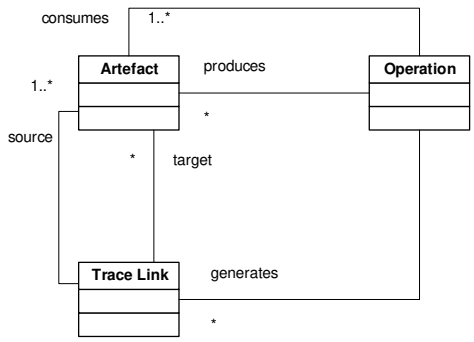

Fig. 1: Basic traceability classification infrastructure

The classification in Fig. 1 is generic, and could thus be used to produce a variety of specialised traceability classifications for different domains and contexts. In each case, TEAP can be used to maintain and extend the basic infrastructure for new domain-specific requirements and contexts.

4.3 Adding explicit and implicit traceability relationships

We start to extend our simple traceability infrastructure by carrying out TEAP cycles. Our initial cycle was triggered by the obvious observation that the trace link model in Fig. 1 did not satisfy all MODELPLEX requirements for traceability. We thus carried out elicitation (what general kinds of trace links exist?), analysis (what information did these trace links require?), and built a simple classification. The cycle focused on the notions of implicit and explicit traceability. Implicit traceability involves trace links that are created and manipulated by application of MDE operations. Explicit traceability is defined in terms of trace links that are concretely represented in models. Therefore, our initial TEAP cycle was very simple and refines the traceability infrastructure to that shown in Fig. 2.

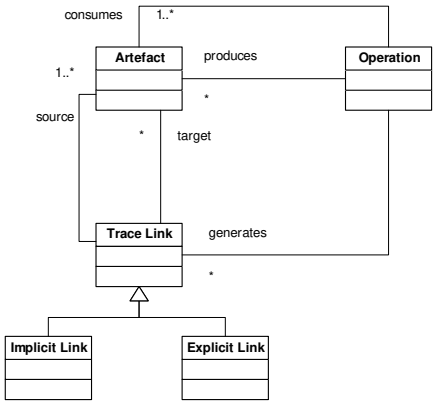

Fig. 2: Explicit and implicit trace link classification

4.4 Iterations for implicit trace link classification

The next cycle was triggered by two observations: the classification of implicit trace links was weak; and, by obtaining more precise requirements about the operations that were to be supported in MODELPLEX. We thus carried out a TEAP cycle for improving our classification of implicit trace links. As usual, there is elicitation (what kinds of MDE operations are relevant?), analysis (what information do operations require, and how should this information be constrained?), and classification.

MDE operations implicitly define a variety of different trace links between two or more artefacts (note that many artefacts will be models, e.g., for model-to-model transformation, but non-model artefacts such as code and requirements may be involved too). An MDE operation takes a set of artefacts as input (if the artefacts are models, they may be from one or more different modelling languages) and produces a set of artefacts and a set of trace links as output. Trace links can be recorded either in the source or target artefacts, or as a separate model [Kolovos 2006]. The basic MDE operations are elicited from studying the relevant standards—particularly UML, MOF, and QVT—which indicate how trace links can be generated. The operations we initially identify are: *query*, *transformation*, *composition* (sometimes called *merging*),

update (also called *update-in-place*), *creation*, *deletion*, *model-to-text*, and *sequences* of operations. The resulting classification (focusing strictly on subclasses of Implicit Link from Fig. 2) is shown in Fig. 3. As well, new operations (subclasses of Operation) are added for each, e.g., Query Operation, Delete Operation, etc.

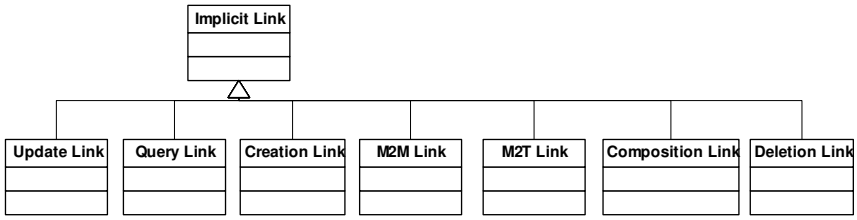

Fig. 3: Additions for implicit trace links

Examples of the well-formedness constraints elicited and produced in the analysis phase are shown below.

```
context QueryOperation inv:
   self.consumes->forAll(a |
      self.generates->exists(t | t.source->includes(a) and
      self.produces->includesAll( t.target )));
   self.generates->forAll( t | t.oclIsTypeOf(QueryLink))

context CreationOperation inv:
   self.consumes->isEmpty();
   self.generates.target->includesAll(self.produces);
   self.generates->forAll( t | t.oclIsTypeOf(CreationLink));
```

4.4 Iterations for explicit traceability relationships

We next carried out a TEAP cycle for improving our classification of explicit trace links. This cycle was triggered by refined MODELPLEX requirements for explicit representation of trace information in models and in domain-specific languages. Recall that explicit trace links are explicitly defined between artefacts, using one or more languages. For example, a UML dependency constitutes a specific kind of explicit trace link. Obviously, there are many different kinds of explicit trace links, and many of them will be domain specific (and language specific). We illustrate the results of the TEAP process for MODELPLEX's explicit trace links. As was the case for implicit trace links, we carry out elicitation (what kinds of explicit trace links are relevant?), analysis (what information do these links require?), and classification.

The initial elicitation and analysis identified two basic kinds of explicit trace links: *model-model* links (e.g., the aforementioned UML dependency), and *model-artefact* links (e.g., between a model and a spreadsheet). Trace links entirely between non-model artefacts were determined to be out of scope, and managed by other tools.

The model-model links were then further analysed. These were determined to be divisible into *static* links (which represent structural relationships that do not change over time) and *dynamic* links, which represent information regarding models that may

change over time. A variety of both static and dynamic links were collected from MODELPLEX's requirements. Some examples of static model-model links are:

- *consistent-with* links, where two models must remain consistent with each other, e.g., a sequence and class diagram.
- *dependency* links, where the structure and meaning of one model depends on a second. Dependency links can be further subdivided into: *is-a* links (e.g., sub-typing), *has-a* links (e.g., references), *part-of* links, *import* and *export* links, *usage* links (e.g., one component uses another's services), *refinement* links (e.g., where a component reduces non-determinism in a second component).

Some examples of dynamic model-model links include: *calls* links (where one model calls behaviours provided by a second model), *notifies* links (where it is necessary to record information that cannot be handled automatically, such as changes that require human intervention). Furthermore, there are design-time relationships, such as *generates* or *builds* links that indicate where information from one model is used to produce or deduce the second model; and *synchronized-with* relationships, where behaviours between models are synchronized. These usually apply when there is some kind of tracking mechanism introduced between models. A further example includes the *consistent-with* trace links that can exist between an early requirements specification such as those in i*, and models of functional requirements [Alencar, 2000].

Model-artefact links are important in MODELPLEX, to support trace links between MDE artefacts (including UML models as well as domain-specific models) and non-MDE artefacts, particularly spreadsheets, requirements databases, and results of simulations. The scope of model-artefact links is broad, and we did not attempt to elicit all such links in our classification. We provide important links in MODELPLEX, while giving a classification that can be extended in the project.

The intent of most model-artefact links is to enable coverage checking, e.g., of requirements. This is the case in MODELPLEX. The trace links of interest in MODELPLEX were the following:

- *satisfies* links, to indicate that properties or requirements captured in an artefact are satisfied by a model. Variants on satisfies links include *verifies* links (which involve a specific mechanism, such as testing) and *certifies* links (which also link to external standards and arguments for safety or security).
- *allocated-to* links, used when information in a non-model artefact is allocated to a specific model that represents the information.
- *performs* links, indicating that a task described in an artefact is carried out by a specified model
- *explains* and *supports* links, indicating that, e.g., a model is explained by a non-model artefact (e.g., natural language documentation).

These trace links are summarised in Fig. 4 (focusing strictly on the explicit trace link part of the classification).

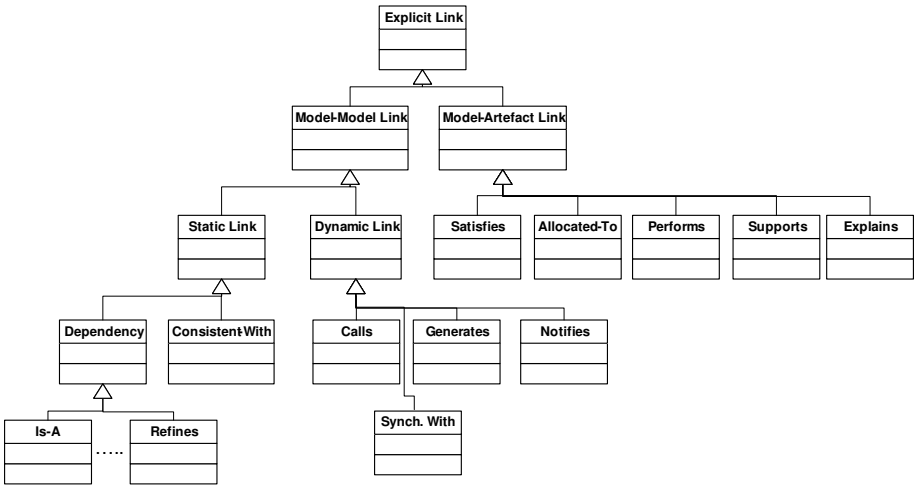

Fig. 4: Summary of explicit trace links

As this brief discussion suggests, the space of explicit trace links is rich and complicated, and encompasses many domain- and language-specific characteristics.

5. Conclusions

We have presented a lightweight process for building traceability classifications, and illustrated its application to a conceptual classification for the MODELPLEX process. The classification identifies basic categories of traceability – implicit and explicit – and populates these categories with trace links from different MDE operations (such as transformation and query) and from different modelling scenarios relevant to MODELPLEX. The classification developed above is a living document, and will be extended iteratively and incrementally over the course of the project. Furthermore, it will be *refined* from a conceptual classification to a concrete design that can be supported in a trace tool that includes a repository and capabilities for retrieving, storing, and updating trace links.

TEAP has so far proved to be suitably lightweight, yet helpful in guiding the construction and the iterative improvement of traceability classifications. Since most, if not all, classifications must evolve with time, the value of an iterative and incremental process for evolving classifications is substantial.

Acknowledgments

This research has been co-funded by the European Commission within the 6th Framework Programme project MODELPLEX contract number 034081.

References

[Alencar, 2000] F. Alencar, J. Castro, G. Cysneiros, J. Mylopoulos. From Early Requirements Modeled by the i* Technique to Later Requirements Modeled in Precise UML. In *Proc. of Workshop de Engenharia de Requisitos 1ed. Brasil,* Brazil, 2000.

[Aizenbud-Reshef 2005] N. Aizenbud-Reshef, R.F. Paige, J. Rubin, Y. Shaham-Gafni, and D.S. Kolovos. Operational semantics for traceability. In *Proc Workshop on Traceability 2005*, Nuremberg, Germany, November 2005.

[Aizenbud-Reshef 2006] N. Aizenbud-Reshef, B. Nolan, J. Rubin, and Y. Shaham-Gafni. Model traceability. *IBM Systems Journal* 45(3), 2006: pp 515-526.

[Borland 2007] CaliberRM: http://www.borland.com.

[Chan 2005] Z.E. Chan and R.F. Paige. Designing a Domain-Specific Contract Language: a Metamodelling Approach, in *Proc. EC-MDA 2005*, LNCS 3748, Springer-Verlag, November 2005.

[Chiastek 2007] Chiastek, Reqtify: http://www.chiastek.com/products/reqtify.html

[Egyed 2003] A. Egyed, A Scenario-Driven Approach to Trace Dependency Analysis. *IEEE Transactions on Software Engineering*, 2003. 29: p. 17.

[GCT 2006] Grand Challenges in Traceability, Center of Excellence for Traceability http://www.traceabilitycenter.org/downloads/documents/GrandChallenges/

[IEEE 2004] IEEE Std 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology, 2004.

[Jouault 2005] F. Jouault. Loosely Coupled Traceability for ATL. In *Proc. ECMDA-FA Workshop on Traceability*, SINTEF Technical Report STF90, November 2005.

[Kolovos 2006c] D.S. Kolovos, R.F. Paige, and F.A.C. Polack On-Demand Merging of Traceability Links with Models. *ECMDA 06 Traceability Workshop* Bilbao, 2006.

[Limon 2005] A. Limon and J. Garbajosa. The Need for a Unifying Traceability Scheme. In *Proc. Workshop on Traceability 2005*, Nuremberg, November 2005.

[Obeo 2007] Acceleo Pro Traceability, http://www.acceleo.org

[Olsen 2007] G. Olsen and J. Oldevik. Scenarios of traceability in model to text transformations. In *Proc. ECMDA-FA 2007*, LNCS, Springer-Verlag, June 2007.

[Walderhaug 2006] S. Walderhaug, U. Johansen, E. Stav, and J. Aagedal. Towards a generic solution for traceability in MDD. In *Proc. Workshop on Traceability 2006*, Bilbao, Spain, July 2006.