# A Model Management Framework for Maintaining Traceability Links

Thomas Beyhl, Regina Hebig and Holger Giese
Hasso Plattner Institute for IT Systems Engineering at the University of Potsdam
Prof.-Dr.-Helmert-Street 2-3, 14482 Potsdam, Germany
{prename}.{surname}@hpi.uni-potsdam.de

**Abstract:** In MDE diverse modeling and model transformation languages are applied to describe and derive the envisioned system. Traceability is a prerequisite for maintaining consistency between different development artifacts. Thereby, the usefulness of traceability links increases with their completeness and correctness. In practice, automatic creation and maintenance of traceability links is required to be useful. This is addressed by heuristic approaches that derive traceability information statically or by model transformation technologies that provide traceability links as additional execution result. However, the maintenance of traceability links for a set of diverse languages and transformation technologies as combined in MDE is still a challenging task. In this paper, we present a framework that provides and treats all traceability information using the common format of hierarchical megamodels. Thereby, different approaches for gaining traceability information can be combined. Information provided by transformation technologies is translated into this common format.

## 1 Introduction

In MDE diverse modeling and model transformation languages are applied to describe and derive an envisioned system. Traceability is a prerequisite for maintaining consistency between different development artifacts. Thereby, the usefulness of traceability links increases with their completeness and correctness. In practice, a huge amount of development artifacts is used to describe the envisioned system. Therefore, the manual creation and maintenance of traceability links is not feasible and automatic creation and maintenance of traceability links is required. This is addressed by heuristic approaches that derive traceability information statically, e.g., [An02], or by model transformation technologies, which provide traceability links as additional execution result, e.g. [Jo05]. Winkler et. al [WP10] give a survey of traceability in requirements engineering and MDE. However, the maintenance of traceability links for a set of diverse languages and transformation technologies as combined in MDE is still a challenging task. First, many transformations are performed manually or by transformation technologies, which provide no traceability information. Second, the proprietary format of traceability links provided by different transformation technologies prevents further treatment of this information for analysis of the whole set of artifacts, e.g. for impact analysis or consistency checks. In the following, we present a framework for maintaining traceability links that addresses the diversity of modeling and model transformation languages in MDE. We developed the framework as a basis of a set of research projects that build on traceability, e.g. a model transformation composition framework [Se11], a framework extension for version control capabilities[1] and a corresponding model management build server [SHG12]. Thereby, we address the two challenges by allowing a combination of different traceability approaches for gaining traceability information and by providing and treating all traceability information using the common format of hierarchical megamodels.

## 2 Model Management Framework

Barbero et. al [BB08] introduce the concept of megamodels. Such megamodels capture models and relationships between models. Thus, megamodels are well suited for capturing, providing and maintaining diverse models and traceability links between them. Hierarchical megamodels combine high-level traceability models (megamodels) and low-level traceability models (models containing fine-grained traceability links) [SNG10]. Thereby, hierarchical dependencies between high-level and low-level modeling artifacts and between traceability links are defined. Thus, traceability information is combined into one common traceability model. In our case, a hierarchical megamodel provides a logical view of the local workspace by capturing representatives for artifacts (e.g. models, model elements, source code). The framework[2] presented here a) supports an easily extensible set of modeling and model transformation languages, b) monitors artifact changes and updates the set of representatives within the hierarchical megamodel concerning the development artifacts in the local workspace, and c) gains and maintains traceability links automatically, based on monitored development artifact changes, model transformation executions and available traceability approaches.

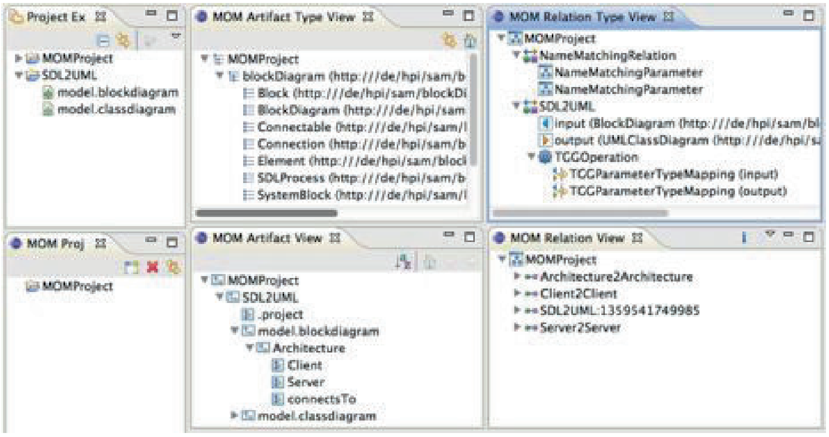
Project Ex
MOMProject
SDL2UML
model.blockdiagram
model.classdiagram
MOM Artifact Type View
MOMProject
blockDiagram (http:///de/hpi/sam/b
Block (http:///de/hpi/sam/blockDi
BlockDiagram (http:///de/hpi/sam
Connectable (http:///de/hpi/sam/l
Connection (http:///de/hpi/sam/b
Element (http:///de/hpi/sam/block
SDLProcess (http:///de/hpi/sam/b
SystemBlock (http:///de/hpi/sam/l
MOM Relation Type View
MOMProject
NameMatchingRelation
NameMatchingParameter
NameMatchingParameter
SDL2UML
input (BlockDiagram (http:///de/hpi/sam/bl
output (UMLClassDiagram (http:///de/hpi/s
TGGOperation
TGGParameterTypeMapping (input)
TGGParameterTypeMapping (output)
MOM Proj
MOMProject
MOM Artifact View
MOMProject
SDL2UML
.project
model.blockdiagram
Architecture
Client
Server
connectsTo
model.classdiagram
MOM Relation View
MOMProject
Architecture2Architecture
Client2Client
SDL2UML:1359541749985
Server2Server

Figure 1: Views of the hierarchical megamodel

[1]http://www.hpi.uni-potsdam.de/giese/gforge/mdelab/?page_id=108
[2]http://www.hpi.uni-potsdam.de/giese/gforge/mdelab/?page_id=63

## 2.1 Type System

Information about physical artifacts, types, model operations and traceability links are captured in a hierarchical megamodel. Development artifacts (e.g. models, model elements, code) in the local workspace are represented as *artifacts* within the hierarchical megamodel. Correspondingly, all captured and retrieved traceability information is represented as *relations*. The framework includes an additional type layer, where metamodels for used *artifact types* can be registered to improve the quality of the information in the hierarchical megamodel. This type information is used to assign an artifact type to each captured artifact. Thus, the framework can support arbitrary modeling languages. In addition, *relation types* can be specified to describe the signature of model operations and traceability links (i.e. information about types of input and output artifacts of a relation). To support an extensible set of model transformation languages a plug-in mechanism is provided. Thereby, adapters can be plugged-in, which hide the model transformation technology internals. For example, we provide adapters to execute ATL[3] and Xpand[4] model transformations. Adapters are used to trigger the execution of model transformations and to retrieve traceability information about the execution. When executing a model transformation an *executable relation* is created within the hierarchical megamodel and the corresponding relation type is assigned. Traceability links are represented by *non-executable relations* in the hierarchical megamodel, which connect input and output artifacts. Figure 1 depicts different views (artifact types, relation types, artifacts, relations) of the hierarchical megamodel. For example, the application of the model transformation $SDL2UML$ and automatically created traceability links (e.g. $Client2Client$) are depicted.

## 2.2 Updating Artifact Representations

Development artifacts can be registered for being monitored by the framework. Changes on these registered development artifacts are monitored and a notification mechanism provides change events when these monitored development artifacts change. Further, the model transformation adapters may throw change events when model transformations lead to a change, deletion or creation of models or model elements. Events are propagated to the workspace builder, which is responsible to update the internal hierarchical megamodel (i.e., create, update, or delete artifacts). Further, events are propagated to tools that base on the framework.

## 2.3 Maintaining Traceability Information

Special kinds of the mentioned tools are traceability adapters, which enrich the hierarchical megamodel with statically derived traceability information implementing heuristic

[3]http://www.eclipse.org/atl/
[4]http://www.eclipse.org/modeling/m2t/?project=xpand

techniques. Furthermore, model transformations are directly triggered via our framework using model transformation adapters. Thereby, the execution of model transformations can be monitored in more detail. For example, also proprietary forms of traceability links that are created during the execution of model transformations (at runtime), e.g. [Jo05], can be translated to be stored in our hierarchical megamodel. We implemented different technology-specific model transformation adapters. Thereby, we experienced how strong the quality of retrievable traceability information and also the control over execution configuration parameters depends on the model transformation technology. For example, it turned out that the very flexible model transformation technology Jet[5] is hard to control and provides only little traceability information. For example, the question which target artifacts are generated cannot be answered before executing the model transformation. This circumstance makes the integration of Jet into model transformation chains unfeasible. In contrast, the less flexible and more restricted model transformation technology Xpand, allows better control and traceability information can easily be retrieved during the execution of the model transformation. For all gained traceability information traceability links (non-executable relations) are created within the hierarchical megamodel. Traceability links may become incorrect when a model transformation was executed or artifacts were changed manually. Based on the automated update of the hierarchical megamodel the framework recognizes such situations and identifies the need to update or delete specific traceability links as well. Traceability links retrieved by monitoring model transformation executions are deleted (or marked as invalid), since they can only be restored when executing the model transformation again. However, traceability adapters can directly update all traceability links, which they had created.

## 3 Related Work

The AM3Core [BB08] provides a metamodel for megamodels that includes the concept of models, relationships between models and chains of relationships. The Model Management Tool Framework (MMTF) [Sa07] is an environment that enables different software developers to work on different related parts of models and their relationships with the help of a Model Interconnection Diagram (MID). However, both approaches are suitable for automatically maintaining and combining traceability links.

## 4 Conclusion

We presented a framework for capturing and maintaining artifacts and traceability links between them within a hierarchical megamodel. Thereby, we showed that different traceability approaches can be combined to establish a chain of traceability links within model transformation chains. In addition, we showed using the examples of ATL, Xpand, and Jet that the framework can be extended to support different technologies and that traceability

[5]http://www.eclipse.org/modeling/m2t/?project=jet#jet

information can be extracted automatically. The traceability information stored within the hierarchical megamodel and the events provided by the framework enable other tools to use these traceability information. The framework was successfully applied in different research projects. In future work, we focus on extending the set of model transformation adapters, e.g. QVT Operational[6].

## Acknowledgement

The authors are grateful for the input of Andreas Seibel and the student assistants Henrik Steudel, Dmitry Zakharov, Arian Treffer, Johannes Dyk and Manuel Hegner.

## References

[An02] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.

[BB08] Mikaël Barbero and Jean Bézivin. Model driven management of complex systems: Implementing the macroscope's vision. *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 277–286, 2008.

[Jo05] Frederic Jouault. Loosely coupled traceability for ATL. In *Proceedings of European Conference on Model Driven Architecture workshop on traceability*, pages 29–37, 2005.

[Sa07] Rick Salay, Marsha Chechik, Steve Easterbrook, Zinovy Diskin, Pete McCormick, Shiva Nejati, Mehrdad Sabetzadeh, and Petcharat Viriyakattiyaporn. An Eclipse-based tool framework for software model management. *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*, pages 55–59, 2007.

[SHG12] Henrik Steudel, Regina Hebig, and Holger Giese. A Build Server for Model-Driven Engineering. In *6th International Workshop on Multi-Paradigm Modeling (MPM 2012)*. ACM, 2012.

[Se11] Andreas Seibel, Regina Hebig, Stefan Neumann, and Holger Giese. A Dedicated Language for Context Composition and Execution of True Black-Box Model Transformations. In *4th International Conference on Software Language Engineering (SLE 2011)*, pages 19–39, Braga, 2011.

[SNG10] Andreas Seibel, Stefan Neumann, and Holger Giese. Dynamic hierarchical mega models: comprehensive traceability and its efficient maintenance. *Software & Systems Modeling*, 9(4):493–528, September 2010.

[WP10] Stefan Winkler and Jens von Pilgrim. A survey of traceability in requirements engineering and model-driven development. *Software & Systems Modeling*, 9(4):529–565, September 2010.

[6]http://www.eclipse.org/projects/project.php?id=modeling.mmt.qvt-oml