# From Software Traceability to Global Model Management and Back Again

Andreas Seibel
*Hasso Plattner Institute at the University of Potsdam*
*Prof.-Dr.-Helmert-Str. 2-3,*
*14482 Potsdam, Germany*
*andreas.seibel@hpi.uni-potsdam.de*

***Abstract*—In the past, traceability was concerned with tracing requirements also known as requirements traceability. Meanwhile, model-driven engineering (MDE) becomes popular and in this context, requirements traceability evolved to a more general software traceability, which is about tracing any relationship between software artifacts. Due to increasing heterogeneity and complexity of MDE, global model management (GMM) becomes indispensable to MDE. In state-of-the-art GMM approaches, traceability is primarily concerned with tracing model transformations, which generate traceability information as side-effect. However, this is only a subset of the capabilities of software traceability.**

**My hypothesis is that GMM can benefit from software traceability by applying software traceability techniques into the context of GMM. Thus, I will consider GMM from a traceability perspective and show new possibilities to GMM that arise from taking in this perspective.**

***Keywords*-Software Traceability, Global Model Management, Model-Driven Engineering;**

## I. Introduction

Originally, traceability in the domain of software development was the ability to trace requirements down to code, which is also known as *requirements traceability*. Gotel et al. give the following definition in [1]:

> *"...the ability to describe and follow the life of a requirement, in both a forward and backward direction; i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases."*

In the mean time, model-driven engineering (MDE) became more and more popular in software development. It emphasizes on models as first-class entities and the application of automated model operations, e.g., model transformations or synchronizations to automatically process models. Therefore, requirements traceability evolved to something that is today also known as *software traceability* (cf. [2]), which is about tracing any kind of software artifact. According to this trend, Aizenbud-Reshef et al. provide a much broader definition of traceability in [3]:

> *"We regard traceability as any relationship that exists between artifacts involved in the software-engineering life cycle."*

In recent software traceability approaches, traceability information are usually defined by means of metamodels and, thus, recoded into traceability models, which are instances of these metamodels. The identification of traceability information can be obtained manual, semi-automatic due to false positives [4], [5], [6], [7], and even fully automatic [8], [9], [10], [11], [12], [13], [14], [15], [16]. Not only identification of traceability plays an important role, but also the continuous maintenance of traceability information, which should be automated [17] because otherwise traceability information get deteriorated due to constant changes.

Due to the increasing heterogeneity and complexity of models and model operations in MDE, global model management (GMM) becomes indispensable to MDE. There are basically two primary concerns of GMM: manage the existence of heterogenous models [18], [19] (e.g., repository for models) and manage heterogenous model operations (transformations) [20], [21], [22].

In [23], [24], which can be seen as state-of-the-art GMM approach, Bézivin et al. show how traceability is considered in GMM. There, traceability information is generated as side-effect of model transformations (ATL in their case [10]).[1] The generated traceability information is integrated into a GMM by recording them into traceability models that only reflect the applied model operation. Thus, traceability gets subject to GMM.

Capturing traceability information from model transformations automatically is important, but this is a somehow a restricted perspective to software traceability in GMM. When looking especially at [24], one can see that this approach is itself subject to traceability because their foundation, a megamodel (cf. [18], [19]), can be interpreted as a traceability model itself. In [24] the following definition is given: *"A megamodel contains relationships between models."* The megamodel in [24] records traceability information at two different levels. Global level traceability captures the application of model operations (in this case only ATL transformations [10]), and local level traceability information encode detailed fine-grained relationships between model elements. The local level traceability is

[1]There are also other model transformation and synchronization approaches that support the generation of traceability information [11], [12], [13], [15].

1534-5351/11 $26.00 © 2011 IEEE
DOI 10.1109/CSMR.2011.58

recorded into separate weaving models related to the global level traceability information. This principle is illustrated in Figure 1.

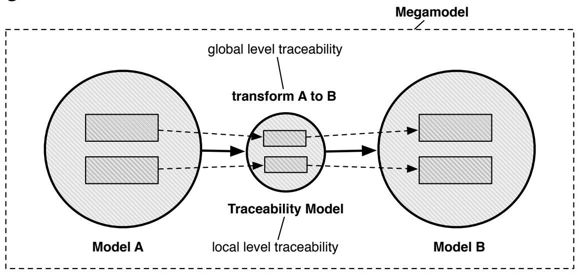

Figure 1. Simplified illustration of a megamodel

My primary hypothesis is that considering GMM from a traceability perspective provides new possibilities and that it will generally improve GMM through easing the software development or maintenance process.

The approach that I will shown in my PhD thesis will consider GMM from a software traceability perspective. Therefore, I go back to software traceability and apply techniques from that discipline to improve GMM. The approach is separated into the following working packages:

- **WP1** – automated traceability identification beyond the manual application of model operations, i.e. information retrieval, in the context of GMM. Thus, not only model operations should be automatically identified but also overlaps between models, for example.
- **WP2** – automated and scalable maintenance of existing traceability information preventing them from deteriorating in the context of GMM.
- **WP3** – utilize traceability to automatically build and execute compositions of model operations. The composition of model operations will be based on traceability information.

To validate my hypothesis, I will apply a prototype to a given case study in the area of software product modeling and deployment. Further, I will evaluate my approach against existing GMM approaches and show that my approach provides not yet available possibilities. The rest of this paper is structured as follows: In Section II a brief overview of my approach is given. In Section III related work that is closely related to my approach is discussed. Finally in Section IV, the current state of my research and a conclusion is given.

## II. Approach Overview

### A. Definition and Recording (WP1)

The foundation of my approach is a traceability model that is sufficient for software traceability in the context of MDE as well as for GMM. The basic constituents of this traceability model are:

- **artifact** – is the representation of any kind of artifact, e.g., models, model elements, documents, etc. These are possible traceable entities.
- **traceability link** – interrelates a set of artifacts. A traceability link can be a mapping, a dependency or a trace of a model operation. A traceability link is *n-ary* and can represent any direction.

The traceability model allows the specification of traceability links at different levels-of-detail. Thus, it can record global level traceability links (e.g., between models) as well as local level traceability links (e.g., between elements of models). Figure 2 illustrates abstract instantiation of such a traceability model.

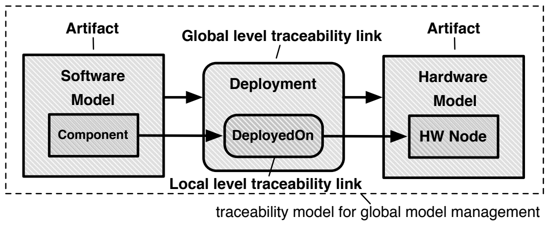
Artifact
Global level traceability link
Artifact
Software Model
Component
Deployment
DeployedOn
Hardware Model
HW Node
Local level traceability link
traceability model for global model management

Figure 2. Example of global level and local level traceability links

The semantic of traceability links is not restricted to a specific type. A user can define the types on her own. We however distinguish between two kinds of traceability links:

- **fact traceability link** – is any traceability link which intention is to show some overlap between artifacts (classical traceability link). As long as such a traceability link exists, its intention is valid. If the overlap does not exist, due to changes, the traceability link has to be removed. The traceability links in Figure 2 are typical fact links.
- **obligation traceability link** – is any traceability link which intention is expressed by means of some additional operational semantic. That can be any model operation. The intention can be invalidated whenever the related artifacts change. Invalidation does not enforce deletion of the traceability link, but the (re-)application of the related model operation. The model operation is applied by taking the inputs and outputs of the related obligation traceability link as parameter for execution.

### B. Identification (WP1)

When we start with a "fresh" traceability model, traceability links are automatically inferred by an automated batch identification process. This process requires a *creation rule* for each traceability link that can automatically inferred. The creation rules are implemented by means of *graph-rewriting-rules*.[2] The creation rule encodes the structural condition when the traceability link should exist and then automatically creates the traceability link in that context.

Nevertheless, only providing automated identification is not feasible because sometimes information is missing for automatically identifying whether a traceability link should exist or not, e.g., deciding whether a certain model operation between two models should be applied or not. Thus,

[2]Currently, we support Story Diagrams [25]

the approach also supports the following two identification modes:

- **manual identification** – a user manually creates traceability links by means of a unified user interface. This can also comprise triggering the creation of obligation traceability links and, thus, the execution of a model operation.
- **semi-automated identification** – the identification process automatically identifies traceability links which, however, need to be manually confirmed. This mode can be used to automatically find possible applications of model transformations. Thus, it supports the user through some sort of guidance.

## *C. Maintenance (WP2)*

We already provide a basic maintenance approach by introducing deletion rules as counterpart to creation rules. These rules are the inverse of creation rules and are used to decide whether existing traceability links should be de-establish. Thus, a batch identification process with creation and deletion rules is already a maintenance process.

However, a simple batch identification process, as provided in WP1, is not sufficient for managing large software projects. Thus, an incremental version of the identification process is provided, which is scalable and, therefore, applicable to even large software projects.

## *D. Utilization (WP3)*

The approach is able to compose model operations by utilizing nesting of traceability links. It is distinguished between *horizontal* and *vertical composition* as explained in the following:

- **horizontal composition** – traceability links are composed by means of overlapping inputs and outputs. This kind of composition can be considered as a chain of traceability links and, therefore, also model operations.
- **vertical composition** – traceability links are composed by means of nesting.

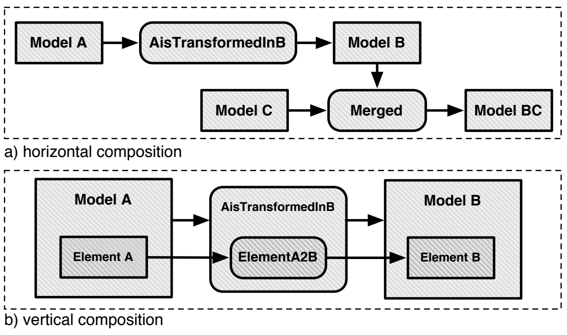

Figure 3. Horizontal and vertical composition of traceability links

The approach provides an *execution process* that is able to execute this compositions appropriately. It is obvious that the execution process will influence the identification and maintenance process because it modifies the artifacts of the traceability model. Thus, there is a strong relationship between them, which needs to be well defined.

Figure 3 shows a horizontal composition (a) and a vertical composition (b). In (a) “Model B” is the output of “AisTransformedInB” and also the input of “Merged”. Thus, the model merge operation depends on the outcome of the model transformation related to “AisTransformedInB”. (b) shows that “AisTransformedInB” is composed of another obligation traceability link called “ElementA2B”. The vertical composition has the benefit that different technologies can be organized toward a common goal (e.g., transforming “Model A” to “Model B”).

## III. Related Work

Salay et al. introduced the notion of *macromodeling* in [26], which is somehow similar to *megamodeling*. Their foundational model is called *macromodel*. In contrast to megamodels, macromodels reflect existing and required models as well as relationships. Their approach is used to guide the modeling process. It supports automatic identification of relationships by means of overlaps. However, they do not focus on maintenance nor they support tracing model operations.

Recently, Bézivin et al. published a subsequent continuation of [24] in [27]. In that paper the authors agree that other techniques for identifying traceability links in GMM than model transformations are necessary. However, the paper does not present a solution to that.

## IV. Conclusions

Recently, I started my fourth year as PhD student. Beside foundational research in the area of software traceability and GMM, I have implemented working package WP1 and partly WP3, which is already published in [28]. There it is shown that is it feasible to apply creation/deletion rules to automatically establish/de-establish traceability links in a GMM context. Working package WP2 is implemented but not yet published as contribution to the research community. The foundation for WP3 is already given by WP1 and a first prototype has been implemented. However, the concepts has to be generalized and to be published.

When each working package is implemented, I will put everything into a compound implementation that will be provided as open-source to the community in 2011.

## References

[1] O. Gotel and C. W. Finkelstein, “An analysis of the requirements traceability problem,” 1994, pp. 94–101.

[2] G. Spanoudakis and A. Zisman, *Software Traceability: A Roadmap*, 3rd ed., ser. Handbook of Software Engineering & Knowledge Engineering: Recent Advances. World Scientific Publishing Company, 2005, pp. 395–428.

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

[4] A. Egyed, “A scenario-driven approach to traceability,” in *ICSE ’01: Proceedings of the 23rd International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 123–132.

[5] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, “Recovering traceability links in software artifact management systems using information retrieval methods,” *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 4, p. 13, 2007.

[6] A. De Lucia, R. Oliveto, and G. Tortora, “Adams re-trace: traceability link recovery via latent semantic indexing,” in *ICSE ’08: Proceedings of the 30th international conference on Software engineering*. New York, NY, USA: ACM, 2008, pp. 839–842.

[7] W. Jirapanthong and A. Zisman, “Xtraque: traceability for product line systems,” *Software and Systems Modeling*, vol. 8, pp. 117–144, 2009.

[8] J. Richardson and J. Green, “Traceability through automatic program generation,” 2003.

[9] N. Aizenbud-Reshef, R. F. Paige, J. Rubin, Y. Shaham-Gafni, and D. S. Kolovos, “Operational Semantics for Traceability,” in *ECMDA-TW’05: Proceedings of 1st Workshop on Traceability, Nurnberg, Germany*. SINTEF, 7-10 November 2005, pp. 7–14.

[10] F. Jouault, “Loosely coupled traceability for atl,” in *In Proc. of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability*, 2005, pp. 29–37.

[11] J. R. Falleri, M. Huchard, and C. Nebut, “Towards a Traceability Framework for Model Transformations in Kermeta,” in *ECMDA-TW’06: Proc. of 2rd Workshop on Traceability, Bilbao, Spain*, T. Neple, J. Oldevik, and J. Aagedal, Eds. SINTEF, 10-13 July 2006.

[12] G. K. Olsen and J. Oldevik, “Scenarios of Traceability in Model to Text Transformations,” in *Model Driven Architecture- Foundations and Applications, Third European Conference, ECMDA-FA 2007, Haifa, Israel*, 2007, pp. 144–156.

[13] I. Kurtev, “State of the Art of QVT: A Model Transformation Language Standard,” in *Applications of Graph Transformations with Industrial Relevance: Third International Symposium, AGTIVE 2007, Kassel, Germany,*. Berlin, Heidelberg: Springer-Verlag, 10-12 October 2007, pp. 377–393.

[14] B. Amar, H. Leblanc, and B. Coulette, “A Traceability Engine Dedicated to Model Transformation for Software Engineering,” in *ECMDA Traceability Workshop 2008, Berlin, 12/06/08-12/06/08*, J. Oldevik, G. K. Olsen, T. Neple, and R. Paige, Eds. Springer, juin 2008, pp. 7–16.

[15] H. Giese and R. Wagner, “From model transformation to incremental bidirectional model synchronization,” *Software and Systems Modeling (SoSyM)*, vol. 8, no. 1, 28 March 2009.

[16] H. Schwarz, J. Ebert, and A. Winter, “Graph-based traceability: a comprehensive approach,” *Softw. Syst. Model.*, vol. 9, pp. 473–492, September 2010.

[17] P. Mäder, O. Gotel, and I. Philippow, “Enabling Automated Traceability Maintenance Through the Upkeep of Traceability Relations,” in *Proceedings 5th European Conference on Model-Driven Architecture Foundations and Applications (ECMDA2009) – LNCS5562*, Enschede, Netherlands, Jun. 2009, pp. 174–189.

[18] J. Bézivin, F. Jouault, and P. Valduriez, “On the Need for Megamodels,” in *Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2004.

[19] F. Allilaire, J. Bézivin, H. Brunelière, and F. Jouault, “Global Model Management In Eclipse GMT/AM3,” in *Proceedings of the Eclipse Technology eXchange workshop (eTX) at ECOOP’06*, 2006.

[20] T. Reiter, E. Kapsammer, W. Retschitzegger, and W. Schwinger, “Model integration through mega operations,” in *Proc. of the Int. Workshop on Model-driven Web Engineering (MDWE), Sydney*, 2005.

[21] D. S. Kolovos, R. F. Paige, and F. A. C. Polack, “On Demand Merging of Traceability Links with Models,” in *ECMDA-TW’06: Proceedings of 2rd Workshop on Traceability, Bilbao, Spain*. SINTEF, 10-13 July 2006.

[22] D. S. Kolovos, R. Paige, and F. Polack, “A Framework for Composing Modular and Interoperable Model Management Tasks,” in *MDTPI workshop, EC-MDA, Berlin, Germany*, June 2008.

[23] J. Bézivin, F. Jouault, P. Rosenthal, and P. Valduriez, “Modeling in the Large and Modeling in the Small,” in *Model Driven Architecture*, ser. Lecture Notes in Computer Science (LNCS), vol. 3599/2005. Springer-Verlag, 2005, pp. 33–46.

[24] M. Barbero, M. D. D. Fabro, and J. Bézivin, “Traceability and Provenance Issues in Global Model Management,” in *ECMDA-TW’07: Proceedings of 3rd Workshop on Traceability, Haifa, Israel*, J. Oldevik, G. K. Olsen, and T. Neple, Eds. Haifa, Israel: SINTEF, June 2007, pp. 47–55.

[25] H. Giese, S. Hildebrandt, and A. Seibel, “Feature Report: Modeling and Interpreting EMF-based Story Diagrams,” in *Proceedings of the 7th International Fujaba Days*, 2009.

[26] R. Salay, J. Mylopoulos, and S. Easterbrook, “Managing Models through Macromodeling,” in *ASE ’08: Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 447–450.

[27] F. Jouault, B. Vanhooff, H. Brunelière, G. Doux, Y. Berbers, and J. Bézivin, “Inter-DSL Traceability and Navigability Support by Combining Megamodeling and Model Weaving,” in *In Proc. of Special Track on the Coordination Models, Languages and Applications at the 25th Symposium On Applied Computing (SAC 2010), Sierre, Switzerland, March 22-26*, 2010.

[28] A. Seibel, S. Neumann, and H. Giese, “Dynamic Hierarchical Mega Models: Comprehensive Traceability and its Efficient Maintenance,” *Software and System Modeling*, vol. 009, no. s10270, 2009.