

# Metamodel-based Tool Integration with MOFLON

Carsten Amelunxen, Felix Klar, Alexander Königs, Tobias Rötschke, Andy Schürr  
Technische Universität Darmstadt  
Real-Time Systems Lab  
Merckstr. 25  
D-64283 Darmstadt, Germany  
contact@moflon.org

## ABSTRACT

Nowadays, a typical software development process involves many developers which participate in the development process by using a wide variety of development tools. As a consequence, the data representing the project as a whole is distributed over different development tools. For the purpose of consistency, maintainability, and traceability it is an essential task to be aware of the relationships between semantic equivalent data in different tool repositories. The Real-Time Systems Lab at the Technische Universität Darmstadt performs research in the area of tool and metamodel integration to provide solutions to overcome this gap. In this demonstration we present the metamodeling framework MOFLON that addresses these issues by bringing together the latest OMG standards with graph transformations and triple graph grammars. Using MOFLON, developers can generate code for specific tools needed to perform analysis and transformation on one development tool or to incrementally integrate data of different modeling tools.

## Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques—*Computer-aided software engineering (CASE)*

## General Terms

Design, Languages

## Keywords

model-driven software development, tool integration, triple graph grammar, model transformation

## 1. INTRODUCTION

During the last decades, industrial software engineers have been facing ever growing complexity in their development processes. This complexity does not only originate from the sheer code size produced by dozens of developers during the evolution of long-lived industrial projects to reflect changing and sprawling requirements. Apart from that, it is also very difficult to consistently keep up the whole project with evolving technologies like hardware platforms, operating systems, tools, and programming as well as a plethora of

modeling languages. OMG's vision of model-driven application development (MDA) helps to manage the first kind of complexity by defining corresponding layers of abstraction for each project. However, the related tools even add to the second kind of complexity.

Taking the development of embedded systems in the automobile industry as an example, the development processes involve quite a number of different tools each specialized in certain tasks (e.g. requirements engineering, modeling of software and hardware functionality, test case maintenance). Thus, the data of a project as a whole is distributed over different tools. Typically, these tools are commercial off-the-shelf (COTS) that are rarely designed to integrate with each other. Nevertheless, the data stored in the separate tools is related and must be kept consistent. Thus, we need an appropriate tool integration approach that supports the rapid development of analysis, transformation, and integration operations on models that are stored in different COTS tools.

For this purpose we are developing the MOFLON metamodeling framework [19], which features metamodel editing, code generation, XMI import/export, graph transformations, and triple graph grammars. MOFLON is used to create tools that analyze, transform and integrate models which are created using already existing COTS tools.

## 2. METAMODELING WITH MOFLON

The integration of development tools, or rather of the metamodels which are implemented by the development tools, requires a metamodeling language that provides sufficient features for clearly structured and well-formed metamodels. Since, known from experience, metamodels of development tools are usually quite big and complex, only a metamodeling language which is able to deal with large and complex specifications is suitable for the purpose of metamodel integration. In other words, the potential of an integration approach is directly dependent on the used metamodeling language's capacity. The most promising approach concerning the size of specifications is provided by OMG's latest metamodeling standard MOF 2.0 [14].

The Meta Object Facility (MOF) basically provides, for the purpose of metamodeling, an optimized version of the popular and well-known class diagrams of the Unified Modeling Language (UML) [16]. The main benefit of MOF 2.0 compared to its predecessors consists of its sophisticated features for modularization and refinement. Modularization is on the one hand provided by the advanced package concept which allows for the specification of namespace hierarchies

and on the other hand by an association concept which treats associations as first class modeling constructs that can be instantiated and queried as an independent link repository. Particularly, the latter feature is absolutely necessary, since for the purpose of integration, it is very important that links can be established without modifying the involved classes.

Refinement, which has been available for classes since the early days of visual data modeling, is in terms of MOF 2.0 expanded to encompass complete packages and associations as well. The feature called package merge offers the possibility to refine complete packages as collection of classes and associations and, therefore, allows to spread metamodels over several packages on different levels of abstraction and facilitates reuse of such metamodel abstractions. Since, such a package wide refinement is only possible, if it can be transferred to the included classes and associations, the refinement of associations in form of subsetting and redefinition becomes even more important. MOFLON provides full MOF 2.0 support by offering graphical editors for the creation of MOF 2.0 compliant metamodels as well the considerably more important feature of code generation. It generates a repository implementation which is compliant to the Java Metadata Interface (JMI) [5] and, therefore, provides standardized [5] interfaces for the creation, storage, access, discovery, and exchange of metadata as well as interfaces for reflective usage.

Beside the specification of a modeling language’s abstract syntax with MOF 2.0, MOFLON provides the possibility to specify a language’s static semantics with the Object Constraint Language (OCL) [15]. Beside pre- and postconditions for operations, the textual constraint language OCL particularly allows for the specification of invariants and derivation rules. MOFLON includes specified constraints in its code generation process and generates repository implementations which are enriched by code for the surveillance of invariants or the calculation of derivation rules, respectively. On the one hand, OCL constraints can be used for the specification of a modeling language’s abstract syntax as well as on the other hand for the specification of the modeling language’s application in form of modeling guidelines. The metamodel-based specification of modeling guidelines is an important task since in some domains, for instance in the domain of embedded systems, models have to adhere to high quality standards. Thus, it is very beneficial to consider modeling guidelines in form of OCL statements already at the stage of metamodeling.

The combination of MOF and OCL enables MOFLON to support the specification of a modeling language’s abstract syntax and static semantics. Thus, the last component for a language specification which meets the before mentioned demands for being integrable, is a technique for the specification of dynamic semantics. For this purpose, MOFLON integrates the technique of Story Driven Modeling (SDM) [21] which is basically a combination of UML activity diagrams and graph transformation rules. This technique is used for the specification of classes’ behavior, whereas the control flow is modeled with activity diagrams which, in turn, control the application of local in-place model transformations. A model transformation rule is declaratively specified by the left-hand and right-hand side of a graph transformation.

The crucial point of the integration of graph transformation techniques into MOFLON is, that MOF 2.0 acts as graph schema language rather than old versions of UML or

proprietary approaches, which is a considerable novelty in the community of graph transformation tools. Thus, the promising features of MOF 2.0 like association subsetting and redefinition can be applied in the context of graph transformations. The great benefit of SDM consists of the possibility to directly map the specified transformation onto source code. Thus, MOFLON’s code generation mechanism is finally able to generate repository implementations which are additionally able to analyze and modify models on the base of complex metamodel specifications. Hence, it is not only possible to enrich a metamodel with formal specifications of modeling guidelines, it is also possible to formally specify how violations of such guidelines can be automatically repaired (see [2] for more details). Fig. 1 gives an overview about the before mentioned components as part of MOFLON’s internal architecture and the generated repositories.

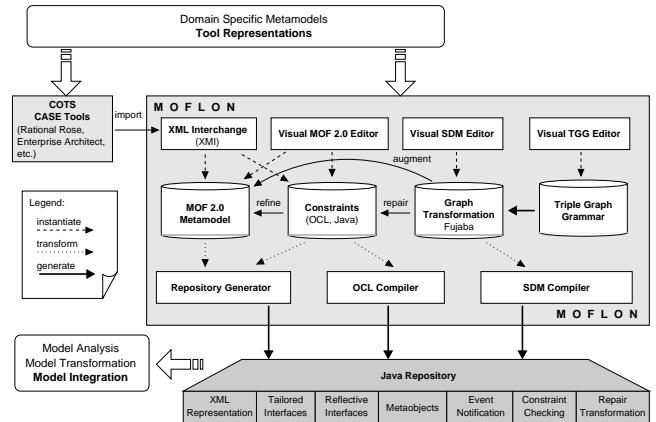


Figure 1: Overview MOFLON architecture

Considering the purpose of metamodeling, this graph transformation technique can be used to specify a modeling language’s dynamic semantics and as such, together with static semantics specified in OCL based on a MOF 2.0 compliant specification of the language’s abstract syntax, enables MOFLON to generate a repository which meets the demands of tool integration. In the following we will demonstrate how MOFLON’s combined metamodeling approach can be used as base for the application of integration techniques between different metamodels.

### 3. TOOL INTEGRATION

The tool integration approach realized in MOFLON is based on the technique of triple graph grammars (TGG). TGGs have been introduced by Schürr in 1994 [18]. They can be used to visually define the simultaneous evolution of two graphs, the construction of links between nodes in both graphs and bidirectional model transformation. These links form a third graph, which holds explicit information about which nodes from both graphs correspond to each other. The third graph also holds information about the transformation process itself. This allows to check for consistency between source and target graph and to recover consistency if it has been destroyed. Incremental updates are also possible by this approach. If nodes have changed in one of the graphs—after links have already been established between nodes—these changes may be propagated to the other graph.

Originally TGGs have been introduced in the world of graphs, but they have been applied to the world of meta-modeling, also [8]. Throughout this paper, we will use some terms from both worlds interchangeably. The following table shows the terms that correspond to each other:

world of graphs	world of metamodeling
schema	metamodel
graph	model
node type	class
node	object
edge type	association
edge	link

TGGs consist of a schema and a set of declarative rules. The schema declares types for correspondence links and associates these correspondence link types with classes from the metamodels of the tools that will be integrated. The schema supports concepts for modularization, refinement and reuse [7] which is an advantage when dealing with large integration specifications, i.e. when the development tools have many elements that must be integrated. Each link type owns a TGG rule that defines a mapping between elements of the integrated metamodels. During runtime, instances of link types will play the role of traceability links that map elements of one metamodel to elements of the other metamodel and vice versa.

To be able to integrate two development tools, two metamodels are required that represent the structure of the tool’s data. These metamodels may be created by MOFLON or by another CASE-tool (e.g. IBM’s Rational Rose) and then be imported into MOFLON. If this prerequisite is fulfilled, we begin with the specification process of the TGG.

After all elements that should be integrated have been identified and it is clear which elements from the source metamodel correspond to elements in the target metamodel, we start creating the TGG schema. For this purpose, we use the TGG Editor that comes as part of the MOFLON metamodeling framework. As the TGG schema specifies the structure of links, it defines which elements of the integrated metamodels may be linked with each other. The schema definition somehow naturally affects the definition of TGG rules. As mentioned earlier, a TGG rule is declarative and is always associated with exactly one correspondence link type. Each rule is responsible for creating links between corresponding elements. In our approach the rule may only create links that are instances of the link type this rule belongs to. A rule typically contains a context matching part and a creation part, whereas the creation part is only executed, if the matching part succeeds. For a more detailed description of TGG schemas and TGG rules please refer to [9] and [7].

After the TGG has been successfully created, the TGG is translated into a MOFLON MOF representation. In this step correspondence link types are translated into classes and associations and declarative TGG rules into operational rules. An operational rule is represented as an operation that is owned by the class of the corresponding link type. The operation contains an SDM diagram that is derived from the declarative TGG rule. From each declarative TGG rule a set of operational rules is created. These operational rules can be used to perform forward and backward transformation of tool data as well as link creation or deletion and consistency checking.

The MOF representation of the TGG is an intermediate representation only. A TGG repository is generated from it which contains the mapping directives (i.e. operational rules) that are able to integrate data of both development tools. Figure 2 schematically shows how the repositories are used by our integration tool “integrator” to perform tool integration tasks.

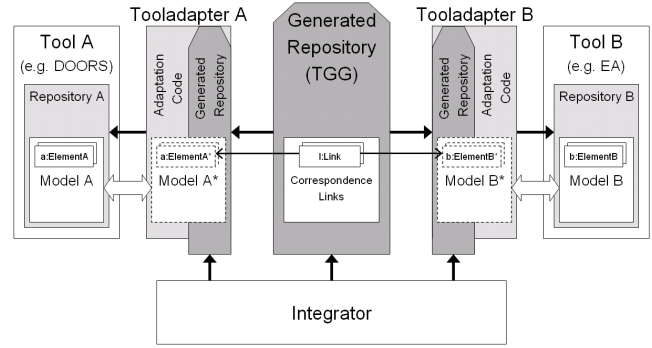


Figure 2: Schematic view of the integration process

The integrator operates on the reflective JMI interfaces that are provided by the generated repositories. As mentioned in chapter 2 for each tool a repository is generated. These repositories must be adapted, so they can directly modify tool data. The adaptations that have to be done are tool specific and depend on how a tool gives access to its data. We are currently doing research on how adapters can be partly generated. The main tasks of the integrator are to establish correspondence links between source and target model by invoking according operations in the repositories. And to keep track of changes in source and target models and incrementally propagate these changes to the other model. The integrator also visualizes elements of both tools and links between these elements and supports basic navigation operations on traceability links.

## 4. RELATED WORK

Like MOFLON, there are a number of approaches for metamodeling and model transformation. In this section we focus on tools that are specifically interesting for our approach.

Some tools like GREAT [1], and MDR [11] value OMG standard-compliant schemata like MOF and UML, while GME [1], *MetaEdit+* [12], PROGRES [17], prefer proprietary metamodels. Fujaba [21] mixes standard and proprietary elements, EMF Transformation Engine / GMF [10, 20] only implements EMOF, a small subset of the current standard, and tools like AToM<sup>3</sup>[4] use ER-Diagrams that could be considered “standard” some time ago.

Only MOFLON puts a strong emphasis on complete standard compliance with MOF 2.0 to benefit from its new features. Among these are strong *modularization and refinement* possibilities. Tools like [4] provide no schema modularization at all, others like [21, 1, 12, 17] provide either hierarchies or view mechanism to structure data. MOFLON uses package imports, package merges, element imports, re-definition of association ends etc. with full effect on identifier visibility not only in schemata but also constraints and graph transformations.

Many tools [4, 1, 20, 13, 3], often called meta-CASE tools, deal with the *concrete syntax* of modeling languages to create diagram editors. MOFLON is more about model analysis, transformation and integration and hence, does not support concrete syntax.

MOFLON provides *local model transformations* through graph transformations, which is also true for [17, 21, 1, 4]. Other tools [10, 6] only provide textual model transformations or none at all [11]. Like MOFLON, only some of these tools [17, 21, 1] use visual *rule application strategies* to compose large transformations. Only Fujaba and hence MOFLON use standard-compliant UML activity diagrams to this end.

MOFLON and Tefkat[10] provide declarative model-to-model transformations that are QVT-like. Opposed to that, GReAT only provides the possibility to define operational unidirectional model-to-model transformations.

## 5. CONCLUSION

MOFLON combines the OMG metamodeling standards MOF 2.0 and OCL 2.0, for the specification of a modeling language's abstract syntax and static semantics, with the technique of SDM graph transformations for the specification of dynamic semantics. Since MOFLON basically aims to integrate development tools (with existing concrete syntax), the definition of a modeling language's concrete syntax, as done in traditional meta-CASE tools, is out of MOFLON's scope. MOFLON rather provides the specification of declarative integration rules between different meta-models with triple graph grammars. The integration rules can finally be translated into executable Java code which synchronizes semantically equivalent data in different development tools at runtime.

## 6. REFERENCES

- [1] A. Agrawal, T. Levendovszky, J. Sprinkle, F. Shi, and G. Karsai. Generative Programming via Graph Transformations in the Model Driven Architecture. In *Proc. Workshop on Generative Techniques in the Context of Model Driven Architecture*, Nov. 2002.
- [2] C. Amelunxen, E. Legros, A. Schürr, and I. Stürmer. Checking and Enforcement of Modeling Guidelines with Graph Transformations. In A. Schürr, M. Nagl, and A. Zündorf, editors, *Proceedings of the Third International Symposium on Applications of Graph Transformations with Industrial Relevance*, October 2007.
- [3] B. Böhlen, D. Jäger, A. Schleicher, and B. Westfechtel. UPGRADE: A Framework for Building Graph-Based Interactive Tools. In T. Mens, A. Schürr, and G. Taentzer, editors, *Proc. International Workshop on Graph-Based Tools*, volume 72(2) of *Electronic Notes in Theoretical Computer Science*, 2002.
- [4] J. De Lara Jaramillo, H. Vangheluwe, and M. A. Moreno. Meta-modelling and Graph Grammars for Multi-Paradigm Modelling in AToM<sup>3</sup>. *Software & Systems Modeling*, 3(3):194–209, Aug. 2004.
- [5] R. Dirckze. *Java<sup>TM</sup> Metadata Interface (JMI) Specification, Version 1.0*. Unisys, June 2002.
- [6] F. Jouault and I. Kurtev. Transforming Models with ATL. In *Proc. Workshop on Model Transformations in Practice*, 2005.
- [7] F. Klar, A. Königs, and A. Schürr. Model transformation in the large. In *The 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 285–294, Dubrovnik, Croatia, September 03 - 07 2007. ACM New York, NY, USA.
- [8] A. Königs. Model Transformation with Triple Graph Grammars. In *Proc. Workshop on Model Transformations in Practice*, 2005.
- [9] A. Königs and A. Schürr. Tool Integration with Triple Graph Grammars - A Survey. In R. Heckel, editor, *Proc. SegraVis School on Foundations of Visual Modelling Techniques*, volume 148 of *Electronic Notes in Theoretical Computer Science*, pages 113–150, Amsterdam, 2006. Elsevier Science Publ.
- [10] M. Lawley and J. Steel. Practical Declarative Model Transformation With Tefkat. In J. Bézivin, B. Rumpe, A. Schürr, and L. Tratt, editors, *Proc. Workshop on Model Transformations in Practice*, October 2005. <http://sosym.dcs.kcl.ac.uk/events/mtip05/>.
- [11] M. Matula. *NetBeans Metadata Repository*. SUN Microsystems, März 2003.
- [12] MetaCase. MetaEdit+@metaCASE tool. <http://www.metacase.com>, 2006.
- [13] M. Minas. Concepts and Realization of a Diagram Editor Generator-based on Hypergraph Transformation. *Science of Computer Programming*, 44:157–180, 2002.
- [14] Object Management Group. *Meta Object Facility (MOF) Core Specification*, January 2006. formal/06-01-01.
- [15] Object Management Group. *Object Constraint Language*, May 2006. formal/06-05-01.
- [16] Object Management Group. *Unified Modeling Language: Superstructure*, February 2007. formal/2007-02-05.
- [17] A. Schürr, A. Winter, and A. Zündorf. *PROGRES: Language and Environment*, volume 2, pages 487–550. World Scientific, 1999.
- [18] Schürr. Specification of Graph Translators with Triple Graph Grammars. In Mayr and Schmidt, editors, *Proc. WG'94 Workshop on Graph-Theoretic Concepts in Computer Science*, pages 151–163, 1994.
- [19] Team MOFLON. MOFLON. <http://www.moflon.org/>.
- [20] The Eclipse Foundation. Eclipse Graphical Modeling Framework. <http://www.eclipse.org/gmf/>, 2006.
- [21] A. Zündorf. *Rigorous Object Oriented Software Development*. University of Paderborn, 2001. Habilitation Thesis.