Development of Tool Extensions with MOFLON

Ingo Weisemöller, Felix Klar, and Andy Schürr

Fachgebiet Echtzeitsysteme
Technische Universität Darmstadt
D-64283 Darmstadt, Germany
{weisemoeller|klar|schuerr}@es.tu-darmstadt.de

Abstract. The increasing complexity of embedded systems is accompanied by an increasing number and complexity of models, modeling languages and tools in the development process. This results in a need for appropriate tool support at the metamodel level. Besides the necessity to develop new languages and tools, there is also a large demand for extensions to existing tools as well as for integration frameworks. Such frameworks ensure consistency between data that is distributed over several tools. In this chapter, we present MOFLON, a metamodeling tool primarily focused on tool extension and integration. It adopts several standards such as MOF 2.0 and JMI. It also supports story driven modeling as a means of describing on-model transformations as well as a combination of MOF QVT and triple graph grammars for model-to-model transformations and integration. We present a typical application of these features to tools used in the development of embedded systems.

1 Introduction

Because the number of software development processes, especially for embedded systems, has rapidly increased recently, the number of modeling languages and commercial off-the-shelf (COTS) modeling tools has increased as well. Therefore, documents created with these modeling tools are also becoming harder to manage and maintain. These documents and models may be difficult to understand and to develop further. Therefore, modeling guidelines are a wide spread approach to improve readability and maintainability of these documents. Such guidelines may also enforce properties of the model that are necessarily required for automatic processes, such as code generation. Data spread across several documents may be redundant and needs to be kept consistent. These documents are usually developed with different COTS tools. Most of such tools neither provide proper interfaces to couple them with one another, and they do not provide a way to define domain specific rules for data consistency between several documents. Thus, alignment and adjustment of this data is usually performed manually, which results in considerable efforts and costs.

Since new tools are not usually an option in ongoing processes, tool extensions are a more adequate way to enforce modeling guidelines and to ensure consistency between several models. The metamodeling tool MOFLON is focused

[WKS10] I. Weisemöller, F. Klar, A. Schürr
Development of Tool Extensions with MOFLON
In: MBEERTS: Model-Based Engineering of Embedded Real-Time Systems, p. 337 - 343
International Dagstuhl Workshop
Dagstuhl Castle, Germany, LNCS 6100, Springer Berlin, October 2010
Note: The original publication is available at www.springerlink.com.
www.se-rwth.de/publications

on efficient development of such extensions. We use MOFLON to develop tool adapters that comply to the Meta Object Facility (MOF) [1] and to the Java Metadata Interface (JMI) [2], and thus provide standardized access to model data. Based on these adapters,we use model transformations to describe rules for analysis and semi-automatic repair of models according to guidelines. MOFLON also allows to define model-to-model transformations and consistency rules in a declarative notation based on MOF Query/View/Transformation (QVT) [3].

The remainder of this chapter is outlined as follows: In Section 2 we describe the core features and briefly introduce the standards adopted by MOFLON. Section 3 provides an overview of usage scenarios for MOFLON, and in Section 4 we give a short summary and present some ideas for future versions of MOFLON.

2 History and Overview of Features

The development of MOFLON began in 2002. Based on code generated by the MOF Model Compiler (MOMoC) [4] from a simplified version of the MOF metamodel, we developed a graphical editor as a plugin for the UML tool Fujaba [5]. Besides this editor, the graph transformation environment of Fujaba was reused for model transformations. This step required a refactoring of the existing environment in order to make it work on an abstract metamodel interface, which could be implemented by plugins. Having completed these steps successfully, we released MOFLON 1.0 in December 2006.

More recent versions of MOFLON introduced an editor and code generator for model-to-model transformation rules based on triple graph grammars (TGGs) (MOFLON 1.1, July 2007), a compiler for the Object Constraint Language (OCL) [1] based on the Dresden OCL toolkit [6] (MOFLON 1.2, December 2007) and modularization concepts for model-to-model transformations (MOFLON 1.3, December 2008).

2.1 MOF Editor and Code Generation for MOF models

MOFLON adopts the MOF 2.0 standard [1] by the Object Management Group (OMG). MOF compliant metamodels describe the abstract syntax of modeling languages in a notation based on UML class diagrams. MOFLON supports the complete MOF (CMOF); in comparison to its subset essential MOF (EMOF), which is, for instance, supported by the Eclipse Modeling Framework [7], CMOF has much more sophisticated association and modularization concepts, which are substantial for metamodeling in the large. Constraints can be added to metamodels using the Object Constraint Language (OCL) [1] in MOFLON.

The code generated from metamodels by MOFLON complies to the JMI standard by Sun. This defines tailored interfaces, which are specific to the respective metamodel, and reflective interfaces, which provide generic access to model and metamodel data. Our mapping from MOF 2.0 to JMI is an extension of the JMI mapping for MOF 1.4 defined by the OMG. Because JMI does not describe an event mechanism, MOFLON metamodels implement the interface of Netbeans' metadata repository (MDR) [8] for events.

2.2 Additional Frontends

Besides the graphical MOF editor, MOFLON provides import modules for several other frontends. UML models can be imported from Rational Rose or Sparx Systems Enterprise Architect. For Enterprise Architect, there is also a plugin [9] that introduces MOF diagrams, provides a toolbox for editing MOF models, and performs checks on these models to ensure they can be imported and used for code generation in MOFLON. The import from UML tools is based on the XML Metadata Interchange (XMI) standard. Because many tools have their own extensions to or interpretations of XMI, one can run XSL Transformations on the XMI data before the import. This results in low efforts to develop import modules for further tools. Currently, we are also working on a textual frontend.

2.3 Model Transformations

Since one of our core areas of application is model analysis and repair, MOFLON can be used to describe rules and constraints for this. We make extensive use of OCL constraints, pattern matching and model transformations for model analysis and repair. MOFLON uses the transformation engine provided by Fujaba, with a set of code generation templates that has been adopted to MOF and JMI.

Model transformations in MOFLON are described in story diagrams [10], which are a combination of UML activity diagrams and an adopted version of collaboration diagrams. The control flow of a transformation is specified in an activity diagram. Inside each activity, pattern matching and replacement is described in an extended collaboration diagram. Chapter 14 of this book gives an example of model transformations with MOFLON.

2.4 Triple Graph Grammar Editor

Model-to-model transformations can be specified using the MOFLON triple graph grammar editor. TGGs [11] are a formal transformation language that allows to relate model elements with each other. TGGs specify bidirectional model-to-model transformations in a declarative manner. TGG rules can be translated into operational transformation rules. These can be used to perform forward and backward transformations as well as consistency checks on related models. TGGs are closely related to the model transformation standard QVT [12]. However, since QVT is not based on a formal foundation and, therefore, also suffers from a lack of precision, we decided to base our transformation implementation on TGGs, which have formally and precisely defined semantics.

3 Usage Scenarios

Extensions to COTS tools, which we develop with MOFLON, typically perform analysis and repair tasks on single models, or they keep data across several tools consistent. A combination of both kinds of extensions is possible.

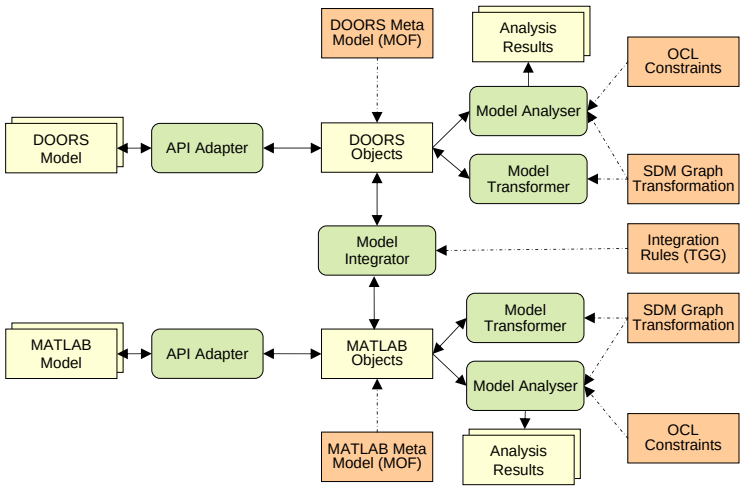

Fig. 1. Integration Scenario Including Model Analysis and Repair

Figure 1 provides an overview of such a combination. It shows the integration of the requirements engineering tool DOORS with the systems modeling environment MATLAB/Simulink. Adapters provide standardized interfaces to the data in each tool, i.e. the adapter provides JMI compliant objects to all other components. This is, for instance, required for the model transformation rules to work properly. For both the DOORS and the MATLAB data, there is a model analyzer and transformer, which take OCL constraints and model transformation rules as input and apply them to the models. Moreover, there is a model integrator, which applies TGG rules to keep data between the tools consistent.

3.1 Tool Adapters

The code generated by MOFLON for model transformations requires a JMI compliant metamodel to run. In order to perform analysis and repair actions on models in tools, we need a JMI compliant interface to this data. We use MOFLON to describe the API and data structure of the tool in a metamodel, and to generate the interfaces and a substantial part of the adapter implementation with a customized set of templates. As an example, Figure 14.2 shows the metamodel of the modeling and simulation tool MATLAB/Simulink.

Since adapters use calls to the proprietary tool API, a part of it needs to be written manually. An evaluation based on the MATLAB/Simulink adapter has shown that about 95% of the adapter (measured in lines of code) can be generated. This includes the interfaces and most of the implementation of the reflective methods, whereas calls to the tool API must be implemented manually. Further increment of this percentage will be possible, if some API calls like setting attribute values in model elements are generated with tool specific templates.

3.2 Model Analysis and Repair

With the JMI compliant tool adapter, one can perform model analyses and repairs, which are implemented by means of OCL constraints and model transformations. Minor repairs may be performed automatically, but more complex actions require a user to choose one of several possible repair actions. Analyses and repairs with MOFLON, especially on MATLAB/Simulink models are discussed in detail in chapter 14 of this book.

3.3 Integration Framework

Integration rules specified in the TGG editor can be translated to operational graph analysis and transformation rules by MOFLON. Figure 2 provides a more detailed view of the integration between DOORS and MATLAB/Simulink models.

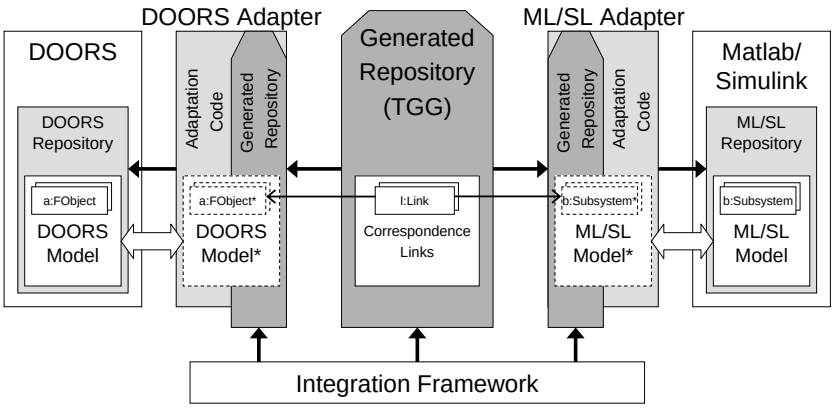

Fig. 2. Integration between DOORS and MATLAB/Simulink [13]

Access to the tool repositories is provided by the JMI adapters. The integration framework applies the TGG rules to the models. For instance, it may ensure that for every use case in DOORS, which is specified in a so called formal object (FObject in the figure), a corresponding subsystem must implement this use case in the MATLAB/Simulink model.

4 Conclusions and Future Work

The metamodeling tool MOFLON is designed for the rapid development of tool extensions rather than for developing tools from scratch. It includes editors and code generators for MOF compliant metamodels, OCL constraints, endogenous and exogenous transformations. Typical areas of application are model analysis and repair as well as model-to-model consistency checking and integration.

Future versions of MOFLON will provide enhanced possibilities to use commercial or open source tools for metamodel and transformation editing as well as more sophisticated modularization concepts for metamodeling in the large [14].

Acknowledgments

We would like to thank Tobias Rötschke, Alexander Königs and Carsten Amelunxen, who have initiated the MOFLON project and contributed a lot to it.

References

1. OMG, Inc.: Catalog of OMG Modeling and Metadata Specifications (Nov 2008) http://www.omg.org/technology/documents/modeling_spec_catalog.htm.
2. Dirckze, R.: Java Metadata Interface (JMI) Specification, v1.0 (June 2002)
3. Amelunxen, C., Königs, A., Rötschke, T., Schürr, A.: MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations. In Rensink, A., Warmer, J., eds.: Model Driven Architecture - Foundations and Applications: 2nd European Conference. Volume 4066 of LNCS., Springer Verlag (2006) 361–375
4. Bichler, L.: Tool Support for Generating Implementations of MOF-based Modeling Languages. In: Proceedings of The Third OOPSLA Workshop on Domain-Specific Modeling. (2003)
5. Zündorf, A.: Rigorous Object Oriented Software Development. University of Paderborn (2002) http://www.se.eecs.uni-kassel.de/fileadmin/se/publications/Zuen02.pdf.
6. Loecher, S., Ocke, S.: A Metamodel-Based OCL-Compiler for UML and MOF. Electr. Notes Theor. Comput. Sci. **102** (2004) 43–61
7. The Eclipse Foundation: Eclipse Modeling – EMF – Home (2008) http://www.eclipse.org/modeling/emf/.
8. netbeans.org: Metadata Repository (MDR) Project Home (2008) http://mdr.netbeans.org/.
9. Patzina, S.: Anpassung eines UML-Modellierungswerkzeuges für die Metamodellierung domänenspezifischer Sprachen. Master's thesis, TU Darmstadt (2008)
10. Amelunxen, C., Rötschke, T., Schürr, A.: Graph Transformations with MOF 2.0. In Giese, H., Zündorf, A., eds.: Proc. 3rd International Fujaba Days 2005. Volume tr-ri-05-259., Universität Paderborn (9 2005) 25–31
11. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In Tinhofer, G., ed.: WG'94 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science. Volume 903 of LNCS., Springer Verlag (1994) 151–163
12. Königs, A.: Model Integration and Transformation - A Triple Graph Grammar-based QVT Implementation. PhD thesis, Technische Universität Darmstadt (2009)
13. Amelunxen, C., Klar, F., Königs, A., Rötschke, T., Schürr, A.: Metamodel-based Tool Integration with MOFLON. In: 30th International Conference on Software Engineering, ACM Press (2008) 807–810 Formal Research Demonstration.
14. Weisemöller, I., Schürr, A.: Formal Definition of MOF 2.0 Metamodel Components and Composition. In Czarnecki, K., ed.: MoDELS 2008. Volume 5301 of Lecture Notes in Computer Science (LNCS)., Heidelberg, Springer Verlag (2008) 386–400