

Shortcomings in Metamodel Based Tool Development

Andy Schürr and Ingo Weisemöller

Technische Universität Darmstadt, Fachgebiet Echtzeitsysteme
Merckstraße 25, 64283 Darmstadt, Germany
{schuerr|weisemoeller}@es.tu-darmstadt.de
<http://www.es.tu-darmstadt.de/>

Abstract. It has become common practice to define the abstract syntax and the static semantics of a graphical modeling language in a metamodel. Based on this metamodel, changes on the models, i.e. the words of the modeling language, can be described as model transformations. Therefore, programming in the small has been partially replaced by metamodeling and modeling transformations in the development process of graphical modeling tools. In this position paper, we argue that modularization concepts need to be developed for metamodeling in the large, and that quality assurance of metamodel based tools needs to be improved by the introduction of appropriate testing procedures for model transformations. From our point of view, these improvements are the next steps to take towards a wholistic approach to model driven engineering, often referred to as *megamodeling* [1], and towards the systematic development of model transformations, which we call *model transformation engineering*.

1 Introduction

Though the activities of software development processes, such as Rational Unified Process or waterfall model, differ from each other in several points, there is an agreement that testing is an indispensable part of software development. Moreover, most process models also include a step to modularize the system under development, often referred to as design phase. If designed properly, these modules can be reused, either for several purposes in the same system, or in other systems where a similar module is required. In many cases, modules can also be parameterized in order to adopt them to a specific usage.

Since the system design is immediate input for the implementation phase, and since the implemented code is validated in the testing phase, the design and the testing phase are most closely related to programming in a software development process. When a general purpose language (GPL), such as C++ or Java, is used for programming, developers often also gain improvements in terms of modularization or testing from the language.

For instance, a system can be divided into several modules, which are separately developed as dynamically linked libraries and can be composed or exchanged independent of each other, given the fact their interfaces are compatible.

Moreover, some object oriented languages such as Ada have introduced generic classes. These classes can be parameterized at compile time, which results in a further improvement of reusability of modules.

Several approaches, such as coverage criteria for control flow based testing or the classification tree method for function oriented testing, exist for the systematic testing of programs in textual general purpose languages. Nowadays, testing is therefore probably the most important means for quality assurance in software engineering.

Development processes based on metamodels and model transformation lack comparable concepts for both modularization and testing. In section 2 we discuss reusability of solutions to frequently occurring problems in metamodeling: Although metamodeling languages such as MOF [5] introduce packages as a concept for modularization, these modules can hardly be reused properly for the definition of multiple modeling languages, because they do not provide well-defined and parameterizable interfaces, such as modules written in object oriented languages do. Section 3 deals with testing of model transformations. Most approaches for the systematic testing of general purpose languages cannot be applied to model transformations directly. Therefore, existing concepts need to be adopted and new ideas must be developed in this area. Section 4 discusses possible further improvements on metamodel driven development once the problems related to modularization and testing are solved.

2 Patterns and Components in Metamodeling

Our activities in the area of metamodeling during the last years (including research activities as well as industrial cooperations with, among others, SAP AG and Daimler AG) have shown that several aspects and problem solution are not specific to a single modeling language, but reoccur in multiple metamodels instead. For instance, take the definition of scopes, in which an identifier is bound to a named element. Such scopes and contained elements may occur in languages from a wide spread area of domains. Workflow process templates [4] and their activities contained in lanes or UML [3] models that use packages as namespaces are just two examples. Any languages dealing with scopes have to address some or all of the following problems: Names of elements, derivation of fully qualified names, visibility attributes for elements, visibility of elements in nested or nesting scopes, import of elements, import of entire scopes and transitive closure of nesting and/or imports.

There are several other reoccurring properties of modeling languages. Some share a set of modeling elements from their domain of application (such as business processes or software architectures), others have structural features in common (e.g., scopes and visibilities or type-instance-relations), and most languages follow some modeling paradigm, such as rule based or control flow based languages. A programmer who uses a GPL would never consider to develop all his programs from scratch, but rather make use of libraries in order to reuse existing components and design patterns in order to apply known solutions to

reoccurring problems. This will, in general, increase the speed of development and improve the quality of the software product.

Therefore, sophisticated modularization concepts and design patterns should also be introduced for metamodeling. We suggest to use parameterizable components to describe frequently reused modules such as namespaces above, and to instantiate patterns for the solution of smaller modeling problems such as the computation of transitive closures or to forbid cycles for certain sequences of links.

3 Testing of Model Transformations

Whereas a wide variety of model transformation languages and engines has been developed by a considerable number of research institutions, only few effort has been spent on software quality assurance in model driven development. In particular, methods of software testing – as the most important mean of quality assurance – need to be investigated thoroughly with respect to their applicability for MDD.

Not all testing methods known from software development with general purpose languages are suited for tests of model transformations. In functional testing for instance, it is common practice to compute equivalence classes for input parameters of a method to reduce the number of required test cases. This may be a promising approach for model transformations as well, but it needs to be investigated which models build an equivalence class.

Furthermore, coverage criteria for model transformations need to be developed. Coverage criteria applied to generated code do not seem to be suitable here, since code generators may produce unreachable code for certain transformations, which will result in a low value of code coverage no matter how many tests are performed.

4 Conclusions and Outlook

We have pointed out some important areas of interest for future works on model driven development. The topics addressed in the previous sections were related to conceptual issues concerning metamodeling, specification of model transformations and module testing in MDD.

In coherence with conceptual improvements, further tools for model driven development such as debuggers (for which first approaches exist [2]) need to be developed, which are on a par with state of the art tools for GPLs. Furthermore, the impact of using a model driven approach on other phases of the software development process, such as system design or integration testing, needs to be examined. If existing methods turn out to be inappropriate, even software development processes may need to be adopted to model driven development.

References

1. J. Favre. Foundations of Model (driven) (Reverse) Engineering - Episode I: Story of the Fidus Papyrus and the Solarus, 2004.
2. Leif Geiger and Albert Zündorf. eDOBS - Graphical Debugging for eclipse. In *3rd International Workshop on Graph-Based Tools (GraBaTs) ICGT Workshop*, Natal, Brasil, September 2006.
3. Object Management Group, Inc. Unified Modeling Language: Superstructure, Jul 2005. <http://doc.omg.org/formal/2005-07-04.pdf>.
4. Object Management Group, Inc. Business Process Modeling Notation Specification, Feb 2006. <http://www.omg.org/cgi-bin/apps/doc?dtc/06-02-01.pdf>.
5. Object Management Group, Inc. Meta Object Facility (MOF) Core Specification, Jan 2006. <http://doc.omg.org/formal/2006-01-01.pdf>.