Towards more efficient Software Engineering with formal MDA

Sudhir Agarwal and Max Völkel

Institute of Applied Informatics and Formal Description Methods (AIFB),
University of Karlsruhe (TH), Germany.
{agarwal,voelkel}@aifb.uni-karlsruhe.de

Introduction

In Model driven architecture (MDA) domain experts model their knowledge in a modeling language at a higher abstraction level than source code. The mapping from domain models to source code is often performed automatically, but programmers still have to implement many details manually. MDA brings domain experts and programmers closer together and makes the communication gap smaller. Often this is accomplished through visual modeling tools using UML. These and many other widely accepted advantages of MDA contribute to the ever growing popularity of MDA. However, there are still some shortcomings of MDA that may be resolved by incorporating formal descriptions.

Shortcomings of Current MDA Technologies

The transformation from a non-executable *platform independent model* (PIM) to an executable *platform specific model* (PSM) implies losing some modeling strengths. While modeling languages such as ER and UML treat relations as first class citizens, such knowledge is lost in PSM languages like Java.

Furthermore, as the *formal* semantics of the most widely used modeling languages UML has not been defined by the OMG (Object Management Group), the tool vendors had to choose a mapping from PIM to PSM on their own.

Modeling a large software system is a time consuming and difficult task. Since the role of models is often confined to just being the basis of communication among the domain experts, modelers often start doubting the added value of the models considering the time and effort needed to produce the models. However, due to the lack of formal semantics of the underlying modeling languages, it is not possible to *reason* about the model automatically.

Another major drawback of the current MDA technologies is the lack of support for dynamic aspects of a system. Though UML provides diagram types for modeling dynamics of a system, the support for generating executable code from them is often missing in existing MDA tools. However, recent efforts known as "executable UML" are trying to fill this gap [2].

Towards formal MDA

Formal model descriptions enable two aspects. First, automatic reasoning becomes possible. Second, the model can directly be used as a data model for an application.

Advantages of Automatic Reasoning Automatic reasoning can be very useful for finding out inconsistencies in the model automatically. E. g. automatic detection of equivalent concepts (classes) can prevent redundant modeling.

Similarly, often methods with the same or a similar functionality are programmed more than once. Automatic reasoning can be exploited to detect and remove duplicate methods at the early stage of software development, since they make software harder to manage.

Decomposition and composition are among the main techniques of problem solving and engineering. Similarly, composition of models is often needed. With formal semantics, these mappings could possibly be made automatically. Detecting inconsistencies and redundancy in the merged models is also helpful.

The time spent by software developers for reading documentation (e.g. Javadoc) to find the appropriate method, can be further reduced, if API methods are described formally in more detail than just the method signature. Once such formal descriptions of APIs are available, a programmer can define his goal and let a matchmaking engine find the appropriate methods automatically.

Advantages of Executable Models Formal semantics of ontologies makes the conceptual model executable at the same time, thus enabling different views (Ontology view, ER-view, Java view etc.) over the same KB [1]. This allows to select the best suited view to perform a specific task while keeping one consistent KB. For example, iterating over an array is easier in Java, whereas answering complex queries is better done inside an appropriate reasoner. Further, integration of different models is very easy, since Relationships remain first class as no transformation is necessary.

References

1. Andreas Eberhart. Automatic Generation of Java/SQL based Inference Engines from RDF Schemas and RuleML. In I. Horrocks and J. A. Hendler, editors, *Proceedings of the First International Semantic Web Conference: The Semantic Web (ISWC 2002)*, volume 2342 of *Lecture Notes in Computer Science (LNCS)*, pages 102–116, Sardinia, Italy, 2002. Springer.
2. Stephen J. Mellor and Marc Balcer. *Executable UML: A Foundation for Model-Driven Architectures.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. Foreword By-Ivar Jacoboson.