Chapter 5
Evaluation of MDE Tools from a Metamodeling Perspective

João de Sousa Saraiva
INESC-ID/Instituto Superior Técnico, Portugal

Alberto Rodrigues da Silva
INESC-ID/Instituto Superior Técnico, Portugal

ABSTRACT

Ever since the introduction of computers into society, researchers have been trying to raise the abstraction level at which we build software programs. We are currently adopting an abstraction level based on graphical models instead of source code: MDE. MDE is the driving force for some recent modeling languages and approaches, such as OMG’s UML or Domain-Specific Modeling. All these approaches are founded on metamodeling: defining languages that represent a problem-domain. A key factor for the success of any approach is appropriate tool support. However, only recently have tool creators started considering metamodeling as an important issue in their list of concerns. In this paper, we evaluate a small set of MDE tools from the perspective of the metamodeling activity, focusing on both architectural and practical aspects. Then, using the results of this evaluation, we discuss open research issues for MDE-based software development tools.

INTRODUCTION

Ever since the appearance of computers, researchers have been trying to raise the abstraction level at which software developers write computer programs. Looking at the history of programming languages, we have witnessed this fact, with languages evolving from raw machine code to machine-level languages, afterward to procedural programming languages, and finally to object-oriented languages, which allow developers to write software by mapping real-world concepts into modular segments of code (called objects). Still, object-oriented languages are too “computing-oriented” (Schmidt, 2006), abstracting over the solution domain (computing technologies) instead of the problem domain.
Currently, the abstraction level is being raised into the model-driven engineering (MDE) paradigm (Schmidt, 2006). In this abstraction level, models are considered first-class entities and become the backbone of the entire MDE-oriented software development process; other important artifacts, such as code and documentation, can be produced automatically from these models, relieving developers from issues such as underlying platform complexity or the inability of third-generation languages to express domain concepts.

MDE is not a new idea. Already in the 1980s and 1990s, computer-aided software engineering (CASE) tools were focused on supplying developers with methods and tools to express software systems using graphical general-purpose language representations. The developer would then be able to perform different tasks over those representations, such as correction analysis or transformations to and from code. However, these CASE tools failed due to issues such as (a) poor mapping of general-purpose languages onto the underlying platforms, which made generated code much harder to understand and maintain, (b) the inability to scale because the tools did not support concurrent engineering, and (c) code was still the first-class entity in the development process while models were seen as only being suited for documentation (Schmidt, 2006). Currently, there are better conditions for such modeling tools to appear. Software systems today are reaching such a high degree of complexity that third-generation languages simply are not sufficient anymore; another abstraction level over those languages is needed. This need, combined with the choices of IT development platforms currently available (Java, .NET, etc.), to which models can be somewhat easily mapped, is the motivation for the adoption of MDE. There are already a few MDE-related case studies available, such as Zhu et al. (2004) and Fong (2007), but since most MDE work is still in the research phase, there is still a lack of validation through a variety of real business case studies.

There are already multiple MDE initiatives, languages, and approaches, such as the unified modeling language (UML), the MetaObject Facility (MOF), the model-driven architecture (MDA), and domain-specific modeling (DSM) (Kelly & Tolvanen, 2008). There are also other derivative approaches, such as software factories (http://msdn2.microsoft.com/en-us/teamsystem/aa718951.aspx) that follow the MDE paradigm. Nevertheless, it is important to note that these initiatives are not a part of MDE; rather, MDE itself is a paradigm that is independent of language or technology, and is addressed by these initiatives.

All these approaches share the same basic concepts. A model is an interpretation of a certain problem domain, a fragment of the real world over which modeling and system development tasks are focused, according to a determined structure of concepts (Silva & Videira, 2005). This structure of concepts is provided by a metamodel, which is an attempt at describing the world around us for a particular purpose through the precise definition of the constructs and rules needed for creating models (Metamodel.com, n.d.). These basic concepts are the core of metamodeling, the activity of specifying a metamodel that will be used to create models, which is the foundation of MDE.

From the developer’s point of view, a key issue for acceptance of any approach is good tool support so that software programs can be created in an easy and efficient manner. There is a wide variety of modeling tools available today, covering most modeling standards and approaches in existence. For example, Rational Rose and Enterprise Architect (EA)(SparxSystems, n.d.) are only two examples of a very long list of tools that support UML modeling. DSM has recently become popular with the developer community, with tools such as Microsoft’s DSL Tools (MSDSLTools) or MetaCase’s MetaEdit+.