Category: Software & Systems Design

Foundations for MDA Case Tools

Liliana María Favre
Universidad Nacional del Centro de la Pcia. de Buenos Aires, Argentina

Claudia Teresa Pereira
Universidad Nacional del Centro de la Pcia. de Buenos Aires, Argentina

Liliana Inés Martinez
Universidad Nacional del Centro de la Pcia. de Buenos Aires, Argentina

INTRODUCTION

The model driven architecture (MDA) is an initiative proposed by the object management group (OMG), which is emerging as a technical framework to improve productivity, portability, interoperability, and maintenance (MDA, 2003).

MDA promotes the use of models and model-to-model transformations for developing software systems. All artifacts, such as requirement specifications, architecture descriptions, design descriptions, and code are regarded as models. MDA distinguishes four main kinds of models: computation independent model (CIM), platform independent model (PIM), platform specific models (PSM), and implementation specific model (ISM).

A CIM describes a system from the computation independent viewpoint that focuses on the environment of and the requirements for the system. In general, it is called domain model. A PIM is a model that contains no reference to the platforms that are used to realize it. A PSM describes a system with full knowledge of the final implementation platform. In this context, a platform is "a set of subsystems and technologies that provide a coherent set of functionality which any application supported by that platform can use without concern for the details of how the functionality is implemented" (MDA, 2003, p. 2-3). PIMs and PSMs are expressed using the unified modeling language (UML) combined with the object constraint language (OCL) (Favre, 2003; OCL, 2004; UML, 2004).

The idea behind MDA is to manage the evolution from CIMs to PIMs and PSMs that can be used to generate executable components and applications. In MDA is crucial to define, manage, and maintain traces and relationships between different models and automatically transform them and produce code that is complete and executable.

Metamodeling has become an essential technique in model-centric software development. The metamodeling framework for the UML itself is based on architecture with four layers: meta-metamodel, metamodel, model, and user objects. A metamodel is an explicit model of the constructs and rules needed to build specific models, its instances. A meta-metamodel defines a language to write metamodels. OCL can be used to attach consistency rules to models and metamodels. Related OMG standard metamodels and meta-metamodels such as meta object facility (MOF), software process engineering metamodel (SPEM) and common warehouse model (CWM) share a common design philosophy (CWM, 2001; MOF, 2005; SPEM, 2005).

MOF defines a common way for capturing all the diversity of modeling standards and interchange constructs. MOF uses an object modeling framework that is essentially a subset of the UML core. The four main modeling concepts are "classes, which model MOF metaobjects; associations, which model binary relationships between metaobjects; data types, which model other data; and packages, which modularize the models" (MOF, 2005, p. 2-6). The query, view, transformation (QVT) standard depends on MOF and OCL for specifying queries, views, and transformations. A query selects specific elements of a model, a view is a model derived from other model, and a model transformation is a specification of a mechanism to convert the elements of a model, into elements of another model, which can be instances of the same or different metamodels (QVT, 2003).

The success of MDA depends on the existence of CASE (computer-aided software engineering) tools that make a significant impact on software processes such as forward engineering and reverse engineering processes (CASE, 2006). This article explains the most important challenges to automate the processes that should be supported by MDA tools. We propose an integration of knowledge developed by the community of formal methods with MDA. We describe a rigorous framework that comprises the metamodeling notation NEREUS and bridges between MOF-metamodels and NEREUS, and between NEREUS and formal languages. NEREUS can be viewed as an intermediate notation open to many other formal specifications. We analyze metamodeling techniques for expressing model transformations such as refinements and refactorings. Our approach focuses on interoperability of formal languages in model driven development (MDD).

This article is organized as follow. We first analyze the limitations of the existing MDA-based CASE tools. Then, we describe the bases of rigorous MDA-based processes. Next, we show how the formalization of MOF metamodels and metamodel-based model transformations allows us automatic software generation. Finally, we highlight the key directions in which MDA is moving forward.

BACKGROUND

To date, there are about 120 UML CASE tools that vary widely in functionality, usability, performance, and platforms (CASE, 2006). Some of them can only help with the mechanics of drawing and exporting UML diagrams. The mainstream object-oriented CASE tools support forward engineering and reverse engineering processes and can help with the analysis of consistency between diagrams. Only a few UML tools include extension for real time modeling. The tool market around MDA tools is still in flux and only about 10% of them provide some support for MDA. Table 1 exemplifies a taxonomy of the UML CASE tools (CASE, 2006).

The current techniques available in the commercial tools do not allow generating complete and executable code and after generation, the code needs additions. A source of problems in the code generation processes is that, on the one hand, the UML models contain information that cannot be expressed in object-oriented languages while, on the other hand, the object-oriented languages express implementation characteristics that have no counterpart in the UML models.

Moreover, the existing CASE tools do not exploit all the information contained in the UML models. For instance, cardinality and constraints of associations and preconditions, postconditions, and class invariants in OCL are only translated as annotations. It is the designer's responsibility to make good use of this information either selecting an appropriate implementation from a limited repertoire or implementing the association by himself.

On the other hand, many CASE tools support reverse engineering, however, they only use more basic notational features with a direct code representation and produce very large diagrams. Reverse engineering processes are facilitated by inserting annotations in the generated code. These annotations are the link between the model elements and the language. As such, they should be kept intact and not be changed. It is the programmer's responsibility to know what he or she can modify and what he or she cannot modify.

UML CASE tools provide limited facilities for refactoring on source code through an explicit selection made for the designer. However, it will be worth thinking about refactoring at the design level. The advantage of refactoring at UML level is that the transformations do not have to be tied to the syntax of a programming language. This is relevant since UML is designed to serve as a basis for code generation with MDA (Sunye et al., 2001).

Techniques that currently exist in UML CASE tools provide little support for validating models in the design stages. Reasoning about models of systems is well supported by automated theorem provers and model checkers, however, these tools are not integrated into CASE tools environments. Another problem is that as soon as the requirements specifications are handed down, the system architecture begins to deviate from specifications (Kollmann & Gogolla, 2002). Only research tools provide support for formal specification and deductive verification.

All of the MDA CASE tools are partially compliant to MDA features. They provide good support for modeling and limited support for automated transformation. In general, they support MDD from the PIM level and use UML class diagrams for designing PIMs. Some of them provide only one level of transformation from PIM to code (Codagen, Ameos, Arcstyler) and, in general, there is no relation between QVT and the current existing MDA tools. As an example, OptimalJ from Compuware supports MDD from PIM level. It allows generating PSMs from a PIM and a partial code generation. It distinguishes three kinds of models: a domain model that correspond to a PIM model, an application model that includes PSMs linked to different platforms (Relational-PSM, EJB-PSM and Web-PSM), and an implementation model.

Table 1. UML CASE tools

Basic drawing tools	Visio
Main stream object oriented case tools	Rational Rose, Argo/UML, Together, UModel, MagicDraw, MetaEdit+, Poseidon
Real time/embedded tools	Rapsody, Rational Rose Real Time, RapidRMA
MDA-based tools	OptimalJ, AndroMDA, Ameos, Together Architect, Codagen, ArcStyler, MDE Studio, Objecteering