Forward Engineering of UML Static Models[1]

Liliana Favre
CIC, Argentina

Liliana Martinez
Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina

Claudia Pereira
Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina

INTRODUCTION

The Unified Modeling Language (UML) has emerged as a modeling language for specifying, visualizing, constructing, and documenting software-intensive systems. It unifies proven software modeling languages that incorporate the object-oriented community's consensus on core modeling concepts. It also includes additional expressiveness to handle problems that previous visual languages did not fully address (Rumbaugh, Jacobson & Booch, 1999).

UML emerged in response to a call for a standard object-oriented design method by the Object Management Group (OMG) in 1997. In mid-2001, OMG started working on a major upgrade to UML, and its evolution will result in version 2.0 (OMG, 2004).

The UML notation includes diagrams that provide multiple perspectives of the system under development. It defines 12 types of diagrams divided into three categories that represent static application structure, aspects of dynamic behavior, and ways for organizing and managing application modules.

The model's elements are defined in terms of their abstract syntax, well-formed rules, and precise text (OMG, 2004). These well-formed rules are expressed in the Object Constraint Language (OCL). Recently, a new version of OCL, version 2.0, has been formally defined and it has been adopted by OMG (Warmer & Kleppe, 2003).

UML is used in many ways and different domains for expressing different types of concepts such as language-independent software specification, high-level architecture, Web site structure, workflow specification, and business modeling. It has been applied successfully to build systems for different types of applications running on any type and combination of hardware, operating system, programming language, and network.

Although UML does not prescribe any particular development process, the OMG presented the Software Process Engineering Metamodel (SPEM) as a standard in November 2001 (OMG, 2004). This metamodel is used to describe a concrete software development process or a family of related software development processes that use the UML notation. SPEM has a four-layered architecture of modeling for describing performing process, process model, process metamodel, and MetaObject Facility (MOF). Several processes fit SPEM. The most popular is Rational Unified Process, developed and marketed by Rational Software (now a division of IBM). RUP is use-driven, architecture-centered, iterative, and risk-driven. Various industry sectors around the world use RUP in different applications: telecommunications, transportation, aerospace, defense, manufacturing, and financial services (Jacobson, Booch & Rumbaugh, 1999; Krutchen, 2000).

The international standardization of UML leads to improvements in CASE tools, methods, and standard modeling libraries. In the market, there are about 100 UML CASE (computer-aided software engineering) tools that vary widely in functionality, usability, performance, and platforms (CASE, 2004).

A recent OMG initiative is the Model-Driven Architecture (MDA), which promotes the creation of abstract models that are developed independently of a particular implementation technology and automatically transformed by tools into models for specific technologies (Kleppe, Warmer & Bast, 2003).

MDA is emerging as a technical framework to improve productivity, portability, interoperability, and maintenance. It defines how models expressed in one language can be transformed into models in other languages. The MDA process is divided into three main steps:

- Construct a model with a high level of abstraction that is called Platform-Independent Model (PIM).
- Transform the PIM into one or more Platform-Specific Models (PSMs), each one suited for different technologies.
- Transform the PSM to code.

The PIM, PSMs and code describe a system in different levels of abstraction. Using MDA, the business is modeled in Platform-Independent Models, which are transformed into Platform-Specific Models. This is carried out in an automatic manner.

The success of MDA depends on the definition of transformation languages and tools that make a significant impact on full forward engineering processes and partial round-trip engineering processes.

BACKGROUND

UML is having a significant impact on the software development industry. So far, there are about 100 UML CASE tools that vary widely in functionality, usability, performance, and platforms (CASE, 2004). Table 1 shows a taxonomy of the UML CASE tools.

The competing tools can be compared and contrasted by the following requirements: easy interface, modeling productivity, implementation productivity, and extensibility. The main stream object-oriented CASE tools can help with the mechanics of drawing and exporting UML diagrams, eliminating syntactic errors and consistency errors between diagrams, and supporting code generation and reverse engineering.

The current techniques available in the commercial tools are not sufficient for MDA-based forward engineering. A source of problems in the code generation process 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. For instance, languages like Java, C++, and Eiffel do not allow explicit associations. These can be simulated by pointers and references, but then the structure of the system is not apparent. This often leads to problems during forward engineering between the specification and code.

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.

UML CASE tools provide limited facilities for refactoring 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 the 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 the MDA paradigm (Sunyé, Pollet, Le Traon & Jézéquel, 2001).

Many UML 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.

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).

To solve these problems a lot of work has been carried out dealing with the semantics for UML models, advanced metamodeling techniques, and rigorous processes that fit MDA.

The Precise UML Group, pUML, was created in 1997 with the goal of giving precision to UML (Evans, France, Lano & Rumpe, 1998). It is difficult to compare the existing formalizations and to see how to integrate them in order to define a standard semantics since they specify different UML subsets and they are based on different formalisms (Ahrendt et al., 2002; McUmber & Cheng, 2001;

Table 1. UML CASE tools

Main Stream Object-Oriented CASE Tools	Rational Rose, Argo/UML, Poseidon, Together, GDPro, Stp/UML, MagicDraw
Real-Time/Embedded Tools	Tau UML Rhapsody Rational Rose Real Time
Basic Drawing Tools	Visio