Chapter XII

Model-Based Development: Metamodeling, Transformation and Verification

Juan de Lara
Universidad Autónoma de Madrid, Spain

Esther Guerra
Universidad Carlos III, Spain

Hans Vangheluwe
McGill University, Canada

Abstract

Since the beginning of computer science more than 50 years ago, software engineers have sought techniques resulting in higher levels of quality and productivity. Some of these efforts have concentrated in increasing the level of abstraction in programming languages (from assembler to structured languages to object-oriented languages). In the last few years, we have witnessed an increasing focus on development based on high-level, graphical models. They are used not only as a means to document
the analysis and design activities, but also as the actual “implementation” of the application, as well as for automatic analysis, code, and test case generation. The notations used to describe the models can be standard and general purpose (for example, UML) or tightly customized for the application domain. Code generation for the full application is only accomplished for specific, well-understood application domains. A key initiative in this direction is OMG’s Model-Driven Architecture (MDA), where models are progressively transformed until executable code is obtained. In this chapter, we give an overview of these technologies and propose ideas following this line (concerning metamodeling and the use of visual languages for the specification of model transformation, model simulation, analysis and code generation), and examine the impact of model-based techniques in the development process.

Introduction

Stakeholders in the development process have different interests. Managers want the product on time and within cost, users want more functionality and low prices, and developers want to reduce the effort in building the application. One of the means of reducing this effort is by increasing the level of abstraction of programming languages (that is, conceptually using a higher-level virtual machine). Using higher abstraction level notations, programs become more compact and easier to understand, write, and maintain. In this way, developers deal with less (accidental) details about the system they are building and concentrate on describing its essential properties (Brooks, 1995). Usually, powerful abstract constructs are only available in well-understood application domains, such as editors for visual languages (de Lara & Vangheluwe, 2002a). Ideally in these domains, from (possibly graphical) high-level descriptions of the application to be built, the program code is automatically generated. Other times, these models of the application are used for analysis of the program properties (such as efficiency, scalability, or design correctness). This is possible if the model semantics are formally defined (and have adequate analysis techniques) or if the model is translated into a formalism for which verification techniques are available (Guerra & de Lara, 2003; Heckel, Küster, & Taentzer, 2002).
Related Content

Design Patterns as Laws of Quality
[www.igi-global.com/chapter/design-patterns-laws-quality/27360?camid=4v1a](www.igi-global.com/chapter/design-patterns-laws-quality/27360?camid=4v1a)

What Are the Actual Industry Expectations and Needs with Regard to Object Oriented Technology?
[www.igi-global.com/chapter/actual-industry-expectations-needs-regard/27354?camid=4v1a](www.igi-global.com/chapter/actual-industry-expectations-needs-regard/27354?camid=4v1a)

Detailed Design of the Transactions and Class Methods
[www.igi-global.com/chapter/detailed-design-transactions-class-methods/18742?camid=4v1a](www.igi-global.com/chapter/detailed-design-transactions-class-methods/18742?camid=4v1a)

Information Systems Analysis with FOOM
[www.igi-global.com/chapter/information-systems-analysis-foom/18738?camid=4v1a](www.igi-global.com/chapter/information-systems-analysis-foom/18738?camid=4v1a)