Chapter II

A Systematic Approach to Transform UML Static Models to Object-Oriented Code

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

Silvia Clérici
Universidad Politécnica de Cataluña
España

ABSTRACT

Formal and semi-formal techniques can play complementary roles in object-oriented software development. The purpose of this chapter is to present a reuse-based rigorous method to forward engineering UML static models. This approach is based on the integration of semi-formal notations in UML with the algebraic style. The main contributions of this chapter are the definition of the GSBL\textsuperscript{oo} algebraic language to cope with concepts of the UML models, the definition of the SpReIm model for the description of the structure of reusable components and the definition of a rigorous process with reuse that assists in the transformation of UML class diagrams to object-oriented code. Eiffel was the language of choice in which to demonstrate the feasibility of our approach.

INTRODUCTION

The Unified Modeling Language, UML, is rapidly becoming a de-facto standard for modeling software systems (OMG, 1999). It is a set of graphical and textual notations for specifying, visualizing and documenting object-oriented systems. An important feature of UML is that it is a modeling language, not a method, and therefore it does not prescribe any particular development process. UML is not a visual programming language, but its models can be directly connected to a variety of object-oriented programming languages through forward engineering processes. These processes transform UML models into code through a mapping to an implementation language (Booch, Rumbaugh & Jacobson, 1999).
UML is too imprecise and ambiguous when it comes to simulation, verification, validation and forecasting of system properties and even when it comes to generating implementations from specifications. Another problem is that models written in UML are semantically richer than any other current object programming language. For example, an object-oriented language does not possess an explicit syntax to express UML 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/reverse engineering between the specification and code. A variety of advantages have been attributed to the use of formal software specification to solve these problems. It is commonly accepted that a formal specification can reveal gaps, ambiguities and inconsistencies. Furthermore, a precise semantics provides the basis for automated forward engineering and reverse engineering (the process of transforming code into a model through a mapping from a specific implementation language).

However, semi-formal approaches still dominate even in areas where formal approaches should provide a competitive advantage. The success of graphical modeling techniques is due to several reasons. On the one hand, they provide a set of graphical and textual descriptions that are easily understandable for both system developers and expert users. In general, there are also good tools to support them. On the other hand, formal methods provide minimal guidelines for eliciting and structuring requirements, and the notation of formal specification languages requires familiarity with symbolic logic that most designers and implementers do not currently have. Besides, the learning curve for the application of these new techniques requires considerable time.

Formal and informal techniques can play complementary roles in software development. The integration should be based on the following considerations:

- Formal models must preserve the interpretation of object-oriented models, constructions and underlying concepts reinforcing the informal interpretations.
- Specification and reasoning must be integrated with code.
- It is necessary to define mappings between formal models and object-oriented model constructs and concepts that provide the basis for assisted generation of formal specifications from informal specifications without forcing the user to change specification style.
- Only those parts that can benefit from more rigorous specification and analysis must be formalized.

In this chapter we present a rigorous process with reuse to forward engineer UML static models. This approach is based on the integration of semi-formal notations in UML with algebraic specifications. The primary objective of this integration is to make easier the analysis, reuse, evolution and maintenance of the software. Rather than requiring that developers manipulate formal specifications, we want to provide formal semantics for graphical modeling notations and develop rigorous tools that permit developers to directly manipulate models they have created.

The bases of our approach are:

- The definition of the GSBL $^\infty$ algebraic language.
- The definition of the $SpRelm$ model for the description of the structure of a reusable component.
- The definition of a method with reuse that assists in the transformation of UML class diagrams to object-oriented code.

The GSBL$^\infty$ language enriches GSBL (Clérici & Orejas, 1988) with constructs that allow us to express different kinds of UML relations: generalization/specialization, depen-
Related Content

A Secure and Dynamic Mobile Identity Wallet Authorization Architecture Based on a XMPP Messaging Infrastructure
www.irma-international.org/chapter/secure-dynamic-mobile-identity-wallet/73171/

Formalizing UML Class Diagrams
www.irma-international.org/chapter/formalizing-uml-class-diagrams/30541/

Negotiating Early Reuse of Components - A Model-Based Analysis
www.irma-international.org/chapter/negotiating-early-reuse-components-model/30538/

Sharing Ontologies and Rules Using Model Transformations
www.irma-international.org/chapter/sharing-ontologies-rules-using-model/35871/

A Generic Framework for Defining Domain-Specific Models
www.irma-international.org/chapter/generic-framework-defining-domain-specific/30535/