Chapter IX

Forward Engineering and UML: From UML Static Models to Eiffel Code

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

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

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

ABSTRACT

This chapter describes a reuse-based rigorous process to transform UML static models to object-oriented code. The bases of this approach are the GSBL algebraic language to cope with concepts of UML static models and the SpReIm model for defining structured collections of reusable components. We have defined a mapping between UML static models and GSBL. The emphasis in this chapter is given to the last steps in the road from UML to code. Eiffel™ is the language of choice in which we chose to demonstrate the feasibility of our approach. We analyze how to transform GSBL specifications into code. In particular, we show how to translate different kinds of UML associations to Eiffel. Also, we describe how to construct assertions from GSBL specifications. All of the proposed transformations can be automated; they allow traceability and can be integrated into the iterative and incremental software development processes supported by the existing UML CASE tools.
INTRODUCTION

Unified Modeling Language (UML) has emerged as a standard modeling language in the object-oriented analysis and design world. It is a set of graphical and textual notations for specifying, visualizing, and documenting object-oriented systems (OMG, 2001; Booch, Rumbaugh & Jacobson, 1999).

There exists a great number of UML case tools that facilitate code generation and reverse engineering of existing software systems. Unfortunately, techniques currently available in these tools provide little support for validating models in the design stages. Additionally, they are not sufficient for the complete automated code generation. Probably, this is mostly due to the lack of a precise semantics of UML and OCL (OMG, 2001). Another source of problems in these processes is that, on the one hand, UML models contain information that cannot be explicited 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 example, languages like C++, Java, and Eiffel do not allow us to explicit associations, their cardinality, and their OCL constraints. It is the designer’s responsibility to make good use of this information, by either selecting an appropriate implementation from a limited repertory or by personally implementing associations.

A variety of advantages has been attributed to using formal software specifications to solve these problems. A formal specification can reveal gaps, ambiguities, and inconsistencies. Any verification of UML models could take place on their corresponding specification using reasoning techniques provided by formalism. However, formal specifications alone do not address the need of industrial practitioners, who require an understandable and scalable semantics that can be integrated by using tools.

Favre and Clérici (2001) propose a rigorous process to forward engineering, UML static models using the algebraic language GSBL\textsuperscript{oo}. Our contribution was towards an embedding of the code generation within a rigorous process that facilitates reuse. The GSBL\textsuperscript{oo} language was designed to cope with concepts of the UML models. This language is relational-centric: it expresses different kinds of relations as primitives to develop specifications. The transformation of UML static models specified in OCL into GSBL\textsuperscript{oo} and a system of transformation rules have been described (Favre, 2001). The formal model SpReIm for defining structured collections of reusable components that integrate algebraic specifications and object-oriented code was defined. The manipulation of SpReIm components by means of reuse operators is the basis for the reusability.

The primary objective of this integration is to simplify the analysis, 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.

In this chapter, the emphasis is given to the last steps in the road from UML to code. Eiffel was chosen to prove the feasibility of our approach (Meyer, 1997). Our approach is based on the “Design by Contract” principle. Contracts imply obligations and benefits for clients and contractors, and are made explicit by the use of Eiffel assertions. These features facilitate the integration of axioms of specifications with object-oriented code. We describe how to transform GSBL\textsuperscript{oo} specifications into Eiffel and analyze the transformation of different kinds of UML relations. We also describe how to construct Eiffel assertions from GSBL\textsuperscript{oo} specifications.
Related Content

**Index Structures for XML Databases**
[www.irma-international.org/chapter/index-structures-xml-databases/41501/](www.irma-international.org/chapter/index-structures-xml-databases/41501)

**An Interactive Viewpoint on the Role of UML**
[www.irma-international.org/chapter/interactive-viewpoint-role-uml/30582/](www.irma-international.org/chapter/interactive-viewpoint-role-uml/30582)

**Formalizing and Analyzing UML Use Case Hierarchical Predicate Transition Nets**
[www.irma-international.org/chapter/formalizing-analyzing-uml-use-case/4935/](www.irma-international.org/chapter/formalizing-analyzing-uml-use-case/4935)

**Specification of Business Components Using Temporal OCL**
[www.irma-international.org/chapter/specification-business-components-using-temporal/30537/](www.irma-international.org/chapter/specification-business-components-using-temporal/30537)

**Making Programming Exercises Interoperable with PExIL**
[www.irma-international.org/chapter/making-programming-exercises-interoperable-pexil/73172/](www.irma-international.org/chapter/making-programming-exercises-interoperable-pexil/73172)