Chapter XII

Describing and Extending Classes with XMI: An Industrial Experience

Giacomo Cabri, Università di Modena e Reggio Emilia, Italy

Marco Iori, OTConsulting, Italy

Andrea Salvarani, OTConsulting, Italy

Abstract

This chapter reports on an industrial experience about the management and the evolution of classes in an automated way. Today’s software industries rely on software components that can be reused in different situations, in order to save time and reuse verified software. The object-oriented programming paradigm significantly supports component-oriented programming, by providing the class construct. Nevertheless, already-implemented components are often required to evolve toward new architectural paradigms. Our approach enables the description of classes via XML (eXtensible Markup Language) documents, and allows the evolution of such classes via automated tools, which manipulate the XML documents in an appropriate way. To grant standard descriptions compliant with the UML (Unified Modeling Language) model, we exploit the XMI (XML Metadata Interchange) interchange format, which is a standard, defined by OMG (Object Management Group), that puts together XML, UML and MOF (Meta Object Facility).
### Introduction

Today’s information systems are quite complex, and software developers are eager for appropriate methodologies and tools to support their work. Object-oriented technology has been a milestone in programming, but we are still far from an effective and complete adoption of it. With no doubt the UML language has proposed a valid help for developers, but they still have to face different issues.

One of the scarcest resources in today’s software development is, without a doubt, time. Customers want to have their products in a more and more short time, because they must accomplish the rhythms of today’s life, which are very quick and do not allow any delay. In addition, sometimes developers have to reinvent the wheel, because they cannot reuse existing solutions; this may happen for different reasons, from the inexperience of the developers to the inadequacy of the supporting languages and tools, from the chance of a wrong analysis to the incompatibility of previous solutions. This means a waste of time. So, software developers must face this situation, and find an appropriate help.

A methodology that is helpful is the one based on components. A component is a software entity that provides services via well-defined interfaces (Szyperski, 1998). The analogy with electronic components is clear. The aim of this methodology is to provide developers with reusable and tested building blocks that can be assembled to construct applications. Even if the idea is quite clear, the practical exploitation of components is not always easy. The two challenges we are going to face in this chapter are software storage (Mili et al., 1998) and software evolution (Casais, 1995; Yang & Ward, 2003).

The former concerns the storing of classes in a repository, to keep track of the completed work and, of course, to reuse developed solutions. Even if this is not a new problem, and solutions do exist (Vitharana, 2003; Arnold & Stepoway, 1987; Devanbu et al., 1991; Lillie, 1991; Meling, 2000), our aim is to propose a solution that takes into account also the evolution of the software in an integrated and interoperable way. Interoperability can be useful because it allows us to manage classes in an independent format and even to translate them into different programming languages.

The latter challenge relates to the evolution of existing classes in terms of their extension. For instance, let us consider a component that provides a given service, say S. If such a component is bound to a given architecture, it becomes very hard to reuse it. Instead, it would be better to develop a “generic” component that provides the service S, and that can be adapted to different architectures, for instance Enterprise Java Beans (EJB Specifications) and Remote Method Invocation (RMI Specifications). Therefore, our aim is to define an appropriate model of component that enables an automatic evolution of the components in terms of extensions to fit specific architectures. At the logical level, we would like to obtain the situation depicted in Figure 1: the same service can be exploited in different architectures without the need of recoding the component.

Of course this can be made by hand, but it becomes infeasible when a lot of classes are involved. Moreover, generally the extension does not require particular competencies, and can be fruitfully delegate to automatic tools.

OTConsulting (http://www.otconsulting.com) is a software industry that has been developing large-scale distributed object-oriented applications for several years. As a