A Pragmatic Approach for Building a User-Friendly and Flexible UML Model Repository

Mariano Belaunde
France Telecom CNET,
Technopole ANTICIPA, 2, avenue Pierre-Marzin
22307 Lannion, France
mariano.belaunde@cnet.francetelecom.fr

Abstract. In France Telecom research center in Lannion (France) we have been working for three years on OO modeling as a promising technology for unifying the representation of data. This has led us to develop a Model Repository Tool, which offers, as its default configuration, a full support for the UML 1.3 metamodel. The tool enables the manipulation of models by means of a Java or Python API. It provides a rich and flexible registration capability based on an explicit identification, relying on a two-leveled hierarchical naming space. The paper focuses on the design aspects of the repository tool and highlights its similarities and differences with the design principles of OMG Meta Object Facility specification.

1 Introduction

An Object Oriented Model Repository tool is basically a software that offers facilities to access and store data models that conform with a generic object oriented representation of data (metamodels).

This kind of tools is currently emerging because OO modeling standardization efforts have received a significant acceptance by the IT industry, allowing thus, the possibility for sharing very high-level information between people and between tools themselves.

In France Telecom research center in Lannion (France) we started in 1996 the development of a Model Repository Tool in order to provide support for a home-made metamodel, which we intended to use for our internal projects. As we had to be aware of managing metamodel evolutions, we have built very soon an architecture based on a meta-metamodel level, that allows generating automatically the implementation classes.

In 1998, we decided to implement the UML metamodel [UML]. At the same time, when we compared our work to the OMG MOF specification [MOF], we noticed that most of the principles were identical. Nevertheless, regarding other aspects, like the...
internal details of the interface definition, or the lack of support for *association classes*, we found that the MOF specification was not entirely satisfactory as to welcome a full alignment with it.

We have achieved two distinct implementations of the repository tool. The first has Python [PYTHON] as the OO target programming language, while the second has Java [JAVA]. Except for some subtle differences, due to the distinct capabilities of the two languages, the design principles are identical for both implementations.

In this paper we will present all along the pragmatic design principles that we have chosen in order to obtain a flexible and user-friendly Model Repository Tool. By the way, we think that this work can be used as a useful material for evaluating and improving the MOF specification.

1.1 UML Graphical Notation Support Versus Metamodel Support

In end users’ point of view the Unified Modeling Language is above all a graphical notation that allows people to communicate by means of very intuitive diagrams.

But for the software industry another important point is that the syntax and the semantics of this graphical notation has been, as better as possible, formalized in the form of an object oriented metamodel containing software abstractions such as *metaclasses*, *inheritance relationships*, *association links* and *constraints*.

This leads to the possibility of developing really open model repositories, meaning that they can offer access, storage and navigation capabilities which are strictly based on a non proprietary standard for model representation.

Most current commercial CASE tools are nowadays more concerned with supporting the graphical notation than explicitly supporting the UML metamodel. The situation is likely to evolve as soon as the OMG standard for model interchange [XMI] will be implemented by tool vendors.

2 Meta-metamodel Support

Our first concern was to develop a Model Repository based on a common and unique OO metamodel. As UML arises as a standard, we indeed adopted it. Even if, at first, we were interested in supporting a single metamodel, for practical reasons, we felt the need to have a meta language for describing similar metamodels, so that we could easily manage any metamodel evolution, and handle automatic production of the metamodel’s specific parts of the implementation.