Chapter III

Version Control of Software Models

Marcus Alanen, Åbo Akademi University, Finland
Ivan Porres, Åbo Akademi University, Finland

Abstract

We review the main concepts and algorithms behind a software repository with version control capabilities for UML and other MOF-based models. We discuss why text- and XML-based repositories cannot be used to manage models and present alternative solutions to build a model repository that takes into account specific details of MOF-based modeling languages.

Introduction and Motivation

In this chapter, we study how to store and manage large models during the lifetime of a software project. The first generation of UML editors used to store a whole model as a single file. This approach assumes that once a model is ready there will be no major changes and it can be distributed to the programmers as documentation. Programmers use the model as a reference design or blueprint for the code to be developed, but the model is no longer updated. In this scenario, software evolution and maintenance reverts over to the program source code, not to the UML model.

However, this approach is not satisfactory if we plan to use models instead of source code as the main and most important description of our software. This requires that any model should always be up-to-date. In this context, there will be different developers working
simultaneously on the same models. Different versions of the same model will be targeted to different platforms or customer requirements, and evolution and maintenance will be carried out over the models. This implies that we need to use a proper configuration management system to keep track of our models that comprise the final product.

Configuration management is a well-studied topic, and there are many tools available on the market. It involves several different subtopics such as version control as well as change, build, and release management. Configuration management is a key element in the management of any software development project. It is possible to construct a self-made system using a combination of open-source tools such as CVS, autoconf, make, and Tinderbox, or to use complete commercial solutions such as IBM Rational ClearCase.

However, most of the existing tools are designed to manage either program code or informal documents in natural language. The question now is if we can use existing configuration management systems to keep track of evolving models or if we need new tools and methods customized to the idiosyncrasies of the Object Management Group (OMG) standards. This research is centered on what we consider the central element of a configuration management system for models: a model repository with version control capabilities.

The objective of this chapter is to raise different issues that appear when we try to use inappropriate methods and tools to manage models while discussing possible alternatives that comply with the existing standards.

### Modeling Languages and the Meta Object Facility

According to the OMG standards, the information stored in a model is organized internally according to a metamodel. A metamodel describes the abstract syntax of a modeling language. Each class in a metamodel describes a concept or abstraction. Each class may have a number of attributes. An association connecting two classes represents a relation between these elements and is usually split into two opposing properties. A property has several characteristics, such as a name, multiplicity ranges, and directionality. Instances of the elements have connections to each other obeying the properties. The interconnections can be either ordered or unordered, and an element can occur either once or several times.

We can illustrate these concepts in a small modeling language of our invention that is much simpler than UML. Our example language is called FSM and is a language for describing finite state machines. A state machine has a finite number of states and transitions. Each transition connects two states and can be triggered by a token. The set of tokens in a state machine is called the alphabet. One or more states may be marked as final states, while one of the states is marked as initial. These concepts are described as a class model on the left side of Figure 1. We call this kind of diagram a metamodel. This diagram is similar to the metamodels shown in the OMG UML standards. In our example language, the fact that each state machine has an initial state is represented by the
Related Content

A Survey on JSON Mapping With XML/RDF
Gbéboumé Crédo Charles Adjallah-Kondo and Zongmin Ma (2019). Emerging Technologies and Applications in Data Processing and Management (pp. 92-113).
www.igi-global.com/chapter/a-survey-on-json-mapping-with-xmlrdf/230685?camid=4v1a

Extension of the Unified Modeling Language for Mobile Agents
www.igi-global.com/chapter/extension-unified-modeling-language-mobile/30575?camid=4v1a

Design Recovery of Web Applications Transactions
Scott Tilley, Damiano Distante and Shihong Huang (2005). Advances in UML and XML-Based Software Evolution (pp. 1-19).
www.igi-global.com/chapter/design-recovery-web-applications-transactions/4928?camid=4v1a

Mapping Policies to Web Rules: A Case of the KAoS Policy Language
www.igi-global.com/chapter/mapping-policies-web-rules/35875?camid=4v1a