Migration of Persistent Object Models Using XMI

Rainer Frömming, 4Soft GmbH, Germany
Andreas Rausch, Technische Universität Kaiserslautern, Germany

Abstract

Embrace the change! Change is a constant reality of software development, a reality that must be reflected in not only our software process but also our software production environment. With ever-changing customer requirements, modifications to the object model are required during software development as well as after product distribution. The associated migration of existing persistent object data is a nontrivial problem. This chapter presents the conceptualization and implementation of a tool for the automated migration of persistent object models. The migration is controlled by an XMI-based description of the difference between the old and the new object model. Both, the schema and the data of the persistent object model are migrated efficiently and reliably.

Introduction

“Time-to-market” is one of the major factors for success of software products today. Competitive pressure is forcing companies to introduce new versions of their software products with increased functionality to the market in ever-decreasing time intervals. For
instance, extreme programming (Beck, 2000) promises a development process to cope with this dramatically shortened life cycle.

However, agile software development with associated code-refactoring has its downsides as well. Short development cycles and fast assimilation to market requirements are possible, but this entails frequent enhancements and changes to the object model of the application under development. While changes to the object model and its corresponding program code are not necessarily crucial, due to modern development environments such as eclipse (Eclipse Foundation, n.d.), the migration of existing persistent object data in databases exposes the developer to supplementary, time-consuming tasks. Usually, there are two possibilities to address this problem: The first is to write additional program code besides the related product which handles the migration of data. The second is to include the logic how to access old and new data in the application itself. The drawback with the first possibility is that there is no framework which handles the basics of data migration to support the developer. This part can be very tricky, for example, without security features to backup your database or an architecture that can cope with class-naming-conflicts. When adapting the second possibility, the application is littered with code fragments only needed to differ between old and new data. This gets even worse when many new versions are built. Up to this point in time, no database vendor provides a suitable tool that addresses this migration problem in full complexity.

This results in the need for a tool which is capable in migrating the persistent object data from an aged software system to a new one in an automated, efficient, and reliable way. With the aid of such a tool, data migration expenses will be greatly reduced and the overall quality of the product will be improved considerably.

This chapter focuses on the concept and the functionality of such a tool, referred to here as ShapeShifter. The application domain of ShapeShifter is mainly concentrated on the object oriented software development cycle in conjunction with databases, where it improves data migration in terms of efficiency, traceability, and quality. ShapeShifter is written in Java, and in its present state, is able to migrate object-oriented databases from the Versant Corporation (n.d.). However, ShapeShifter is not basically limited to the Versant Database. Due to its flexible architecture and well designed interfaces, support for additional databases such as Oracle, DB2, or other relational databases is easily achievable and is already in preparation.

---

**XMI: Describing Object Models**

Any changes made to the object model entail knock-on changes to both the database schema and the persistent data. Although, small tools to support migration are delivered by the database vendors, these tools are insufficient and most of the migration work has to be done manually (Nierstrasz & Tsichritzis, 1995).

However, the migration process could be automated to a great extent, because most of the information you need is already at hand. The object models for both the new and old systems are available as source code and therefore also as a Unified Modeling Language (UML) description (Ambler, 2002; Fowler & Kendall, 2003).
Related Content

Issues in Personalized Access to Multi-Version XML Documents
www.irma-international.org/chapter/issues-personalized-access-multi-version/27783/

Using a Graph Transformation System to Improve the Quality of Characteristics of UML-RT Specifications
Lars Gunske (2005). Advances in UML and XML-Based Software Evolution (pp. 20-46).
www.irma-international.org/chapter/using-graph-transformation-system-improve/4929/

Index Structures for XML Databases
www.irma-international.org/chapter/index-structures-xml-databases/41501/

Formal Specifications of Software Model Evolution Using Contracts
Claudia Pons and Gabriel Baum (2005). Advances in UML and XML-Based Software Evolution (pp. 184-208).
www.irma-international.org/chapter/formal-specifications-software-model-evolution/4936/

ROL2: Towards a Real Deductive Object-Oriented Database Language
www.irma-international.org/chapter/rol2-towards-real-deductive-object/35859/