The Schema Evolution and Data Migration Framework of the Environmental Mass Database IMIS

Dirk Draheim
Institute of Computer Science
Freie Universtität Berlin
14195 Berlin, Germany
draheim@acm.org

Matthias Horn and Ina Schulz
IMIS Project
Condat AG
10559 Berlin, Germany
{matthias.horn,ina.schulz}@condat.de

Abstract

This paper describes a framework that supports the simultaneous evolution of object-oriented data models and relational schemas with respect to a tool-supported object-relational mapping. The proposed framework accounts for non-trivial data migration induced by type evolution from the outset. The support for data migration is offered on the level of transparent data access. The framework consists of the following integrated parts: an automatic model change detection mechanism, a generator for schema evolution code and a generator for data migration APIs. The framework has been concepted in the IMIS project. IMIS is an information system for environmental radioactivity measurements. Though the indicated domain especially demands a solution like the one discussed in this paper, the achievements are of general purpose for multi-tier system architectures with object-relational mapping.

1. Introduction

Information systems are data-centric applications. Due to changing requirements, both functional and non-functional, data types change during an information system's life time. That is we have to face with database reorganization [12, 11] and programming language type evolution, which must be in synch. Not only metadata is subject to change, but existing long-lived data, too. Altering schemas is supported by commercial database systems [13], however the definition of necessary data changes can pose problems. Necessary data changes can significantly vary in complexity. In the simplest case, i.e., if a schema evolution step can be described by a mere embedding of the old schema into the new schema, the change of data only amounts to a restructuring of the data along the known schema mapping. However, often more complex data changes are desired, ranging from a vertically or horizontally splitting of the data of one table into new tables to the problem of computing new column data from old column data.

We propose a schema evolution and data migration framework that is tightly integrated into a model-based, tool-supported development approach. The approach provides a model comparison mechanism that automatically reconstructs a schema mapping between a current model and an intended new model. Customizable upgrade programs for data migration are generated. The information of the model comparison phase is exploited in order to detect necessary customizations. Default data migration is provided in the approach that is maximal with respect to the model comparison results. The chosen implementation based on cloning is particularly simple and efficient. Data migration customizations are implemented on the level of transparent data access.

The described approach has been concepted in the IMIS project. The IMIS Integrated Measurement and Information System (Integriertes Meß- und Informationssystem für Radioaktivität in der Umwelt) is a German federal information system for forecast and decision support that gathers and interprets data on radioactivity in the environment. In the IMIS project the need for model evolution, especially non-trivial model evolution, is particularly high: IMIS stores measurements and sample data - measurement technology and methodology proceed steadily and the requirements of future desired queries are hardly predictable. At the same time nearly all the data stored in the IMIS system is long-lived and is therefore affected by model evolution. IMIS is installed at several federal and regional institutions at 60 locations and encompasses about 160 client systems. The system operates 24 hours on a central database and stores data about radioactivity in air, precipitation, inland waterways, north and Baltic seas, food, feed, drinking water, ground water, lake water, sewage, waste, plants and soil - measured manually and automatically by more than 2000 measurement stations. Data is supplied by auto-

Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM'04)
1099-3371/04 $ 20.00 © 2004 IEEE

mated processing, e.g. by import of data exchange files, as well as by manual data input. IMIS provides a broad range of features: manual and automatic data collection, configurable batch processing for data reception and transmission, a generic selection component that provides a decoupling of the technical data model from the terminology presented to the expert end user, data visualization using business diagrams and geographic maps, manual and automatic document generation, document storage and retrieval, and integration of the external forecast system PARK (program system for assessment and mitigation of radiological consequences). IMIS is estimated to store data about approximately one million measurements per year - this is equivalent to several million records. This leads to a forecast of approximately 50 GB measurement data after 10 years, an easily manageable amount of data at first sight – if certain data transforms become necessary due to changing requirements, e.g. for reasons of analytical processing, the actual needed database size has to be reestimated.

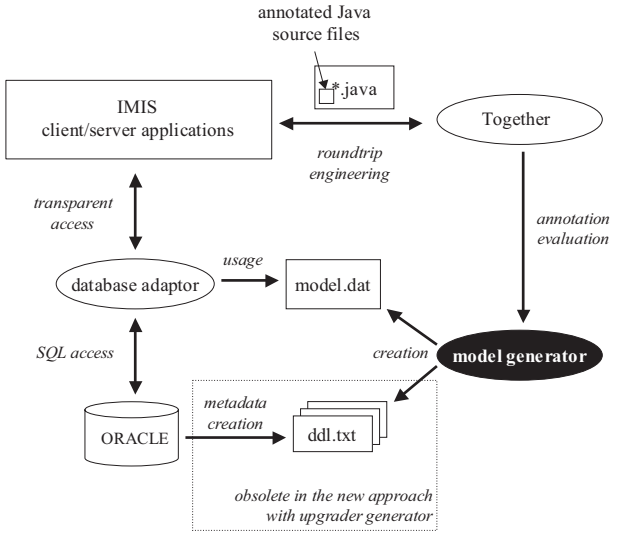

Figure 1. The IMIS model generator.

2. The IMIS Development Approach

All IMIS client and server applications – with exception of the document management system which is based on Zope and Python – are developed using an object-oriented approach using the Java programming language. The development follows a model-driven approach, which is depicted in Fig. 1. It is based on the usage of the case tool Together, a model generator, and a database adaptor software component. Together is used to maintain code and models by simultaneous round trip engineering. Both the database adaptor and the model generator has been developed in the IMIS project. Furthermore, new modules have been developed for the Together tool so that additional model information can be added to the UML models by annotations which are stored inside the Java source files. Similarly, the mapping from object-oriented model elements to the relational models can be specified by annotations of the UML model within the tool.

The model generator [2] makes the model information available to the database adaptor as serialized Java objects in the model.dat-file. The database adaptor realizes a transparent database access layer. It is a generic component that inspects the provided model.dat-file. It exploits the information to generate the SQL statements that are needed by the supported object-oriented access methods. The database generator provides advanced features. For example, the database adaptor is accompanied by an API for the formulation of arbitrary queries against the database. Prior to the final installation in October 2003 the database schema descriptions (ddl.txt-files) were also generated by the schema generator, as indicated by the shaded box in Fig. 1. The new evolution mechanism makes these descriptions obsolete, since it applies model changes incrementally - please take a first look at Fig. 2.

3. The Database Reorganization Process

The evolution of an object model results in changes of the database schema and the stored data. To employ a new software system version with an evolved object model, existing data needs to be transformed from the old database schema to a new one, called database reorganization in the scope of this paper. Therefore we have two tasks after changing an object model: the schema migration and the data migration. In general data migration needs to be defined with the semantic knowledge of the developer. Nonetheless a lot of data migration can be provided by default, i.e. can be generated. Our solution is based upon two parts: first we describe the actual process of database reorganization, split in some main steps, and second we introduce an application, the upgrader generator, which automatically compares two object models and generates the needed parts, i.e. SQL scripts and Java code, for the database reorganization.

The way database reorganization is made depends on many circumstances: on which database is used, on how many applications and on how many versions of the applications are supposed to use the data at the same time etc. The IMIS applications are supposed to run always exclusively on the database. There is only one version running at a time. To deploy an update, including database reorganization, IMIS is shut down for a while and the installation process has exclusive access to the database. Our solution focuses on an efficient and stable process. The process is sep-

arated into the following steps: database cloning, schema migration, and data migration.

We decided to clone the database, because a lot of the data can always supposed to be unchanged and cloning is the most efficient way to bring the data into the new schema.

The second step modifies the structure of the cloned database. This is done via generated SQL scripts. Before modification all existing constraints are disabled. This way the dependencies does not need to be analyzed and the modification steps can be performed in arbitrary order. The generated scripts drop, create and modify tables and columns until the schema fits the requirements of the new model. Similarly, constraints are subject to modification: obsolete constraints are deleted, new constraints are created but not yet enabled. Some data in the database clone is deleted during the modification of its schema.

The transformation and update of the changed objects is the third step. As mentioned earlier, this data migration needs to be performed with the knowledge of the developer about semantics. The relational representation of the objects is transparent to the developer. Therefore knowledge of the model change semantics is provided on object-oriented level in our approach. A generated Java program, called upgrader, performs the data migration. The upgrader program is presented to the developer as an API providing hooks for customizations. After the transformation of the cloned database with the upgrader program, the existing constraints are enabled.

4. The Upgrader Generator

The data reorganization process needs the SQL scripts for schema migration and the upgrader Java program for data migration. Both components have to be created each time a new model version appears. Their structure and behavior depends on the kind of model changes.

We developed a tool that compares the two models under consideration, analyzes the differences and generates the SQL scripts and the upgrader. This tool is called upgrader generator. As explained in Sect. 2, the object model is represented by a model.dat-file of serialized Java objects. The used meta model is separated into two parts: an object-oriented part that models packages and classes with attributes, associations, inheritance etc. and a relational part that allows for the specification of table and column names, attribute constraints like maximal string lengths and number sizes, primary key specifications etc.

The upgrader generator compares the two object models and finds structural differences such as new or removed classes, attributes and associations, new or removed sub- and super-classes, changes in the relational part of the model like changed table and column names, changed constraints etc. The developer can provide auxiliary information in special property files. For example, it is possible to rename a class, to rename an attribute, to move a class in the class hierarchy or to move an attribute from one class to another. As a result the structure of the new model is uniquely defined and the necessary SQL scripts can be generated automatically. Because the data migration in general depends on semantics provided by the developer, the generator ”guesses” a solution and generates Java code for the upgrader. The developer completes this implementation. The functionality of the upgrader generator is sketched in Fig. 2.

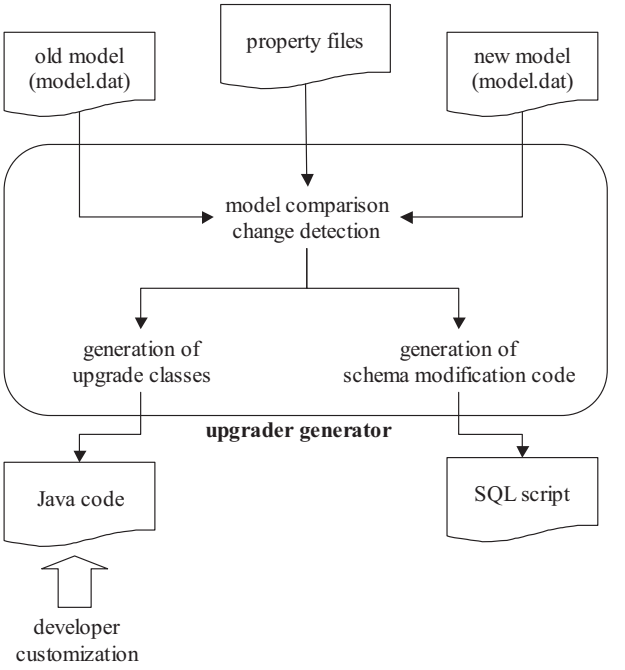

Figure 2. The upgrader generator.

The generator creates a special abstract upgrade class for each changed class in the model. This class serves as the basis for the transformation of its corresponding objects. There might be changes in the model for which the generator cannot guess a solution so that an implementation by the developer must be enforced. For example, if the developer has decreased the maximal string length of an attribute, she needs to implement the effect on too long values. In such a case the generator creates abstract methods that enforce an implementation by the developer. In other cases the guess made might not be correct. As an example, consider the splitting of an address attribute into several attributes of a new address class. The generator only finds a new address class and generates a new upgrade class, which does not create any new objects by default. The developer needs to overwrite the generated methods to create address objects that are computed properly from the former address attributes.

Actually the upgrader generator distinguishes between

two kinds of classes in the new model, i.e., classes that stem from the old model and entirely new classes. The detection of a class that stems from the old model is a good example for the simple way the upgrader constructs the schema mapping between the old and the new model: it is just based on name equality unless there isn't any explicit specification in the respective property file that redefines the origin of the class or redefines the class as entirely new. For an entirely new class the upgrader generator generates a hook for factoring objects. Then for each attribute of a given class it generates a hook that is called for every object of that class.

5. Related Work

An early rigorous analysis of the interdependency between model evolution and data migration with respect to a relationship between semantic data models and relational schemas can be found in [6].

Discussing schema evolution and data migration is particularly challenging with respect to object-oriented type systems, which are comparatively rich. The object-oriented database ORION [3] takes into account the physical level in the discussion of its data migration solution. ORION offers a solution with dynamic schema evolution, i.e. the administrator can trigger online database schema changes. ORION follows an adaptional approach: the model under consideration is converted and the database and applications are tailored with respect to the new model. The TSE [9] approach supports schema versioning by means of views: a base model is always only augmented and object deletions are emulated by views. The O2 [1] approach is a combined adaptional and schema versioning approach that targets the goal to minimize the need for application reconstruction.

The OTGen [5] generator produces a data migration program from a declarative description of a schema mapping, which is provided by the database administrator. The system Tess [4] picks up and further improves the contributions of the OTGen system. Tess takes a description of an old schema and a new schema and produces a data transform program. A schema mapping is automatically constructed for this purpose. An overview of automatic schema matching is provided by [10]. We want to mention the Clio [8, 7] system, which consists of a correspondence engine for schema matching and a mapping generator for producing view definitions.

6. Conclusion

In practice data migration must often still be done by hand coded SQL scripts, whereas, in a typical multi-tier setting, the developer must always be aware of all details of the object-relational mapping. This is the case even in the presence of commercial tools like TopLink. Our approach supports the type evolution in a multi-tier system that is based on an object-oriented transparent data access layer. The described framework consists of an automatic schema mapping mechanism and a generator for data migration APIs. Customization can be done solely on the level of transparent database access.

References

[1] F. Ferrandina and S.-E. Lautermann. An Integrated Approach to Schema Evolution for Object Databases. In *3rd International Conference on Object-Oriented Information Systems*, pages 280–294. Springer, December 1996.

[2] M. Horn, V. Triestram, and J. van Nouhuys. Data Evaluation Using the Generic Selection Component of the New IMIS System. In *EnviroInfo 2003 - 17th International Conference Informatics for Environmental Protection*. Metropolis, 2003.

[3] J. Banerjee, W. Kim et.al. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. *ACM SIGMOD Record*, 15(4), February 1987.

[4] B. S. Lerner. A Model for Compound Type Changes Encountered in Schema Evolution. *ACM Transactions on Database Systems*, 25(1):83–127, 2000.

[5] B. S. Lerner and A. N. Habermann. Beyond Schema Evolution to Database Reorganization. *SIGPLAN Notices*, 25(10):67–76, 1990.

[6] V. M. Markowitz and J. A. Makowsky. Incremental Reorganization of Relational Databases. In *13th International Conference on Very Large Data Bases*, pages 127–135. Morgan Kaufmann, 1987.

[7] R. J. Miller, L. M. Haas, and M. Hernandez. Schema Mapping as Query Discovery. In *Proceedings of the International Conference on Very Large Data Bases*, pages 77–88. Morgan Kaufmann, 2000.

[8] R. J. Miller, M. A. Hernández, L. M. Haas, L. Yan, C. T. H. Ho, R. Fagin, and L. Popa. The Clio Project: Managing Heterogeneity. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 30(1):78–83, 2001.

[9] Y.-G. Ra and E. A. Rundensteiner. A Transparent Object-Oriented Schema Change Approach Using View Evolution. In *11th IEEE International Conference on Data Engineering*. IEEE Press, 1995.

[10] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal: Very Large Data Bases*, 10:334–350, 2001.

[11] J. Roddick. A Survey of Schema Versioning Issues for Database Systems, 1995.

[12] G. H. Sockut and R. P. Goldberg. Database Reorganization - Principles and Practice. *ACM Computing Surveys*, 11(4):371–395, 1979.

[13] C. Türker. Schema Evolution in SQL-99 and Commercial (Object-)Relational DBMS. In *9th International Workshop on Foundations of Models and Languages for Data and Objects - Database Schema Evolution and Meta-Modeling*, volume 2065 of *LNCS*. Springer, 2000.