A Framework for Multi-Tier Type Evolution and Data Migration - Technical Report B-04-01 -

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

Matthias Horn
IMIS Project
Condat Informationssysteme AG
10559 Berlin, Germany
mch@condat.de

Ina Schulz
Institute of Computer Science
Freie Universtität Berlin
14195 Berlin, Germany
ischulz@inf.fu-berlin.de

January 2004

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. Thereby 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 the paper, the achievements are of general purpose for multi-tier system architectures with object-relational mapping.

Contents

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 [13, 12] and programming language type evolution, which must be in synch. Thereby not only metadata is subject to change, but existing long-lived data, too. Altering schemas is supported by commercial database systems [14], 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.

The framework that is described in this paper supports the type evolution in a multi-tier system that is based on an object-oriented transparent data access layer. The system development is centered around a tool-supported, model-based object-relational mapping, i.e. it follows a model-driven approach. In our setting model evolution, type evolution and schema evolution are tightly integrated. The described framework basically consists of a generator for data migration APIs. For each combination of a current model and an intended new model a specialized data migration API is generated. On the hand the generated data migration API is intended to be as complete as possible with respect to a schema mapping that can be automatically inferred from the two models under consideration, on the other hand it provides as many hooks as needed to fully customize the data migration. With this approach guidance for the implementation of the data migration is provided, furthermore the implementation can be done on the level of transparent database access. In practice - at least if complex data changes are necessary - 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 object-relational mapping tools like TopLink.

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 is 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.

The paper proceeds as follows. In section 2 we provide a brief overview of the functionality of the IMIS system. In section 3 we describe the model-driven development approach of the IMIS project. An understanding of the development approach is necessary for the explanation of the schema evolution and data migration framework. Section 4 sets the stage by describing the model evolution problem. The solution to the problem with respect to the special situation with object-relational mapping is described in section 5 in general, followed by a more detailed discussion of the migration API generator in section 6. We defer some of the discussion of the driving forces that lead to the several design decisions of the framework to section 7. Than related work is taken into account in section 8.

2 The IMIS System

Following the nuclear accident in Chernobyl the German federal government established a program targeting radiation protection and precaution in 1986. By the end of 1986 the respective federal law StrVG (Strahlenschutz Vorsorge Gesetz) was adopted. Besides other rules the StrVG contains guidelines for the installation of an information system for monitoring and prediction of radioactivity in the environment.

2.1 The IMIS Features

The first version of IMIS was developed between 1989 and 1993 and is currently still in use. In this paper we describe the entirely new IMIS system, which has been developed by Condat Informationssysteme AG in Berlin/Germany. The new IMIS has been installed in October 2003 for final continuous test operation.

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 automated 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, e.g. for data import and data export, document generation, data reception and transmission
- a generic selection component which provides a decoupling of the technical data model from the terminology presented to the expert end user
- data visualization using tables, business diagrams and geographic maps
- manual and automatic document generation
- document storage and retrieval
- integration of the external forecast system PARK (program system for assessment and mitigation of radiological consequences) for data supply, control and result import

From the end user's viewpoint the IMIS system has to be understood as a collection of rather loosely coupled client applications that together provide the aforementioned features.

2.2 The IMIS Data

The IMIS database consists of four schemas, i.e. IMIS, a repository schema, IMISGEO and PARK.

The schema IMIS consists of approximately 150 tables and basically contains the radioactivity data, master data with references to radioactivity data and data about samples. A sample is a portion of material that has been collected for radiological measurements. A sample is described by various attributes, e.g. the kind of collected material. The location from where a sample has been taken can be specified by coordinates or by an administrative district. Each sample is used for a number of measurements using various methods, e.g. alpha spectrometry. A measurement result consists of a number of readings, e.g. nuclides U-234 or U-235.

The repository schema consists of approximately 300 tables and contains configuration and setting data as well as dynamic data not related directly to radioactivity data. The configuration data is used to customize the various functions of IMIS, e.g. the selection and presentation component. For instance, stored messages or journals of automatic processes belong to the dynamic data stored in the repository schema.

The schema IMISGEO contains geographical data, e.g. maps for spatial evaluations. This schema is not covered by the evolution mechanism described in this paper. The schema PARK contains prognosis data computed by the external forecast system PARK. The PARK subsystem is only used in emergency mode. PARK prognosis data has a comparatively short lifetime, therefore data migration is not necessary for this schema. It can be emptied prior to schema evolution.

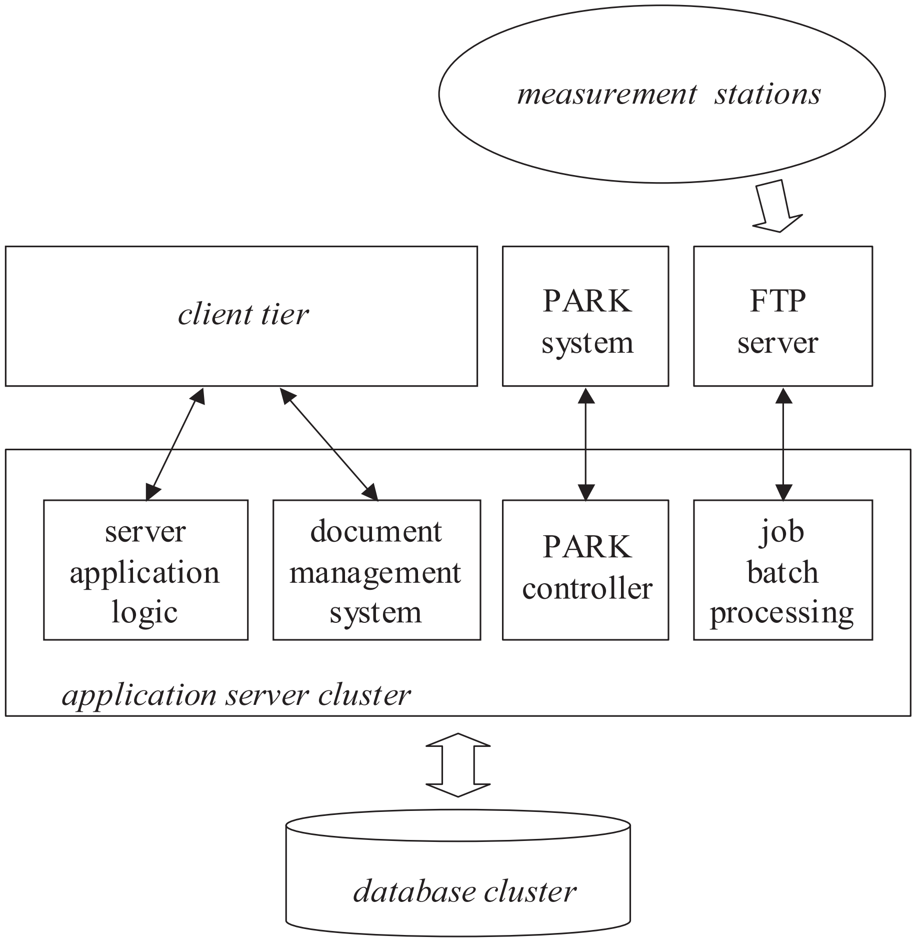

Figure 1: The IMIS Integrated Measurement and Information System.

2.3 System Architecture and Configuration

The system architecture of the IMIS system is depicted in Figure 1. A central Oracle9i database stores the data for evaluation and further processing. Configuration data for the different functions of IMIS is stored in the same database instance. It is running on a Sun V880 high availability cluster server consisting of two nodes. For data storage two Sun T3 storage subsystems are used. Server and communication processes are hosted on four Sun Fire 280 application servers. They are redundant and can replace each other in case of failure. All servers are located at the German federal office for radiation protection Bfs (Bundesamt für Strahlenschutz) in Munich. PCs are used as client systems. The client software follows a straightforward fat client approach. While most of the clients are connected via ISDN to the server LAN, the clients located on site in Munich are connected directly via Ethernet.

Most of the new data that is stored into the IMIS database stems from the measurement stations. These provide data by uploading it to an ftp server. From there the data is written by bulk data transfers, in normal operation mode on a daily basis and on a two hour basis in emergency operation mode. Further data is stored into the database by the external PARK system through the PARK controller. Further few data is entered manually by the user. Up to exceptions all the data stored in the IMIS system is long-lived, all the data stays unchanged. There is no heavy transaction load on the IMIS system. That is the IMIS system has the characteristics of an OLAP system, though currently no typical analytical processing takes places on the data. The client applications enable data browsing, they provide different views on the data. However, in future new complex queries may become requirements - yet another possible reason for schema evolution, though this time triggered by the need of physical database redesign with a footprint in the maintained model and applications.

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.

3 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 Figure 2. It is based on the usage of Coad's 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 an object access prediction is implemented that is exploited to prefetch objects from the database in order to mitigate performance drawbacks that are due to the navigational access patterns brought into play by transparent object-oriented access. The access prediction works cost-based by exploiting access statistics. As another example the database adaptor is accompanied by an API for the formulation of arbitrary queries against the database.

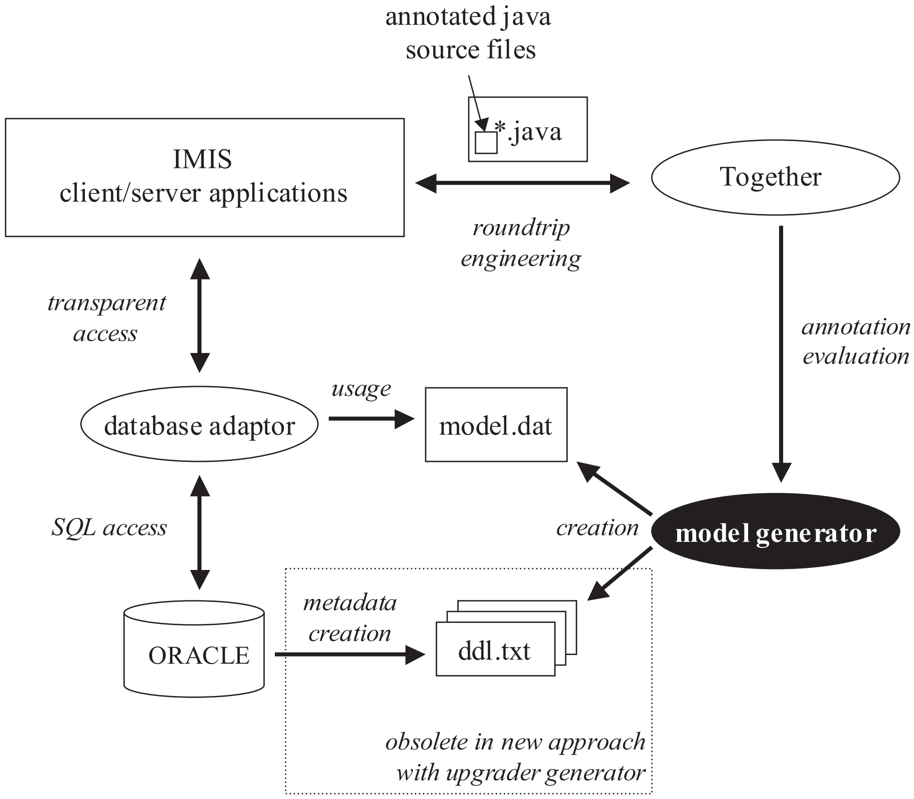

Figure 2: The IMIS model generator.

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 Figure 2. The new evolution mechanism makes these descriptions obsolete, since it applies model changes incrementally - please take a first look at Figure 5 on page 10.

4 The Model Evolution Problem

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.

We delve into an example that is particularly simple and does not stem from the IMIS application domain. Figure 3 shows the model evolution of a Company class with an address attribute and some further attributes. The modified model has a new Address class with a new street attribute, city attribute and zip attribute. The address attribute is removed from the Company class. Furthermore there exists an association between the Company class and the Address class. This way the schema migration is uniquely defined. However the data migration is more complicated and depends on the semantics of the changes. In the current example new objects of Address type have to be created and linked to the correct Company objects, whereas their attributes have to be computed properly from the old address attributes.

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. In our simple example the framework can assume that the remaining attributes of the Company class,

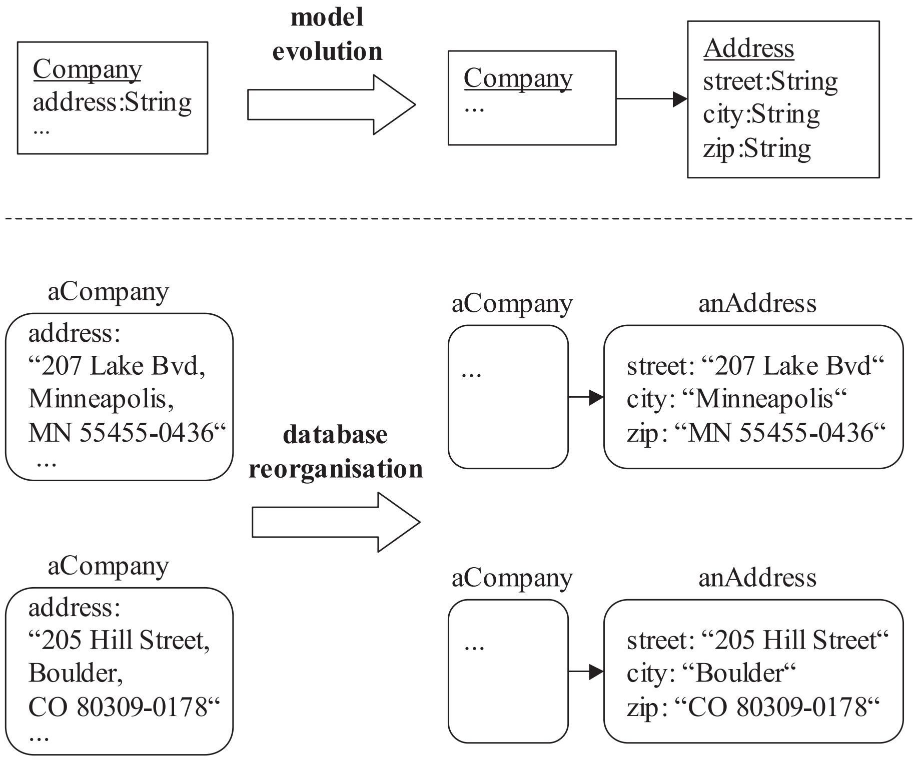

Figure 3: Non-trivial data migration.

i.e. the non-address attributes, are intended to have the same semantics in the new model as in the old model. Based on this assumption the data migration is conceptually just a copying for these attributes. Of course an elaborated approach has to provide a means to override the default behavior of such simple data migration parts, too.

This paper describes our solution for these problems. 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.

5 The Database Reorganization Process

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 and so on. The IMIS applications are supposed to run always exclusively on the database. There is only one version at a time running. 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 separated into the following steps:

- database cloning
- schema migration: modification of the cloned database schema
- data migration: transformation and update of the objects

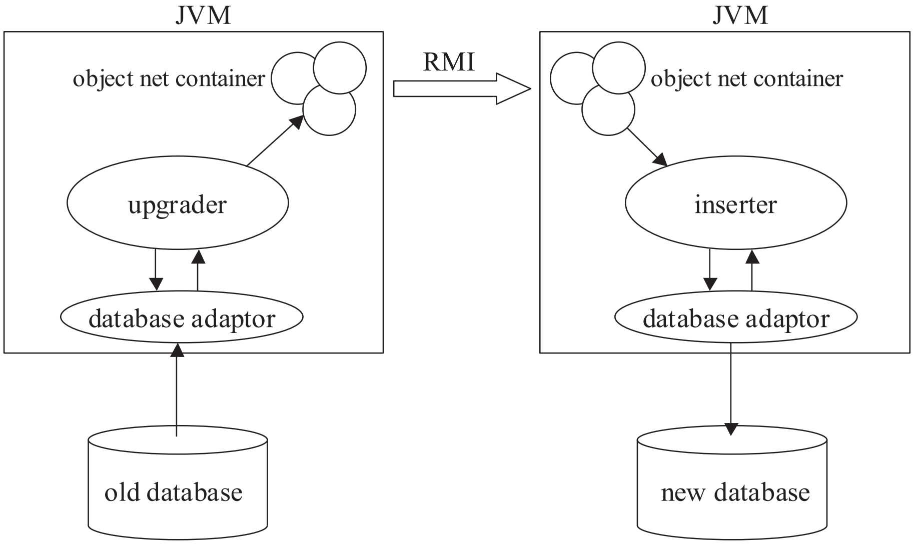

Figure 4: The upgrader concept.

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, too: 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 - in our preceding example the address attribute values.

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.

Figure 4 illustrates technical details of the data migration process. We use the old database as the object source. One database adaptor works on the old model and provides access to the old data objects. The custom Java data migration code is written with respect to the old database adaptor. The upgrader transforms the objects and sends them via RMI to an inserter process running in a separate virtual machine which inserts the new objects into the new, modified database. By the usage of two virtual machines two different name spaces are enforced, so that possible subtle name conflicts between the old and new application are prevented from the outset.

6 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 in quest, analyzes the differences and generates the SQL scripts and the upgrader. This tool is called upgrader generator.

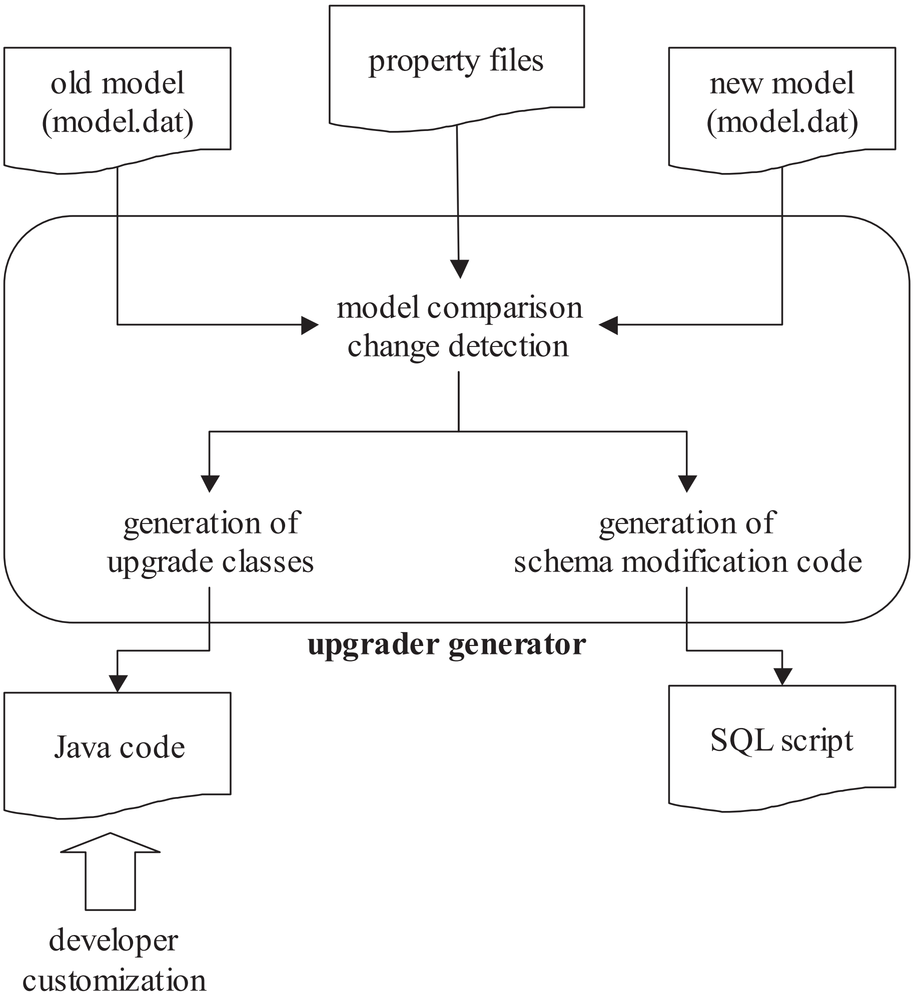

Figure 5: The upgrader generator.

As explained in section 3, 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 Figure 5.

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 provided by the developer. In other cases the guess made might not be correct. Please recall the example from the beginning - the split of the address attribute into several attributes of a new 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 the correctly filled objects and associate them with the corresponding company objects.

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.

The described architecture allows most complex reorganizations of the database. Any information of the old database may be accessed and auxiliary information sources may be included as well, like libraries, property files etc. The complete schema evolution framework consists of approximately 10 clocs of documented Java code.

7 Discussion

The first step in the database reorganization process is the schema migration and there are several ways to do it. One way would be the creation of a new and empty schema. This could easily be done by generating DDL statements from the object model and our model generator implements already this functionality. But creating an empty schema implies that a lot of unchanged data has to be moved from the old to the new schema. A more efficient way is to keep the data in the original schema. That means the schema has to be modified step by step until the structure fits the new model requirements by dropping, adding and modifying tables or columns etc. Only tables with relations to model changes would be touched. But modifying the existing data has its pitfalls. Some object transformation processes may need information of other objects, however these objects may be subject to change, too. With a copy it is not necessary to take dependencies into account, because every information is still accessible in the old schema. The most efficient and easiest way is to duplicate the database - tests have shown that it is at least twenty times faster as an SQL based solution.

Modifying the schema can be done by SQL statements. But modifying means also the loss of some data. We try to keep as much data in the schema as possible. For example if only a single attribute of a class has been changed, we modify only the corresponding column and only the values of one attribute has to be updated. In other cases, by example splitting a class into two, the old objects have to be removed and two new tables have to be created for the two new classes which are empty.

Changing the model doesn't lead only to data reorganization, but also to application migration. And this is also the task of the developer. The IMIS client applications are implemented in Java and the data transformation is done on object level by a Java program which the developer has to complete by implementing transformation rules for the objects.

8 Related Work

Improving schema evolution and data migration with respect to an object-relational mapping has subtle issues, because object-relational mapping is a practical challenge on its own. An early rigorous analysis of the interdependency between model evolution with respect to an underlying relationship between semantic data models and relational schemas can be found in [7].

Discussing schema evolution and data migration is particularly challenging with respect to an object-oriented type system because of the comparatively rich type construction facilities of a typical object-oriented type system. The object-oriented database ORION [3, 4] takes into account the physical level in the discussion of its data migration solution. ORION offers a solution with dynamic schema evolution. The administrator can trigger online database schema changes, i.e. without the need to restart the database. Thereby 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 [10] approach supports schema versioning by means of views. In this approach there exists a base model that is always only augmented. 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 [6] generator produces a data migration program from a declarative description of a schema mapping, which is provided by the database administrator. The system Tess [5] 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 [11]. We want to mention Clio [9, 8] as an exemplary system. The Clio system consists of a correspondence engine for schema matching and a mapping generator for producing view definitions that mitigate between source and target schemas.

9 Conclusion

- This paper tackles the multi-tiered problem of combined type evolution and data migration for software systems with an object-oriented application server tier and a relational database tier.
- The proposed schema evolution and data migration framework 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. Thereby 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.

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 et.al. Data Model Issues for Object-Oriented Applications. *ACM Transactions on Information Systems*, 5(1), January 1987.

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

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

[6] B. S. Lerner and A. N. Habermann. Beyond Schema Evolution to Database Reorganization. *SIGPLAN Notices*, 25(10):67–76, 1990.
[7] 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.
[8] 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.
[9] 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.
[10] 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.
[11] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal: Very Large Data Bases*, 10:334–350, 2001.
[12] J. Roddick. A Survey of Schema Versioning Issues for Database Systems, 1995.
[13] G. H. Sockut and R. P. Goldberg. Database Reorganization - Principles and Practice. *ACM Computing Surveys*, 11(4):371–395, 1979.
[14] 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.