Automating Program Conversion in Database Reengineering
A Wrapper-based Approach

Anthony Cleve
Laboratory of Database Applications Engineering
Computer Science Institute, University of Namur
21 rue Grandgagnage - 5000 Namur, Belgium
acl@info.fundp.ac.be

Abstract

Database reengineering consists in deriving a new database from a legacy database and adapting associated software components accordingly. This migration process typically involves three main steps, namely schema conversion, data conversion and program conversion. This paper presents a wrapper-based approach to automating the program conversion step. The proposed approach combines program transformations and code generation, which are derived from schema transformations.

1. Introduction

Most organizations encounter maintenance problems with their legacy software systems. These large programs are generally written in old programming languages like COBOL or PL/1. Typically, legacy systems are built around legacy databases (like IMS, CODASYL, etc.) or use primitive DMS1 (e.g., COBOL file system) [2].

Database reengineering consists in converting a legacy database according to new technical requirements, whereas the information content is left unchanged. Substituting a modern data management system (e.g., relational DBMS) for an obsolete manager (typically a standard file manager), or improving the logical schema to gain better performance are popular scenarios.

Migrating the application programs is another hard challenge. Indeed, the size and the complexity of the source code of the programs make it difficult to migrate the programs while maintaining the readability of the target code. The paper analyzes the problem of data-centered application programs migration following the migration of their databases. It presents the principles of a wrapper-based approach that couples program transformations and code generation and it describes a prototype tool architecture based upon this approach.

The paper is structured as follows. Section 2 presents the database reengineering issue itself. In Section 3, we develop the transformational techniques through which schema, data and programs are converted during the reengineering process. Section 4 focuses on the program conversion phase for which the use of wrapping techniques is suggested. Our prototype tool architecture is presented in Section 5. Finally, Section 6 gives some concluding remarks and plans to generalize our approach and tools.

2. Problem Statement

One of the biggest challenges in database migration is to provide new data structures that convey all the semantics of the legacy database. In particular, these structures should not inherit any technology-dependent feature from the legacy data structures. Failing to meet this requirement would produce a database that is flawed from its very start, leading to increasing semantic, integrity and performance problems.

The most popular migration approach, called one-to-one strategy in [7], ensures the structural equivalence between both legacy and new data structures. In a file to relational database migration, it consists in converting each record type into a table and each top-level field into a column. Its popularity comes from its extreme simplicity and its low cost: both databases have (as far as possible) the same schema and converting the programs is particularly easy, since each I/O statement is replaced with a functionally equivalent DML sequence. Such an approach naturally produces, but in some exceptional situations, poor data structures, thus hindering maintenance and evolution.

The other approach, called semantics-based strategy in [7], consists in converting the legacy database into new, normalized, data structures that ensure semantic equiva-

---

1DMS stands for Data Management System
3. Transformational Approach

Transformational engineering, that is, defining processes as chains of transformations, has proved to be both an elegant and efficient approach to perform these processes [5]. At the present time, we are provided with sound concepts and techniques to model most database engineering processes, and particularly database reengineering as co-transformations [8, 3]. These coupled transformations simultaneously convert three mutually-dependent software artifacts, namely the database schema, the data instances and the application programs while preserving the consistency that holds between them.

3.1. Schema Transformations

A schema transformation consists in deriving a target schema \( S' \) from a source schema \( S \) by replacing construct \( C \) (possibly empty) in \( S \) with a new construct \( C' \) (possibly empty) in \( S' \).

More formally, a transformation \( \Sigma \) is defined as a couple of mappings \( <T, t> \) so that: \( C' = T(C) \) and \( c' = t(c) \), where \( c \) is any instance of \( C \) and \( c' \) the corresponding instance of \( C' \). Structural mapping \( T \) explains how to modify the schema while instance mapping \( t \) states how to compute the instance set of \( C' \) from the instances of \( C \).

One of the most important properties of a schema transformation is the extent to which the source schema can be replaced with the target one without information loss. This property is called semantics preservation or reversibility [5].

The schema conversion phase of database reengineering can be seen as a chain of semantics-preserving schema transformations. Most of the time the source and target database schemas are semantically equivalent. They are said to have the same descriptive power since they describe the same universe of discourse, although with a different presentation.

3.2. Data Transformations

Schema conversion deals with the conversion of the data format and not of its content. Data conversion is handled by a software component often called Extract-Transform-Load processor, which transforms the data from the data source to the format defined by the target schema. A converter has three main functions. First, it performs the extraction of the data source. Second, it converts these data to make them comply with the target (new) format. Third, it writes legacy data in the target format. Such a converter relies on the mapping between the source and target database schemas.

3.3. Program Transformations

Program transformations form a sound basis for application programs conversion after the database migration. Legacy code alteration aims at reconciling the application programs with the migrated database. This reconciliation consists in rewriting the legacy DML2 primitives with two concerns in mind, namely (1) making them comply with the target DML and (2) adapting them to the target database schema.

The semantic equivalence that holds between the source and the target database schemas allows the logic of the legacy programs to be left unchanged during the program conversion process. The transformation mainly consists in locally replacing legacy DML primitives with an equivalent code fragment using target DML primitives. This transformation is based on the mapping that exists between the source and target DB schemas.

4. Wrapper-based Program Conversion

In migration and interoperability architectures, wrappers are popular components that convert legacy interfaces into modern ones. For instance, a set of standard files is given an object-oriented API suited to modern distributed architectures. Such wrappers allow to reuse legacy components [11, 13].

The wrappers discussed in this paper play the inverse role: they encapsulate the new database and provide access to the migrated data through a legacy API. This is the reason why this kind of wrapper is called inverse wrapper [6].

Our wrapper-based program conversion approach, depicted in Figure 1, is a two-step method:

1. An inverse wrapper, simulating the legacy DMS on top of the new database, is written. This wrapper converts all legacy DMS requests from legacy applications into requests against the new DMS that now manages the data. Conversely, it captures results from the new

---

2DML stands for Data Manipulation Language.
DMS, possibly converts them to the appropriate legacy format [10] and delivers them to the application program. In fact, we use one wrapper for each record type of the legacy database.

2. The legacy code is interfaced with the inverse wrappers. This step consists in replacing each DML primitive with a call to the corresponding wrapper. The program transformation is quite simple since, in most cases, the analysis of the DML statement is sufficient to derive the parameters of the wrapper invocation.

This approach allows (1) the design of a fully normalized database relieved of the flaws of the legacy data, (2) future programs to be developed on a sound basis and (3) legacy programs to work on the new database with minimal alteration, and therefore at low cost.

5. Tool Architecture

The architecture of our reengineering tool relies on two complementary transformational technologies, namely the DB-MAIN [12] CASE tool and the ASF+SDF Meta-Environment [1].

5.1. DB-MAIN

DB-MAIN is a data-oriented CASE environment. Its main purpose is to assist developers and analysts in the development, reengineering, migration and evolution of data-centered applications.

DB-MAIN offers general functions and components that allow to develop sophisticated processors supporting the renovation of data-centered applications. Among them:

- A generic model of schema representation based on the Generic Entity/Relationship (GER) model [4] to describe data structures in all abstraction levels and according to all popular modelling paradigms;
- A graphical interface to view the repository and apply operations;
- A library of transformations rich enough to handle most database engineering and reverse engineering processes.

5.2. The ASF+SDF Meta-Environment

The ASF+SDF Meta-Environment is an interactive development environment enabling automatic generation of interactive systems for manipulating programs, specifications or other texts written in a formal language. It is developed by the SEN1 research group of the CWI.

The ASF+SDF Meta-Environment provides tool generators to support the program conversion step. It allows to define the syntax of programming languages and to specify the transformation of programs written in such programming languages [14].

5.3. Schema Conversion Support

We use the transformation toolkit of DB-MAIN to carry out the schema transformations that are successively applied during the schema conversion phase.

5.4. Wrapper Generation Support

Writing wrappers manually is an expensive and complex task. Thanks to the DB-MAIN representation of the mapping between the source and target schemas, it is possible to automatically generate the inverse wrappers.

So far, we have implemented wrapper generators for COBOL-to-SQL and for IDS/II³-to-SQL. These wrapper generators have been developed using the Voyager 2 (V2) language. V2 is a complete, 4th-generation, semi-procedural language which provides predicative access to the repository of DB-MAIN and allows the user to develop customized functions which can be seamlessly incorporated in the tool.

5.5. Legacy Code Transformation Support

To automate the transformation of the legacy programs, we rely on the ASF+SDF Meta-Environment. We reused an SDF version of the IBM VS COBOL II grammar, which was obtained by Lammel and Verhoef [9]. We defined ASF rewrite rules on top of this grammar to obtain our program transformation tool. This generated tool takes as inputs the legacy program and some extra parameters to produce the renovated program.

³IDS/II is the BULL implementation of CODASYL.
Our program transformation tool is also suitable in the context of partial migration, i.e., when only some legacy record types are actually migrated to the new database platform. In this case, only the DML instructions used to access the migrated data are translated by a wrapper invocation and the other DML instructions, which still access the legacy data, are left unchanged.

6. Conclusions

Coupling generative and transformational techniques provides us with a very powerful CASE tool to tackle the problem of automatically generating the components of semantics-based system migration. Practically speaking, first experiments have shown the validity of the approach, at least for small to medium size programs. But validating it with large scale legacy systems still remains to be done. In particular, the performance impact of the wrappers should be measured in real-life conditions.

We anticipate several directions for future work on this topic. Among them:

- Extending our approach and tools to other legacy models, e.g., IMS and SQL. Indeed, SQL-to-SQL and IMS-to-SQL migrations are economically significant but require specific rules.

- Generalizing our database reengineering approach to face other system evolution contexts. The rules used to propagate schema transformations should be separated from the rules translating legacy primitives into target DML primitives. The first kind of rules could be reused for other evolution scenarios than database reengineering.

- Propagating, even semi-automatically, schema transformations that are not semantics-preserving is another challenge for the next future. This would require a deep understanding of the legacy code through complex program analysis techniques.

Acknowledgments This research was funded by the Région Wallonne and the European Social Fund, in the context of the FIRST Europe Objectif 1 research project 415768 entitled “RISTART: Aide à la migration des grands systèmes d’information par conversion automatique des programmes orientés-données”. This project involves an active collaboration with REVER s.a. and the SEN1 group of the CWI.

The author thanks Jean-Luc Hainaut and Jean-Roch Meurisse for helpful comments on the draft of this paper.

References


