Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
Electronic Notes in Theoretical Computer Science 137 (2005) 5–15
www.elsevier.com/locate/entcs

Co-transformations in Information System Reengineering

Anthony Cleve[1], Jean Henrard[2] and Jean-Luc Hainaut[3]

Database Applications Laboratory
University of Namur
21 rue Grandgagnage, Namur, Belgium

Abstract

Database reengineering consists of deriving a new database from a legacy database and adapting the software components accordingly. This migration process involves three main steps, namely schema conversion, data conversion and program conversion. This paper explores the feasibility of transforming the application programs through code transformation patterns that are automatically derived from the database transformations. It presents the principles of a new transformational approach coupling database and program transformations and it describes a prototype CASE tool based on this approach.

Keywords: database reengineering, schema transformations, program transformations

1 Introduction

Database reengineering consists of transforming a legacy database according to new technical requirements, while keeping the information contents unchanged. Substituting a modern data management system (relational DBMS for instance) for an outdated manager (typically standard file manager), or improving the logical schema to gain better performance are popular scenarios. Transformational engineering that is, defining processes as chains of transformations, has proved to be both an elegant and efficient approach to

[1] Email:acl@info.fundp.ac.be
[2] Email:jhe@info.fundp.ac.be
[3] Email:jlh@info.fundp.ac.be

1571-0661 © 2005 Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2005.07.001

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 migration as semantics-preserving transformations.

Migrating the application programs is another hard challenge. Indeed, the size and the complexity of the source code of the programs make the latter difficult to migrate 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 explores the feasibility of transforming the application programs through code transformation patterns that are automatically derived from the database transformations. It presents the principles of a new transformational approach that couples database and programs migration and it describes a prototype CASE tool based in this approach.

The paper is structured as follows. In Section 2, we define the problem of database reengineering, for which we present a disciplined methodology. Section 3 describes a transformational approach for each step of the reengineering process. In Section 4, we define the concept of co-transformation in the particular context of database applications reengineering. Section 5 describes the architecture of a tool developed to support the reengineering process. Finally, concluding remarks are given in Section 6.

2 Problem Statement

2.1 Information System Reengineering

One way to view Information System Reengineering is the reengineering of the database, i.e., deriving a new database from a legacy database, then adapting the software components accordingly. This description encompasses several objectives and strategies. In this paper, we will consider one of them, namely converting a legacy system to a modern database technology, that is, reengineering due to a technology change. Typically, this process comprises the following three main steps:

(i) *Database schema conversion*: the legacy database schema is translated into an equivalent schema expressed in the target technology.

(ii) *Data conversion*: the database contents are migrated from the legacy database to the new one. This step consists of a schema-driven ETL [4] process.

(iii) *Program conversion*: the legacy programs are modified so that they access

[4] ETL stands for *Extract-Transform-Load*

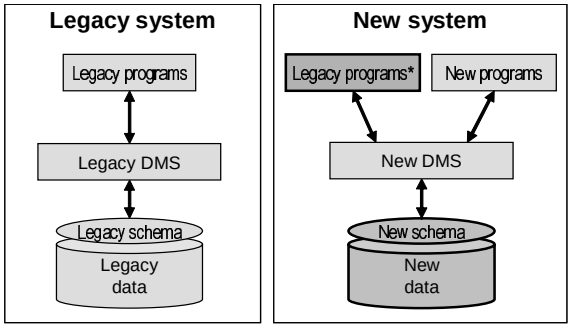

Fig. 1. System Conversion.

the new database instead of the legacy data. In the scenario studied, the functionalities, the programming language and the user interface are kept unchanged. This conversion step is generally a complex process that must rely on the schema conversion of Step 1.

The standard migration strategy we base the discussion on is sketched in Figure 1. The left part shows the main parts of the legacy system, comprising programs that interact with the legacy data through the API of the legacy DMS [5] and through the legacy schema. The right part shows the state of the new system after the legacy DMS has been replaced with a modern DMS (New DMS). The new database comprises the converted schema and the data that have been transformed and migrated according to the new schema. Legacy programs have been transformed in such a way that they now access the data through the API of the new technology and through the new schema. When the converted system is deployed, new programs can be developed, that use the database through the native interface of the new DMS. Later on, if and when needed, the legacy programs could be rewritten according to the new technology.

2.2 Physical VS Semantic Migration Approaches

The complexity of the process depends of its goal. If the organization mainly is interested in the lowest possible cost, then a *physical*, or *one-to-one* approach is sufficient. According to this technique, the translation of the legacy database consists of implementing it in the new technology in a straightforward way. With this approach, the conversion of COBOL files, for example, merely reduces to converting each record type into a table and each top level field into a column. Compound and repeating fields are ignored and converted into large aggregated columns. For network and hierarchical databases, the

[5] DMS stands for Data Management System.

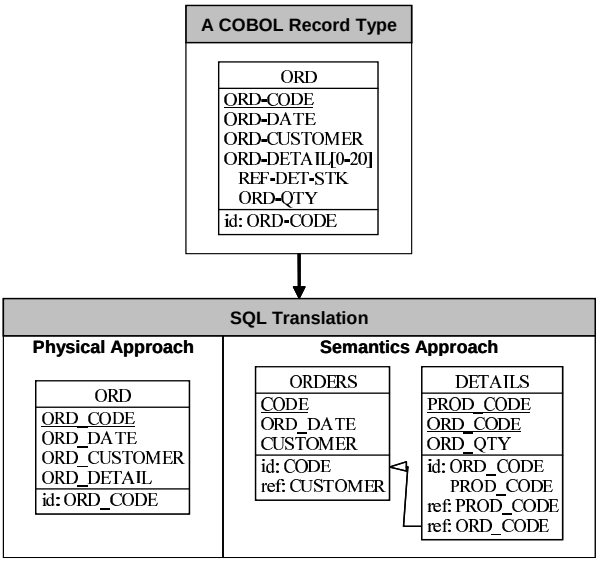

Fig. 2. Physical VS Semantic Migration Approaches.

result is even worse, since the navigation pointers are translated into columns, primary keys and foreign keys, making explicit the technical constructs that were hidden in the source schema.

The resulting database is quite obscure, very difficult to evolve and often very inefficient. However program conversion is quite easy and generally automated. As expected, this process is inexpensive, and therefore quite popular. On the long term, however, this technique oftent is a dead end. Indeed, both the database and the application programs are practically impossible to maintain, due to the poor data structures and to the intricacy of the source code.

The *semantic* migration approach produces a better quality target database. Through a complex DBRE[6] process, the conceptual schema (i.e., the semantics) of the legacy database is recovered. Then, the target database is designed from this conceptual schema, using standard database development techniques. Obviously, the semantic approach is more expensive, but it produces a well-designed, fully-documented database that forms a sound basis for both existing and future applications.

Figure 2 illustrates the differences between the physical and the semantic migration approaches, when migrating a COBOL record type (`ORD`) to a relational database. The physical approach (left) just *translates* the source schema into a SQL-compliant target schema. The `ORD` COBOL record type becomes the `ORD` SQL table. The compound and repeating `ORD-DETAIL` field becomes

[6] DBRE stands for DataBase Reverse Engineering

an obscure aggregated column of the `ORD` table. The semantic migration approach (right) *refines* and *interprets* the source schema before converting it into SQL, which improves the expressiveness of the target schema. In our example, the complex `ORD-DETAIL` field is translated into a distinct SQL table (`DETAILS`).

2.3 CASE Tools

The IS migration approach we develop in this paper relies on two complementary transformational technologies, namely DB-MAIN and ASF+SDF.

DB-MAIN

DB-MAIN [19] is a data-oriented CASE environment developed by the LIBD of the University of Namur. Its objective is to support most database engineering processes. It helps developers and analysts in the development, reengineering, migration and evolution of data-centered applications. DB-MAIN offers general functions and components that allow the development of sophisticated processors supporting data-centered application renovation.

Experience shows that there is no such thing as two similar reengineering projects. Hence the need for programmable, extensible and customizable tools. DB-MAIN (and more specifically its meta functions) includes features to extend its repository and its functions. In particular, it includes a 4GL (*Voyager 2*) that allows analysts to develop their own customized processors for analyzing and transforming data structures.

The ASF+SDF Meta-Environment

The ASF+SDF Meta-Environment [15] is an interactive development environment for the 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. In the context of data reengineering, the ASF+SDF Meta-Environment provides tool generators to support the program conversion step. It allows both defining the syntax of programming languages and specifying transformations of programs written in such programming languages [14].

3 Transformational Approach

In this section, we develop the transformation techniques through which the schema, the data and the programs are converted in the reengineering process.

3.1 Schema Transformation

A schema transformation is an operator, or mapping, T that states how to replace a source construct C (possibly empty) in schema S with construct C' (possibly empty), leading to schema S'. C' is the target of source construct C through T, that is, $C' = T(C)$. Disaggregating a compound attribute of an entity type, replacing a relationship type with an equivalent entity type or with a foreign key are three popular examples of schema transformations.

Completely defining a transformation requires specifying both inter-schema (T) and inter-instance (t) relations, otherwise, the operator is meaningless, or at least undefined. Therefore, a schema transformation is defined as a couple of mappings $<T, t>$ such as: $C' = T(C)$ 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.

A compound transformation $T = T_2 \circ T_1$ is obtained by applying T_2 on the schema that results from the application of T_1 [3]. A transformation $<T, t>$ is said to be semantics-preserving if one can associate with it another transformation $<T', t'>$, called its inverse, such that, for any C and c, instance of C,

(i) both $T' \circ T$ and $T \circ T'$ are the identity mapping between schemas,

(ii) both $t' \circ t$ and $t \circ t'$ are the identity mapping between data.

An important conclusion of the transformation-based analysis of database engineering processes is that most of them, including reverse engineering and database design, can be modelled through compound, semantics-preserving transformations. Database conversion, or reengineering [1], is carried out in two steps, namely database reverse engineering and database design. Therefore, the migration of a database from a technology to another one can be modelled by a complex compound transformation, the structural mapping of which is used to convert the database schema while the instance mapping defines the data conversion rules.

3.2 Data Transformation

A data transformation can be defined by the instance mapping t of a schema transformation. In the database reengineering context, data transformations are used to convert the source data in such a way that their structures match the target format without information loss.

3.3 Program Transformation

Program transformation is a modification or a sequence of modifications to a program. Just like a schema, a program is a structured object with semantics. The structure allows us to transform a program while the semantics gives us the means to compare programs and to reason about the validity of transformations [18]. Converting a program generally involves basic transformation steps that can be specified by means of *rewrite rules*. A rewrite rule recognizes a subterm to be transformed by pattern matching and replaces it with a pattern instance. Term rewriting is the exhaustive application of a set of rewrite rules to an input term (e.g. a program) until no rule can be applied anywhere in the term.

Program transformations form a sound basis for application programs conversion following the database migration. Indeed, the legacy I/O statements have to be rewritten with two concerns in mind, namely making the program comply with the new DMS API, and, more important, adapting the program logic to the new schema. We have explored two code organizations for transforming programs. The first one consists of replacing each legacy I/O statement with an equivalent section on the new schema. In the second organization, the replacement sections are collected in a single module that acts as a wrapper for the new database. In this case, the legacy I/O statements are replaced with wrapper invocations.

4 Co-Transformations

The general concept of *co-transformation* (or *coupled transformation*) is defined in [10] and [11].

> *"A co-transformation transforms mutually dependent software artifacts of different kinds simultaneously, while the transformation is centred around a grammar (or schema, API, or a similar structure) that is shared among the artifacts"*[11].

> *"(...) two or more artifacts of potentially different types are involved, while transformation at one end necessitates reconciling transformations at other ends such that global consistency is reestablished"*[10].

Data reengineering can be seen as a couple of correlated transformations comprising compound schema transformations (Schema and Data Conversion) and compound program transformations (Program conversion). These transformation categories must be consistent with each other, and developed in parallel, hence the name of *co-transformation*. In order to guarantee this consistency, we automatically derive the program transformations from the schema trans-

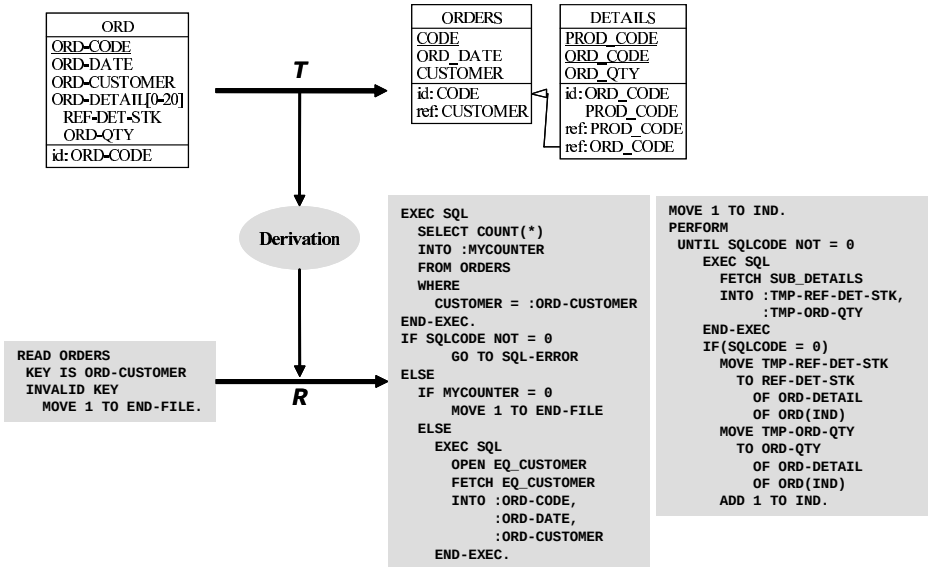

Fig. 3. Example of co-transformations.

formations.

Figure 3 shows an example of database reengineering co-transformations. The top of the Figure depicts the structural mapping T of a compound, semantics-preserving schema transformation replacing a complex COBOL record type (`ORD`) with two SQL tables (`ORDERS` and `DETAILS`). From this structural mapping, we can derive the rewrite rule R that replaces each COBOL random `READ` statement accessing the file `ORDERS` with a piece of code accessing the new SQL tables. This new code simulates the behavior of the initial COBOL `READ` statement in such a way that the application programs logic is left unchanged.

5 Tool support for co-Transformations

5.1 Schema Transformation Support

We use the transformation toolkit of DB-MAIN [3] to carry out schema transformations that are successively applied during the schema conversion phase.

5.2 Mapping Definition Support

DB-MAIN automatically generates and maintains a history log of all the transformations that are successively applied to the legacy DB schema (LDS) to obtain the new DB schema (NDS). This history is formalized in such a way that it can be analyzed and transformed. Particularly, it can be used to derive both the forward and backward mappings between the LDS and the NDS.

These mappings form the T part of the database conversion while its t part defines the data transformation.

5.3 Program Transformation Support

In the wrapper-based code organization, the migration strategy performs program conversion in two steps. First a data wrapper that simulates the DML of the legacy DMS on top of the new database is generated. Then, the legacy programs are transformed such that they access the data through the wrapper generated.

Wrapper generation

Wrapper generators for COBOL-to-SQL and CODASYL-to-SQL data migration have been developed through plug-in of the DB-MAIN tool. These generators take the LDS-to-NDS mapping as an input and generate the code that provides the application programs with a legacy interface to the new database. The wrapper solves three mismatch problems, namely model, API and schema [7].

Legacy programs alteration

The automation of legacy programs transformation relies on the ASF+SDF Meta-Environment. The general architecture we use is presented in [13]. We define the grammar of the legacy language (using the SDF formalism) and we specify a set of rewrite rules (ASF equations) needed to produce the program transformation tool. This transformation tool takes as input the program to be transformed as well as a formalized representation of the LDS-to-NDS mapping generated by DB-MAIN.

6 Conclusions

The problem of automatically migrating both the database and the application programs of large legacy Information Systems still is an unsolved problem, but in some special contexts such as the poor *one-to-one* conversion. The proposed approach has been formalized from the strong database reverse engineering methodology we have developed in the LIBD, and profits from the transformational technology we have built for database conversion.

The prototype tool we implemented in the DB-MAIN platform is now operational and has proved quite efficient. We have used our tools to re-engineer a COBOL application using IDS/II (CODASYL) DMS. This application is composed of 60 programs, totaling 35 KLOC and the database has 24 record

types connected by 13 set types (relationship types). The reverse engineering (recovering the conceptual schema) took 10 days. The design of the new SQL database took one day and the migration of the data one day. The transformation of the programs took about 5 minutes. Adapting the the tools to a new COBOL grammar required 2 days.

References

[1] E. J. Chikofsky and J. H. Cross II, Reverse Engineering and Design Recovery: A Taxonomy, in *IEEE Software*, volume 7, number 1, pp. 13–17, 1990.

[2] A. Cleve, Data-centered Applications Conversion using Program Transformations, Master's Thesis, University of Namur, 2004.

[3] J.-L. Hainaut, C. Tonneau, M. Joris and M. Chandelon, Schema Transformation Techniques for Database Reverse Engineering, in *Proc. 12th Int. Conf. on Entity-Relationship Approach*, Arlington-Dallas, E/R Institute Publish., 1993.

[4] J.-L. Hainaut, Specification Preservation in Schema Transformations - Application to Semantics and Statistics, *Data & Knowledge Engineering*,16(1), Elsevier Science Publish., 1996.

[5] J.-L. Hainaut, Chapter 1. Transformation Based Database Engineering, in *Transformation of Knowledge, Information and Data: Theory and Applications*. IDEA Group, 2005.

[6] J. Henrard, J.-M. Hick, Ph.Thiran and J.-L. Hainaut, Strategies for Data Reengineering, In *Proc. of Working Conference on Reverse Engineering (WCRE'02)*, pp. 211–220, IEEE Computer Society Press, 2002.

[7] J. Henrard, A. Cleve and J.-L. Hainaut, Inverse Wrappers for Legacy Information Systems Migration, In *Proc. of First International Workshop on Wrapper Techniques for Legacy Systems (WRAP'04)*, CS-Report 04/34, 30–43, TU Eindhoven Publish., 2004.

[8] J. Kort, R. Lämmel and C. Verhoef. The Grammar Deployment Kit. In Mark van den Brand and Ralf Lämmel, editors, *Electronic Notes in Theoretical Computer Science*, volume 65, Elsevier Science Publishers, 2002

[9] R. Lämmel, Grammar Adaptation, In *Proc. Formal Methods Europe (FME) 2001*, pp. 550–570, Lecture Notes in Computer Science, volume 2021, Springer-Verlag, 2001.

[10] R. Lämmel, Coupled Software Transformations (Extended Abstract), In *Proc. First International Workshop on Software Evolution Transformations*, 2004.

[11] R. Lämmel, Transformations Everywhere, to appear in *Science of Computer Programming*, 2004.

[12] W. Lohmann, Two co-transformations of grammars and related transformation rules, In *Softwaretechnik-Trends*, volume 24, pp. 78-79, May 2004.

[13] A. Sellink and C. Verhoef, An Architecture for Automated Software Maintenance, in *Proc. of Seventh International Workshop on Program Comprehension (IWPC99)*, May 5-7, 1999 Carnegie Mellon University, Pittsburgh, PA, USA.

[14] M.G.J. van den Brand, P. Klint and J.J. Vinju, Term Rewriting with Traversal Functions. *ACM Transactions on Software Engineering*, 12(2), pp. 152-190, 2003.

[15] M.G.J. van den Brand and P. Klint, *ASF+SDF Meta-Environment User Manual*, September 2004.

[16] A. van Deursen, P. Klint and C. Verhoef, Research Issues in Software Renovation. In J.-P. Finance, editor, *Proceedings Fundamental Approaches to Software Engineering (FASE99)*, pp. 1-23. Lecture Notes in Computer Science, Springer-Verlag, 1999.

[17] C. Verhoef, Towards Automated Modification of Legacy Assets, in *Annals of Software Engineering*, Volume **9**, Issue 1-4, pages 315-336, Science Publishers, 2000.

[18] www.program-transformation.org

[19] www.db-main.be