Large-scale Data Reengineering: Return from Experience

Jean Henrard and Didier Roland
ReveR S.A.
24, Boulevard Tirou
6000 Charleroi, Belgium
{*jean.henrard,didier.roland*}*@rever.eu*

Anthony Cleve and Jean-Luc Hainaut
University of Namur
21, rue Grandgagnage
5000 Namur, Belgium
{*acl,jlh*}*@info.fundp.ac.be*

Abstract

This paper reports on a recent data reengineering project, the goal of which was to migrate a large CODASYL database towards a relational platform. The legacy system, made of one million lines of COBOL code, is in use in a Belgian federal administration. The approach followed combines program transformation, data reverse engineering, data analysis, wrapping and code generation techniques.

1. Introduction

This paper reports on experience gained from a recent industrial migration project conducted in a Belgian public administration. The goal of the project was to migrate a network database (IDS/II[1]) towards a relational platform (DB2). The legacy COBOL system runs on a BULL GCOS8 mainframe and is made of about 1000 programs, totaling more than one million lines of code. The target system must consist of the same COBOL programs running on the mainframe but remotely accessing a DB2 database through a DBSP[2] gateway.

The methodology used in this project comprises two main phases, as shown in Figure 1. The first phase involves the refactoring of the legacy application programs, while the second phase aims at migrating the system towards the relational database platform.

2. System Refactoring

The initial *system refactoring* phase consists in centralizing the data manipulation statements within additional procedures. Each procedure is dedicated to a particular database operation applied to a particular record type.

[1] IDS/II is the CODASYL implementation for BULL mainframes.
[2] DBSP stands for DataBase Server Processor [1].

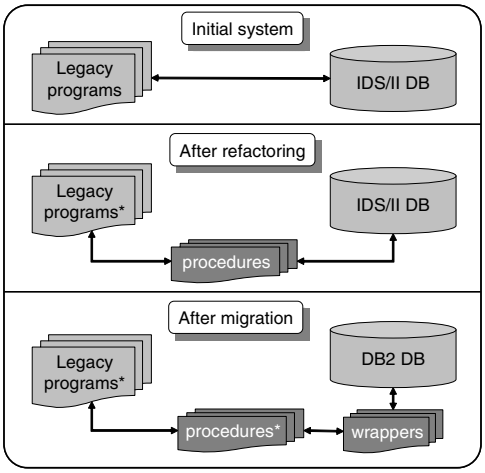

Figure 1. Two phase legacy code migration.

These procedures are generated from the legacy database schema. The application programs are then automatically transformed in such a way that each database operation is replaced with the corresponding procedure call.

The main purpose of the refactoring phase is to facilitate the migration phase itself by isolating the data manipulation primitives from the legacy source code. Indeed, no further alteration of the refactored legacy programs will be necessary at the time of migrating the database.

The system refactoring phase is supported by two distinct tools. The first one is a plugin of DB-MAIN [4] that generates the additional procedures from the source IDS/II schema. The generated procedures are COBOL code sections distributed in a set of COBOL *copybooks* (i.e., include files). Each copybook corresponds to an independent subset of the legacy schema.

The second tool, implemented on top of the ASF+SDF Meta-Environment [6], allows the transformation of the legacy programs. The source code transformation involves (1) the replacement of each IDS/II primitive with a proce-

1095-1350/08 $25.00 © 2008 IEEE
DOI 10.1109/WCRE.2008.14

	Transformed	Manually adapted
# programs	996	15
# copybooks	18	0
# FIND	7 632	21
# READY	5 891	0
# GET	5 345	0
# FINISH	1 144	0
# MODIFY	1 008	0
# STORE	942	0
# ERASE	433	0
# PRECALCULATE	10	0

Table 1. Legacy code refactoring results

dure call, (2) the insertion of new variable declarations and (3) the introduction of the generated procedures by means of COPY statements[3]. The transformation tool takes as inputs (1) a COBOL program, (2) specific information about the legacy schema and (3) conventions for procedure names.

Table 1 summarizes the results obtained during source code refactoring. A total of 22 405 IDS/II statements have been rewritten as procedure calls (PERFORM statements). We notice that more than 98% of the legacy programs were fully transformed without manual intervention. Manual transformations became necessary in the presence of some variants of the FIND statement. For instance, a record can be retrieved based on its technical identifier (called DB-KEY) that is *global* to the entire database. It is not always possible to identify the record type accessed by such a statement using static analysis.

We observed an increase of the legacy source code size of about 40 KLOC (+4%). This small expansion is mainly due to the fact that each rewritten IDS/II statement is kept as a comment in the target source code. The generated procedures consist of 42 000 lines of COBOL code.

3. System Migration

In the second phase (*system migration*), the legacy database is migrated and the additional procedures are regenerated such that they now access the target database through *inverse wrappers* [3]. These wrappers are dedicated data access components providing the legacy programs with a legacy API to the migrated data.

3.1. Schema Conversion based on Data Reverse Engineering

Our approach to converting the database schema involves a data reverse engineering process as a first stage.

[3]The COBOL COPY statement is similar to the *include* verb used in languages like C and C++. It allows programs to include frequently used source code fragments (in this case, the generated copybooks).

	IDS/II	Refined	Conceptual	DB2
# entity types	112	112	105	147
# rel. types	73	118	110	144
# attributes	1 249	1 509	1 210	1 841

Table 2. Comparison of successive versions of the database schema

This process is essential since the quality of the target database largely depends on it. The main goal of data reverse engineering is to recover the precise structure and meaning of the legacy data, by obtaining the so-called *conceptual schema* of the legacy database. This schema comprises both explicit and implicit schema constructs (structure and constraints).

The implicit constructs are identified through legacy code inspection. Dataflow analysis techniques are used to understand the way applications use and manage the data, which allows to recover such implicit constraints as foreign keys, finer-grained record decomposition, and more expressive names for records and fields [2]. Obviously, the implicit constructs recovered by program analyzers are only *candidate* integrity rules, that still have to be validated. This can be done via user validation or data analysis.

The target relational schema is then derived from the recovered conceptual schema using a standard database design methodology [5]. This design phase is supported by DB-MAIN, offering a set of assisted schema transformations.

The schema conversion results obtained in this project are summarized in Table 2. For each schema, the table gives the number of (1) entity types (record types or tables), (2) relationship types (set types or foreign keys) and (3) attributes (fields or columns). Multiple implicit constructs were discovered including 76 foreign keys, finer-grained attribute decompositions, as well as attribute format and cardinality constraints. The refined schema contains a total of 1509 fields among which 655 revealed to be optional.

3.2. Data Analysis

Once the target schema has been produced and implemented, the data instances themselves must be migrated towards the new platform. In this context, data quality is of primary importance. Indeed, the migrated data must comply with the DDL[4] constraints of the target database in order to be succesfully migrated.

In this project we particularly focused on implicit foreign keys and format constraints. The data analysis process involved the following steps:

[4]DDL stands for Data Definition Language.

1. The legacy IDS/II data were loaded into an intermediate DB2 database, in which each column is of type string, allowing error-free data migration. The DB2 database contained more than 415 millions of rows.

2. The underlying database schema was enriched with the following annotations:

   - The expected format of each DB2 column;
   - The validity constraints associated to each column (expressed as SQL *where* clauses);
   - The implicit foreign keys between tables.

3. Starting from this annotated schema, a data analysis program generated SQL queries for inspecting the intermediate DB2 database. This analysis produced an html document reporting on the detected data errors.

The data analysis tool considered 76 implicit foreign keys, among which 24 proved to be violated by the legacy data. Regarding data format, a total of 3497 SQL queries were executed, allowing the detection of data inconsistencies in almost 70% of the record types.

A typical problem regarding format constraints was the presence of invalid dates. In the IDS/II database, date fields are represented as numeric values of the form 'YYYYMMDD'. In the target DB2 database, they are expressed as columns of SQL type `Date`. We encountered a large amount of inconsistent dates like '20070931'. These errors are mainly due to the behavior of some application programs considering day '31' as the last day of the current month, whatever the month. We also identified special date values like '00000000' or '99999999' that are used by programs for simulating the *null* value and a future date, respectively.

For several reasons, all the detected data inconsistencies could not be corrected by the customer, which obliged us to relax some constraints. For instance, a subset of the foreign keys have not been explicitly declared in the target DDL code. They are rather managed by fault-tolerant triggers that produce a warning in a dedicated log table each time the referential constraint is violated by a database operation. This approach allows to manage the referential constraints while avoiding to disturb the execution of legacy applications.

3.3. Data Migration

The data migration itself consists of an *extract-transform-load* process, which can be automated provided that the mapping that holds between the source and target DB schemas is explicitly available. According to our approach, such a mapping is maintained and propagated while successive transformations are applied to the source schema. Both the IDS/II and DB2 schemas are annotated with correspondence information which allows us to automatically generate the data migration program.

3.4. Wrapper Generation and Interfacing

Similarly, the source code of the wrappers is derived from the source-to-target schema mapping by a dedicated DB-MAIN plugin. The generated wrappers simulate each legacy DML statement on top of the target database. In the context of the present project, the wrappers translate IDS/II primitives using Embedded SQL code fragments.

Interfacing the generated wrappers with the legacy code simply consists in rewriting the procedures introduced during the system refactoring phase. Instead of accessing the IDS/II database, the procedures may now invoke the wrappers in order to access the DB2 database.

4. Discussion

Our system migration approach based on data reverse engineering has the merit of producing a high-quality, fully-documented target database. In this project, the DB2 database is designed as a native, normalized relational database, which does not look like the network database it was derived from. This means that new applications (in this particular case, web applications) can now directly access the DB2 database without invoking the wrappers. In addition, this approach allows to smoothly migrate or rewrite the legacy code, until the wrappers become useless.

Our program conversion methodology permits much flexibility in the migration process itself. Once the additional procedure copybooks have been introduced, the database can be incrementally migrated towards the target platform (subset by subset), while iteratively replacing the corresponding copybooks. Furthermore, the legacy programs refactoring preserves the readability of the target source code. The developers of the legacy application still recognize their programs and can continue to maintain them. The "only" difference is that each IDS/II instruction is now written as a procedure call. The logic of the legacy programs is not modified at all. The new database (and its new paradigm) is hidden behind the generated wrappers.

The main drawback of our approach concerns performance. The resulting programs are between 3 and 6 times slower than the original application, depending on the type of database access. There are two explanations for such a performance penalty. It is partly due to the use of wrappers that simulate the legacy data access. In case some programs become too slow, it is possible to rewrite them such that they directly access the migrated database. Such an adaptation can make the program up to five time faster than the version using the wrapper. Indeed, the way of navigating into a

network is completely different from the way of manipulating a relational database. The second cause of performance degradation is the architecture of the target system in this particular project. In the legacy system, the IDS/II database is stored onto disk of the mainframe. In the new system, due to the use of DBSP, the BULL mainframe sends its query to a Unix machine that itself forwards the query to the DB2 server. In other words, each query execution necessitates four traversals of the network!

5. Lessons Learned

Full automation is not realistic Large-scale data reengineering projects require scalable tool support. Although a high level of automation can be reached, fully-automating the process is clearly unrealistic. Indeed, such projects generally require several iterations and involve multiple human decisions.

Forward engineering is not straightforward Deriving a relational DDL code from a conceptual database schema can theoretically be performed automatically. In practice, when dealing with real schemas, external constraints must be taken into account during the schema design process. In this project, several decisions were guided by customer preferences like column type selection, naming conventions, and conversion strategies for compound and multivalued fields.

Data analysis is essential While data reverse engineering allows to discover implicit schema constraints, making all these rules explicit in the target schema is not always achievable. This was the case in this project, where a significant subset of the legacy data instances proved to violate the implicit rules recovered by program inspection. We also learned that database reverse engineering may benefit from data analysis. Indeed, analyzing database contents does not only allow to detect errors or to validate integrity rules. It may also serve as a basis for formulating new hypotheses about potential implicit constraints. The main limitation is that, as seen above, we can not assume that the database is in a consistent state.

Wrapper development is challenging Developing correct wrappers requires a precise knowledge of the legacy data manipulation system. In this project, the task was challenging due to the paradigm shift between network and relational database systems. Indeed, the generated wrappers must precisely simulate the behaviour of the IDS/II primitives, which includes the management of currency indicators, reading sequences and returning status codes. Another challenge, as for the data migrators, was to correctly deal with IDS/II records that had been split into several tables.

6. Conclusions

This paper illustrates the reality of large-scale data reengineering projects, and shows that such projects may benefit from a systematic, tool-supported methodology. It also confirms that data reverse engineering is greatly valuable for producing a high-quality target database. The approach presented in this paper relies on the balanced combination of analysis, generative and transformational techniques, leading to a good compromise between a high-level of automation and maintainability of the target database and programs.

We anticipate different R&D directions for the next future. First, we intend to investigate other program conversion strategies according to which the logic of the programs is adapted to the target database platform. Such a strategy could allow to improve the performance of the target applications. A second research direction concerns the identification and correction of *unsafe data access paths* in application programs, i.e., source code fragments where implicit constraints may potentially be violated. Finally, we are currently exploring the use of dynamic analysis techniques in the context of data reverse engineering. Indeed, our experience shows that static analysis may fail to extract accurate dataflow relationships in the presence of dynamically generated database operations.

Acknowledgments Anthony Cleve was supported by the Belgian *Région Wallonne* and the European Social Fund via the RISTART project.

References

[1] BULL. DBSP: Database Server Processor, 2001. http://www.bull.com/servers/gcos8/products/dbsp/dbsp.htm.

[2] A. Cleve, J. Henrard, and J.-L. Hainaut. Data reverse engineering using system dependency graphs. In *Proc. of the 13th Working Conference on Reverse Engineering (WCRE'06)*, pages 157–166, 2006.

[3] A. Cleve, J. Henrard, D. Roland, and J.-L. Hainaut. Wrapper-based system evolution application to codasyl to relational migration. In *Proc. of the 12th European Conference on Software Maintenance and Reengineering (CSMR'08)*, pages 13–22. IEEE CS Press, 2008.

[4] DB-MAIN. http://www.db-main.be, 2006.

[5] T. J. Teorey, D. Yang, and J. P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Comput. Surv.*, 18(2):197–222, 1986.

[6] M. van den Brand, A. van Deursen, J. Heering, H. de Jong, M. de Jonge, T. Kuipers, P. Klint, L. Moonen, P. Olivier, J. Scheerder, J. Vinju, E. Visser, and J. Visser. The ASF+SDF Meta-Environment: A component-based language development environment. In *Compiler Construction (CC '01)*, volume 2027 of *LNCS*, pages 365–370. Springer-Verlag, 2001.