An Industrial Experience Report on Legacy Data-Intensive System Migration

Jean Henrard and Didier Roland
ReveR S.A.
130, 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 presents an experience report on the migration of a COBOL system of over 2 million lines of code. The main goal of this project was to migrate a legacy CODASYL database towards a relational platform, while preserving the functionalities of the legacy application programs.

1. Introduction

Replacing a database platform with another one should, in an ideal world, only impact the database component of the software system. Unfortunately, the database most often has a deep influence on other components, such as the application programs. Two reasons can be identified. First, the programs invoke data management services through an API that generally relies on complex and highly specific protocols. Changing the database technology, and therefore its protocols, involves the rewriting of the invocation code sections. Second, the database schema is the technical translation of its conceptual schema through a set of rules that is dependent on the data model. Porting the database to another platform, and therefore to another data model, generally requires another set of rules, that produces a significantly different database schema. Consequently, the code of the programs often has to be adapted to this new schema.

This paper reports on an ongoing migration project, which aims at migrating a COBOL system using an IDS/II\textsuperscript{1} database towards a relational (DB2) database platform. The legacy system runs on a BULL GCOS8 mainframe and is made of nearly 2300 programs, totaling more than 2 million lines of code. The target system must comprise the same COBOL programs running on the mainframe but must remotely access a DB2 database through a DBSP\textsuperscript{2} gateway.

\textsuperscript{1}IDS/II is the BULL implementation of CODASYL.
\textsuperscript{2}DBSP stands for DataBase Server Processor [2]

2. Process Followed

In this section, we describe the main phases that we followed during this migration project, namely inventory, database conversion and program conversion.

2.1. Inventory

The purpose of the inventory process is twofold. First, it aims at checking that we have received a complete and consistent set of source code files from the customer. Second, it helps us to get a rapid overview of the application architecture in order to evaluate the complexity of the migration task. In particular, we extract all the data manipulation verbs to evaluate which part of the work cannot be done automatically. We also compute the call graph and usage graph of the application. A call graph represents the calling relations between two programs, while the usage graph represents which record type or file is used by each program.

The inventory step was supported by four tools. The first one cleans the COBOL source code (e.g., removes line numbers), resolves the copybooks and produces a report summarizing the missing copybooks. The second tool parses the source code to check that all the code fragments can be analyzed and to list all the calls and data manipulation instructions. The third tool analyzes the JCL to find out which (physical) files are used by each program. The last one, DB-MAIN\textsuperscript{3}, is used to store and manipulate the call

\textsuperscript{3}DB-MAIN is a general-purpose database engineering CASE tool [4].
and usage graphs that are created from the report produced by the second tool.

We have analyzed 2,273 programs that contain 64,809 calls, 20,856 IDS/II verbs and 23,060 file access verbs. The call graph contains 2,273 nodes (programs) and 9,527 call relationships. The usage graph contains 313 (programs) + 218 (records) nodes and 2,887 usage vertices. Only 313 programs appear in the usage graph because the batch programs make use of data access modules, thus very few batch programs actually contain IDS/II verbs.

2.2. Database Conversion

The database conversion phase aims at migrating the legacy database towards a modern platform. This process consists of two main steps, namely schema conversion and data migration.

2.2.1 Schema Conversion

The objective of the schema conversion step is to translate the legacy database schema into an equivalent schema expressed in the target technology. Our approach to schema conversion, depicted in Figure 1, consists in designing the target database schema after an initial database reverse engineering (DBRE) phase.

Database Reverse Engineering

The goal of the DBRE process is to recover the precise structure and meaning of the legacy database schema. Our tool-supported methodology for DBRE [5] starts with the DDL analysis process, parsing the legacy DDL code (IDS/II schema and sub-schema in this case) to retrieve the raw physical schema. The schema refinement step [6] consists of an in-depth inspection of the way the application programs use and manage the data. Through this process, additional structures and constraints are identified, which are not explicitly declared in DDL code but expressed in the procedural code. For instance, implicit foreign keys and finer-grained decompositions of records are recovered. The existing data can also be analyzed, either to detect constraints, or to confirm or discard hypotheses on the existence of such constraints.

The final DBRE step is the data structure conceptualization interpreting the legacy refined schema into the conceptual schema. Both schemas have the same semantics, but the latter is platform independent and includes no technical constructs.

The various tools we use to support the DBRE phase include DB-MAIN (DDL extraction, schema storage, analysis and manipulation), a COBOL dataflow analysis tool and a data analysis program generator. The dataflow analyzer allows to discover what program variables are used to access database records, in order to produce better record decompositions. It also helps to recover implicit data dependencies that hold between database fields, e.g. foreign keys.

2.3. Database Design

The database design phase consists in converting the conceptual schema, independent from any particular database paradigm, into a relational, DB2 compliant schema. This design phase is supported by the DB-MAIN CASE tool, offering a set of assisted schema transformations. Although this transformation can theoretically be performed automatically [1], the following aspects must be taken into account: (1) the way compound fields are flattened (disaggregation or aggregation) depends on their role and meaning; (2) multi-valued fields (arrays) must be transformed (list of columns, single aggregated column, or separated tables) according to various criteria; (3) the customer’s naming conventions must be respected; (4) the customer preferences also influence the way a hierarchy of records is translated as well as the column types chosen (some numeric fields and dates were expressed as strings).

Table 1 gives a comparison of three successive versions of the database schema. The refined IDS/II schema is the physical schema with a finer-grained structure, obtained by integrating copybooks. Most IDS/II data fields are declared multiple times through redefines clauses, hence the huge total number of attributes. The refined schema also comprises 89 implicit foreign keys, recovered through dataflow analysis and validated by the customer. The conceptual schema contains less entity types than the refined schema because some obsolete IDS/II record types have not been migrated, as asked by the customer. In the conceptual schema, the myriad of redefines have been resolved, which reduces the number of attributes per entity type. The redefine resolution process was supported by a dedicated tool automatically replacing the redefines by the most expressive definition (sometimes by composing a new definition from all the initial ones). Finally, the DB2 schema sees an increase in the number of entity types, due to the decomposition of ar-
Table 1. Comparison of successive versions of the complete database schema

<table>
<thead>
<tr>
<th></th>
<th>Physical</th>
<th>Refined</th>
<th>Conceptual</th>
<th>Relational</th>
</tr>
</thead>
<tbody>
<tr>
<td># ent. types</td>
<td>159</td>
<td>159</td>
<td>156</td>
<td>171</td>
</tr>
<tr>
<td># rel. types</td>
<td>148</td>
<td>148</td>
<td>90</td>
<td>0</td>
</tr>
<tr>
<td># attributes</td>
<td>458</td>
<td>9 027</td>
<td>2 176</td>
<td>2 118</td>
</tr>
</tbody>
</table>

Table 2. Program transformation results

<table>
<thead>
<tr>
<th></th>
<th>Migrated</th>
<th>Manually transformed</th>
</tr>
</thead>
<tbody>
<tr>
<td># programs</td>
<td>669</td>
<td>17</td>
</tr>
<tr>
<td># copybooks</td>
<td>3 917</td>
<td>68</td>
</tr>
<tr>
<td># IDS/II verbs</td>
<td>5 314</td>
<td>420</td>
</tr>
</tbody>
</table>

2.2.2 Data Migration

During the schema conversion phase, the mapping of the various components is recorded between the successive schemas, such that we know precisely how each concept is represented in each schema. From there, we can use a data migration program generator. The generated program reads the IDS/II database entirely, converts the data when necessary, and fills the new relational database without any loss. The first idea was to generate an intermediate XML file for data transfer, but a more efficient format, RDBMS-compliant, has finally been adopted.

2.3. Program Conversion

Once the legacy database has been migrated towards the relational platform, the application programs are converted in such a way that they access the new database instead of the legacy data. Our program conversion approach [3], based on wrapping techniques, consists of two main steps, namely wrapper generation and program transformation.

Wrapper Generation

A data wrapper, simulating the legacy data management system on top of the new database, is generated. 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 DMS, possibly converts them to the appropriate legacy format and delivers them to the application program.

The wrapper generation process takes as inputs the legacy IDS/II schema, the refined IDS/II schema, the target relational schema, as well as the mapping that holds between these three schemas. The generator produces the code that provides the application programs with a legacy interface to the new database. In practice, we generate one wrapper for each legacy record type actually migrated. In this project, 158 wrappers were generated totaling 458 KLOC. Each generated wrapper is a COBOL program containing embedded SQL primitives.

Program Transformation

During the program conversion step, the legacy source code is interfaced with the wrappers. The program adaptation consists of a combination of the following transformation steps: (1) replacing the IDS/II primitives with equivalent wrapper invocations; (2) renaming variables and adapting their type; (3) adding new variable declarations; (4) adding new generated code sections. The exact combination chosen depends on the kind of source code file to be transformed (as identified during the inventory phase).

The input parameters of the program transformation process are automatically generated from the same inputs as for wrapper generation. The automation of the program transformation process itself relies on the ASF+SDF Meta-Environment [8]. We adapted an SDF version of the IBM VS COBOL II grammar [7], by adding a syntax module dedicated to IDS/II primitives and conditions. We specified a set of rewrite rules (ASF equations) on top of this enriched grammar to obtain our program transformation tools.

The results obtained during the legacy code adaptation are summarized in Table 2. A total of 669 programs and 3 917 copybooks were migrated. We notice that about 92% of the IDS/II verbs were converted automatically, while the manual work concerned only 85 distinct source code files.

3. Evaluation and Lessons Learned

Inventory

As in many previous projects, the inventory phase proved to be critical. It required several iterations since we discovered some missing copybooks and programs, as well as code fragments containing syntax errors. From the call and usage graphs, it was possible to list the programs that (indirectly) access the IDS/II database, and thus that must be transformed. The precise knowledge of the IDS/II verbs used helped us to precisely evaluate the proportion of code requiring a manual adaptation.

The call and usage graphs also allowed us to build a consistent subset of programs (and sub-programs) that only use a particular subset of the database. Indeed, it was decided to first perform the complete migration on top of a small subset of the database. This proof-of-concept phase aimed at proving the feasibility of the project to the customer, by verifying the correctness of the overall migration through a systematic testing process. The database subset includes 26 IDS/II record types and 31 sets. The legacy programs se-
lected for conversion comprise 51 KLOC and make use of almost every possible IDS/II statement.

Database Reverse Engineering The program analysis step required more time than expected, because it was the first project of that size and complexity supported by our tools. Several bugs were found, mainly due to different programming styles. A few subtle errors were discovered during the validation of the results, requiring some extra manual work.

The elimination of unnecessary redefine statements also took a lot of time. Although we have developed a dedicated tool to automate this task, we encountered many exceptions requiring to be resolved by hand.

During the data analysis step, we discovered that many implicit referential constraints were actually violated by the legacy data. This is explained by the fact that most rules are simply encoding rules which are not always checked again when data are updated, and by the fact that users find tricks to bypass some rules.

Database Design The larger the schema to convert, the more we would like to automate the process, but the reality shows something different: the larger the schema, the more techniques should be used to convert the numerous structures, and the more important it is to be rigorous on aspects such as naming or type conventions. While previous smaller projects allowed us to automate the schema design process with minor manual corrections, assisted manual conversion becomes necessary when dealing with larger schemas.

Data Migration The initial idea of using a generic scheme of downloading all the data to an XML file and uploading that file to the new database has to face the reality of performance and volume constraints. A dedicated tool browsing through the original database in a single pass and directly generating flat RDBMS-friendly files was necessary.

Program Conversion Writing correct wrapper generators requires a very good knowledge of the legacy DMS. In this project, the difficulties of the wrapper generation task was due to the paradigm shift between network and relational database systems. Simulating IDS/II verbs on top of a native relational database appeared much more complicated than expected. The generated wrappers must precisely simulate the IDS/II primitives behavior, which includes the management of currents, reading sequence orders and returning status codes. Another challenge, as for the data extractors, was to correctly manage IDS/II records that had been split into several tables. The testing effort was also very time-consuming, mainly due to current management issues. The testing process has been performed with the help of one of the customer IDS/II experts.

4. Conclusions

The industrial project described in this paper confirmed that semantics-based database migration is achievable. This approach has the merit of producing a high quality and fully documented target database, enriched with implicit structures and constraints. The resulting native relational database forms a sound basis for future programs in terms of maintainability, while the wrapper-based program conversion strategy allows the legacy programs to access the relational data with minimal adaptation.

Although large-scale system conversion needs to be supported by scalable tools, fully-automating the process is clearly unrealistic. Indeed, such a project typically requires several iterations as well as multiple human decisions. In this direction, the prototyping phase, during which a consistent subset of the data and programs has been fully migrated, allowed us to detect problems early in the process and to better confront our results with the customer requirements.

The last step of the project, that still is ongoing, consists in systematically testing each transformed program. During this process, we intend to collect performance figures in order to evaluate the impact induced by the wrapper-based data access combined with the DBSP gateway.

References