The Nature of Data Reverse Engineering

Jean-Luc Hainaut, Jean Henrard, Jean-Marc Hick, Vincent Englebert, Didier Roland
Institut d'Informatique, University of Namur
rue Grandgagnage, 21 - B-5000 Namur (Belgium) • *jlh@info.fundp.ac.be*

Abstract

Data Reverse Engineering is a specific information system engineering domain aiming at rebuilding the documentation of legacy databases. The paper explores this domain by presenting the basic problems (such as the implicit construct recovery problem) as well as techniques and tools for solving them. Two main phases are described, namely data structure extraction, through which the complete logical schema is elicited, and data structure conceptualization, that interpret them in conceptual terms. The paper describes and discusses a reference process model that identifies the main classes of problems, their solving techniques and tools that support the processes.

1. Introduction

Data reverse engineering is that part of Information System Engineering that addresses the problems and techniques related to recovering of abstract descriptions of files and databases of legacy systems. In this paper, we try to put into light the main aspects that make this engineering domain a discipline of its own.

1.1. What is Data Reverse Engineering?

First of all, we have to delineate the application domain of this discipline. Data reverse engineering concerns *Legacy Information Systems*. According to [7], the latter can be defined as *data-intensive applications, such as business systems based on hundreds or thousands of data files (or tables), that significantly resist modifications and changes.*

The objective of data reverse engineering can then be sketched as follows: to recover the technical and conceptual descriptions of the permanent data of a legacy information system, i.e., its database, be it implemented as a set of files or through an actual database management system.

By *technical description*, we mean the statement of what are the files, the record types, the fields and their data types, the relationships and the constraints. The technical description is formally expressed in the so-called *Logical schema*. Practically, this is the precise description of the structures and time-independent properties of the data that would be required by a programmer in order to develop a new application on the legacy data in a reliable way. As we can guess, this description encompasses, but can go well beyond, the data structures explicitly declared by the DDL[1] specifications of the system.

The *conceptual description* of the data is an abstract (that is, independent of the data management system) expression of what these data structures mean, i.e., its semantics. This description is expressed as a Conceptual schema.

1.2. Is Data Reverse Engineering really that difficult?

When one analyzes the commercial offering in CASE support for data reverse engineering, one gets the feeling that the problem has been strongly exaggerated. Indeed, this problem often is presented as follows. Considering the DDL code below,

```
create table CUSTOMER (
   CNUM .. not null,
   CNAME ..,
   CADDRESS ..,
   primary key (CNUM))
create table ORDER (
   ONUM .. not null,
   SENDER .. not null,
   ODATE ..,
   primary key (ONUM),
   foreign key (CNUM) references CUSTOMER))
```

... we derive the conceptual schema of Figure 1 that expresses its semantics in an abstract way:

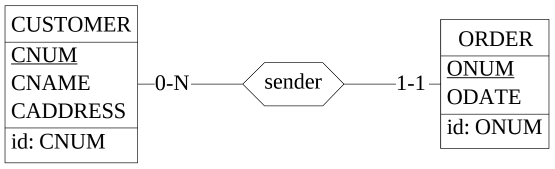


Figure 1. A naive view of data reverse engineering

1 The Data Description Language (DDL) is the part of the database management system facilities intended to declare or build the data structures of the database.

If data reverse engineering were that simple, it would not require much research nor sophisticated tool development. Unfortunately, *true* data reverse engineering is much closer to the following scenario, in which this chunk of COBOL code . . .

```
select CF008 assign to DSK02:P12
 organization is indexed
 record key is K1 of REC-CF008-1.

select PF0S assign to DSK02:P27
 organization is indexed
 record key is K1 of REC-PFOS-1.

fd CF008.
 record is REC-CF008-1.
 01 REC-CF008-1.
    02 K1 pic 9(6).
    02 filler pic X(125).

fd PF0S.
 records are REC-PF0S-1,REC-PF0S-2.
 01 REC-PF0S-1.
    02 K1.
       03 K11 pic X(9).
       03 filler pic 9(6)
    02 filler pic X(180).
 01 REC-PF0S-2.
    02 filler pic X(35).
```

. . . has to be interpreted as the implementation of the conceptual schema of Figure 2.

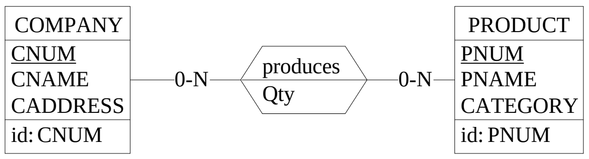


Figure 2. A more realistic view of data reverse engineering.

No miracle here: getting such a result needs additional sources of information, which will prove much more difficult and tedious to analyze than mere DDL sections.

1.3. Why Data Reverse Engineering?

Data reverse engineering is not the most exciting engineering activity. Indeed, diving during weeks and even months into a muddy sea of tricky COBOL, RPG or Basic code cannot be as exhilarating as building a new web site adorned with nice dancing Java applets.

Basically, DRE seldom is a goal *in se*, but most often is the first step in a broader engineering project. It is generally intended to redocument, convert, restructure, maintain or extend legacy applications. Here follow some of the most frequent objectives of database reverse engineering.

Knowledge acquisition in system development. During the development of a new system, one of the early phases consists in gathering and formalizing users requirements from various sources such as user interviews and corporate document analysis. In many cases, some partial implementation of the future system may exist already, for instance in the form of a user-developed small system, the analysis of which can bring early useful information.

System maintenance. Fixing bugs and modifying system functions require understanding the concerned component, including, in data-centered systems, the semantics and the implementation of the permanent data structures.

System reengineering. Reengineering a system is changing its internal architecture or rewriting the code of some components without modifying the external specifications. The overall goal is to restart with a cleaner implementation that should make further maintenance and evolution easier. Quite obviously, the technical aspects as well as its functional specifications have to be clearly understood. The same will be true for the other three objectives whose description follows.

System extension. This term designates changing and augmenting the functional goals of a system, such as adding new functions, or its external behavior, such as improving its robustness.

System migration. Migrating a system consists in replacing one or several of the implementation technologies. IMS-to-DB2, COBOL-to-C, monolithic-to-Client-server, centralized-to-distributed are some widespread examples of system migration.

System integration. Integrating two or more systems yields a unique system that includes the functions and the data of the former. The resulting system can be physical, in which case it has been developed as a stand-alone application, or it is a virtual system in which a dynamic interface translates the global queries into local queries addressed to the source systems. The most popular form of virtual integrated system is the federated database architecture.

Quality assessment. Analyzing the code and the data structures of a system in some detail can bring useful hints about the quality of this system, and about the way it was developed. M. Blaha observed that assessing the quality of a vendor software database through DBRE techniques is a good way to evaluate the quality of the whole system [5].

Data extraction/conversion. In some situations, the only component to salvage when abandoning a legacy system is its database. The data have to be converted into another format, which requires some knowledge on its physical and semantic characteristics. On the other hand, most data warehouses are filled with aggregated data extracted from corporate databases. This transfer requires a deep understanding of the physical data structures, to write the extraction routines, and of their semantics, to interpret them correctly.

Data Administration. DBRE is also required when one develops a data administration function that has to know and record the description of all the information resources of the organization.

Component reuse. In emerging system architectures, reverse engineering allows developers to identify, extract and wrap legacy functional and data components in order to integrate them in new systems, generally through ORB technologies [29] [30].

1.4. Data reverse engineering *vs* Program reverse engineering

Of course, reverse engineering encompasses a much broader domain than data alone. Hence the question: what are the relations between *software reverse engineering* and *data reverse engineering*? We will just point out two observations.

1. It is impossible to fully understand a (business) program until the main data structures, and particularly the file and database structures it uses, are fully understood.

2. As will be discussed in this paper, it is impossible to fully understand the data structures of a set of files or of a database without a clear understanding of the program sections that manipulate them. This is particularly true for the sections that are in charge of checking the correctness of the data before their being stored in the database.

Despite this elegant symmetry, program reverse engineering and data reverse engineering appear to differ substantially as far as their objectives are concerned.

The goal of program RE basically is to extract abstractions or specific patterns from the programs in order to understand some of its aspects. Hence the common term of *program understanding*. Recovering full functional specifications (e.g., in terms of pre- and post-conditions) must be considered as unreachable in the general case.

On the contrary, the very objective of data reverse engineering is to recover the (hopefully) complete technical and functional specifications of the data structures.

1.5. Specific DBRE problems

Recovering conceptual data structures can prove much more complex than merely analyzing the DDL code of the database. Untranslated data structures and constraints, non standard implementation approaches and techniques and ill-designed schemas are some of the difficulties that the analysts encounter when they try to understand existing databases from operational system components. Since the DDL code no longer is the unique information source, the analyst is forced to refer to other documents and system components that will prove more complex and less reliable. The most frequent sources of problems have been identified [2], [4], [18], [26], [27] and can be classified as follows.

Weakness of DBMS models: The logical model provided by the DMS can express only a subset of the structures and constraints of the intended conceptual schema.

Implicit structures: Some constructs have intentionally not been explicitly declared in the DDL specification of the database.

Optimized structures: For technical reasons, such as time and/or space optimization, many database structures include non semantic constructs.

Awkward design: Not all databases were built by experienced designers. Novice and untrained developers, generally unaware of database theory and database methodology, often produce poor or even wrong structures.

Obsolete constructs: Some parts of a database have been abandoned, and ignored by the current programs.

Cross-model influence: Some relational databases are actually straightforward translations of IMS or CODASYL databases, or of COBOL files.

. . . and, of course, *no documentation*!

2. The *Implicit construct* problem

A large part of the difficulties data reverse engineer have to face is eliciting implicit constructs. By this expression, we mean data structures or data properties, such as integrity constraints, that are an integral part of the database, though they have not been explicitly declared in the DDL specifications. Let us explain this concept through two implementation patterns.

Explicit vs *implicit foreign keys*

The following fragment of SQL-DDL code defines two tables that are explicitly linked by a foreign key[2]. The latter declares that, for any row of ORDER, the column OWNER references a row in table CUSTOMER.

```
create table CUSTOMER (
  CNUM   integer primary key,
  C_DATA char 80)
create table ORDER(
  ONUM   integer primary key,
  SENDER  integer
  foreign key (SENDER) references CUSTOMER)
```

In the implicit pattern, no foreign key has been declared. Instead, the application programs include code fragments that obviously ensure that data states violating the referential integrity are identified as being wrong.

2 A *foreign key* is a field (or column), or a sequence of fields, whose value is used to reference a record in another file (or table). The property of foreign key values forming a subset of the values of the unique key of the target record type (table) is called *referential integrity.*

```
create table CUSTOMER (
  CNUM   integer primary key,
  C-DATA char(80))
create table ORDER    (
  ONUM   integer primary key,
  SENDER  integer)
...
exec SQL
  select count(*) in :ERR-NBR
  from ORDER
  where SENDER not in
       (select CNUM from CUSTOMER)
end SQL
...
if ERR-NBR > 0
   then display ERR-NBR,
   'referential constraint violation';
```

Explicit* vs *implicit field structure of a record type

Normally, all the fields that are used in application programs are identified and given meaningful names, such as in the following COBOL fragment.

```
01 CUSTOMER.
   02 C-KEY.
      03 ZIP-CODE pic X(8).
      03 SER-NUM  pic 9(6).
   02 NAME     pic X(15).
   02 ADDRESS  pic X(30).
   02 ACCOUNT  pic 9(12).
```

Using implicit structures consists in coding the record type, or parts of it, as anonymous or unstructured fields, such as through the following code.

```
  01 CUSTOMER.
     02 C-KEY  pic X(14).
     02 filler pic X(57).
```

Of course, the application programs can recover the actual structure by storing records in local variables that have been given the correct detailed decomposition.

3. A reference process model for Data Reverse Engineering

Considering the amount of scientific and technical proposals in the realm of DBRE, we obviously need a general framework, hereafter called the *reference model*, in which each of them can be positioned and its contribution evaluated. The general architecture of the reference DBRE process model is outlined in Figure 3. It shows clearly the three main processes that will be described in the next sections.

During the ***Project Preparation*** phase, the main sources of information are identified and classified into three categories:

- explicit DDL code ($\mathbf{code}_{ddl}$)
- sources that control the implicit constructs ($\mathbf{code}_{ext}$), such as programs and triggers,

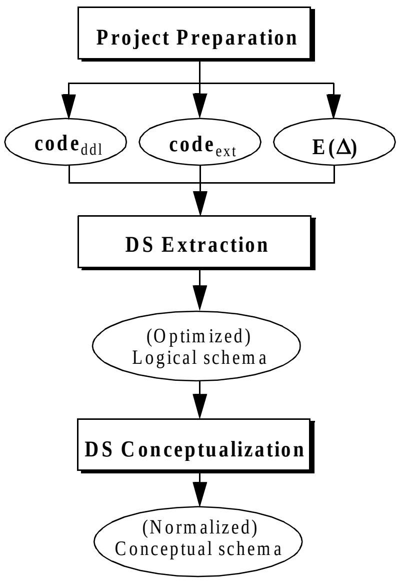


Figure 3. The main processes of data reverse engineering.

- other sources ($\mathbf{E}(\Delta)$) that can give evidence of implicit constructs, such as database contents and documentation.

The ***Data Structure Extraction*** aims at recovering the description of the data structures (the Logical schema) as seen and used by the programmer (relational, files, IMS, CODASYL, etc.).

The ***Data Structure Conceptualization*** consists in interpreting the data structures in abstract terms pertaining to the application domain (the Conceptual schema).

We want to make clear what we call *process* should be considered as a consistent and homogeneous knowledge domain that addresses an identified type of problems and is given a specific goal, such as DDL *code parsing*, *Data analysis* or *Schema normalization*.

4. Data Structure Extraction

This process aims at rebuilding a complete logical schema in which all the explicit and implicit structures and properties are documented. The main source of problems is the fact that many constructs and properties are implicit, i.e., they are not explicitly declared, but rather are controlled and managed through, say, procedural sections of the programs. Recovering these structures uses DDL code analysis, to extract explicit data structures, and data structure elicitation techniques, that lead to the recovery of implicit constructs. We identify four processes (Figure 4).

1. *DDL code Extraction*. Automatic parsing of the code to extract explicit data structures.
2. *Physical Integration*. Merging multiple views of the same data sets.

3. *Schema Refinement*. Recovering the implicit data structures and constraints.
4. *Schema Cleaning*. Removing physical constructs that bears no semantics.

Due to its importance and its complexity, we will describe the Schema Refinement process in further detail.

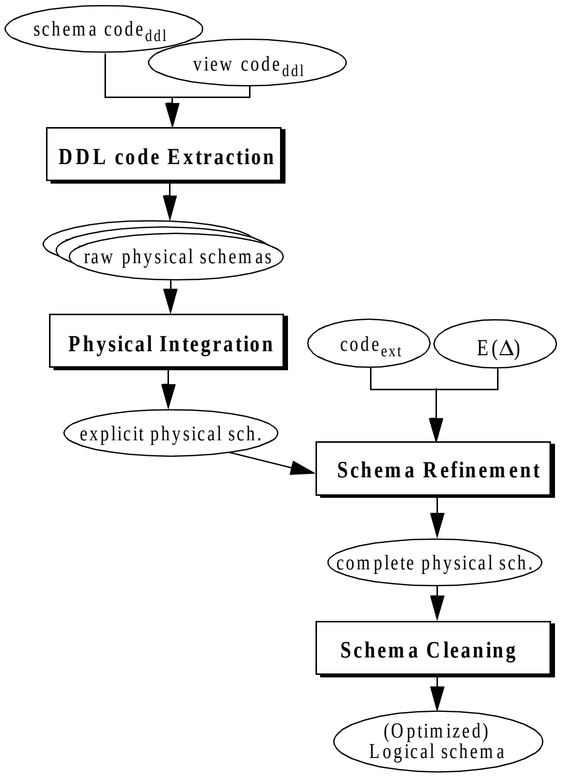


Figure 4. Development of the Data Structure Extraction process.

4.1. Schema Refinement

The organization of the process is sketched in Figure 5, that shows the main information sources. Some of them are discussed below.

4.1.1 The information sources

We will mention the main information sources from which the analyst can extract evidence of implicit constructs.

Application programs

The way data are used, transformed and managed in the programs brings essential information on the structural properties of these data. Programs require specific analysis techniques and tools. Dataflow analysis, dependency analysis, programming *cliché* analysis and program slicing are some examples of useful program processing techniques from the domain of program understanding.

Screen/form/report layout

A screen form or a structured report can be considered as derived views of the data. The layout of the output data as well as the labels and comments can bring essential information on the data.

External data dictionaries and CASE repositories

Third-party or in-house data dictionary systems allow data administrators to record and maintain essential descriptions of the information resources of an organization, including the file and database structures. The same can be said of CASE tools, that can record the description of database structures at different abstraction levels.

Data

The data themselves can exhibit regular patterns, or uniqueness or inclusion properties that provide hints that can be used to confirm or disprove structural hypotheses. The analyst can find evidence that suggests the presence of identifiers, foreign keys, field decomposition, optional fields or functional dependencies.

Non-database sources

Small volumes of data can be implemented with general purpose software such as spreadsheet and word processors. In addition, semi-structured documents are increasingly considered as a source of complex data that also need to be reverse engineered. Indeed, large text databases can be implemented according to representation standard such SGML, XML or HTML that can be considered as special purpose DDL.

Technical/physical constructs

There can be some correlation between logical constructs and their technical implementation. For instance, a foreign key is often supported by an index. Therefore, an index can be an evidence that a field could be a foreign key.

And of course, ***the current documentation***, if any.

4.2. Some elicitation techniques

These sources can be processed manually (i.e., visually), but they ideally should be analyzed through tool-supported specific elicitation techniques. We briefly describe some of them.

Schema Analysis

Constructs and constraints can be inferred from existing structural patterns. For instance, names can suggest roles (identifier, foreign key), data types or relationships between data.

Program Analysis

Through pattern seeking, one can find instances of programming *clichés* that suggest implicit constraints such as identifiers, foreign keys or exclusive fields.

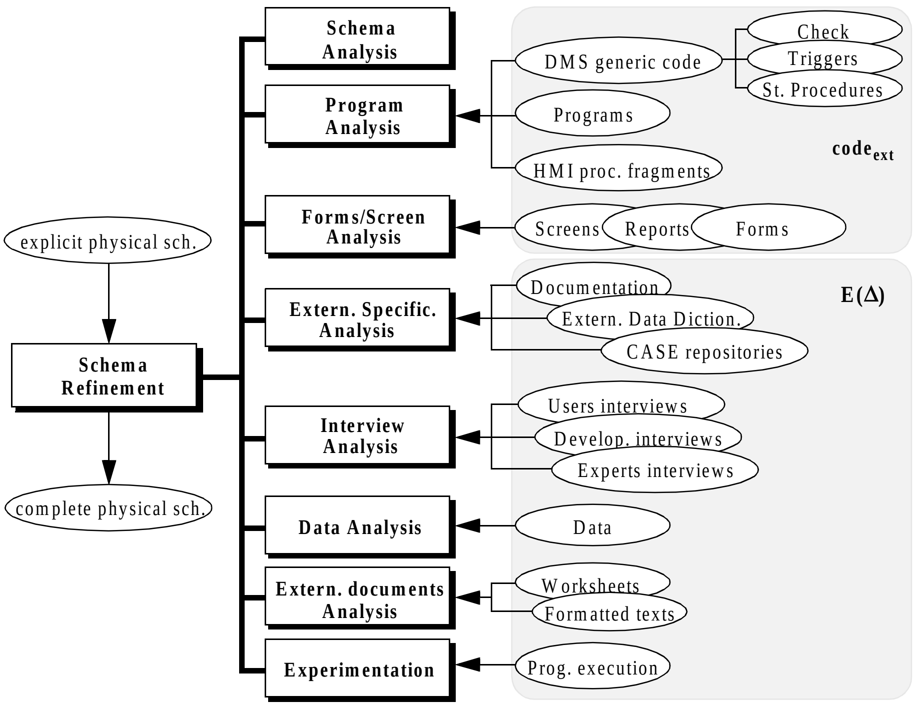


Figure 5. The main information sources and techniques used in the Schema Refinement process.

Frequently, such patterns will be found just before storing data, in data validation sections.

Dataflow Analysis

Analyzing the data flow among the variables of a program can give valuable information on semantic or structural properties of these variables. For instance, variables that share common values at run time should have similar structures and meaning.

Program Slicing

This technique consists in computing the sequence of statements that contribute to the state of an object at a program point, therefore reducing the search space of, e.g., a specific programming pattern. *Example*: the program slice of a record computed at a writing statement should include all the data validation statements.

Data Analysis

This technique aims at finding relationships and patterns in file and database contents. In many cases, it will be used to confirm or disprove hypotheses. *Examples*: finding uniqueness constraints, testing candidate foreign keys, determining enumerated domains.

4.3. Finding implicit foreign keys

In this section, we apply some of these techniques to the

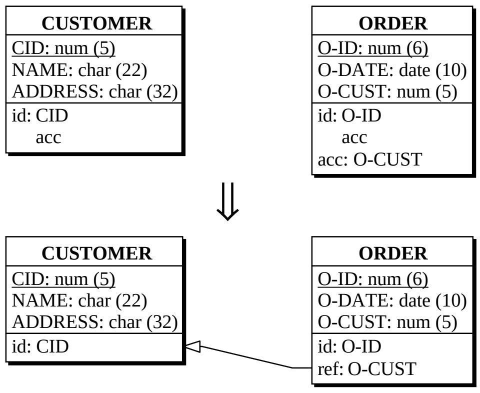


Figure 6. Eliciting an implicit foreign key.

elicitation of implicit foreign keys[3]. The example to which we will apply these techniques is sketched in Figure 6.

Schema Analysis

From structural properties of the schema, we can guess some relationships between fields ORDER.O-CUST and CUSTOMER.CID. In particular,

- the name O-CUST suggests that of CUSTOMER;
- O-CUST and the identifier of CUSTOMER (CID) have

3 Implicit FK can be found in any databases, including, surprisingly, SQL, IMS and CODASYL ones. Moreover actual databases include non standard foreign keys as well, such as multivalued, alternate, multi-target, conditional, overlapping, embedded, or transitive FK.

the same type and the same length;

- O-CUST is supported by an index, which is frequent for foreign keys.

Program analysis

The following pseudo-code section represents a typical access pattern that uses a foreign key to get dependent records.

```
read-first ORDER(O-CUST=CUSTOMER.CID);
while found do
  process ORDER;
  read-next ORDER(O-CUST=CUSTOMER.CID)
end-while;
```

Dataflow analysis

Considering the following COBOL program:

```
DATA DIVISION.
  FILE SECTION.
  FD F-CUSTOMER.
    01 CUSTOMER.
      02 CID pic 9(5).
      02 NAME pic X(22).
      02 ADDRESS pic X(32).
  FD F-ORDER.
    01 ORDER.
      02 O-ID pic 9(6).
      02 O-DATE pic 9(8).
      02 O-CUST pic 9(5).
  WORKING-STORAGE SECTION.
    01 C pic 9(5).
    01 OI pic 9(6).
PROCEDURE DIVISION.
  ...
  display "Enter order number ".
  accept OI.
  move 0 to IND.
  call "SET-FILE" using OI, IND.
  read F-ORDER invalid key go to ERROR-1.
  ...
  if IND > 0 then move O-CUST of ORDER to C.
  ...
  if C = CID of CUSTOMER then
     read F-CUSTOMER invalid key go to ERROR-2.
  ...
```

we can build the data flow diagram of Figure 7. It suggests that the fields ORDER.O-CUST and identifier CUSTOMER.CID can have the same value at some execution time point, what suggests that O-CUST can be a foreign key.

Data Analysis

Let us suppose that we suspect ORDER.O-CUST to be a foreign key to CUSTOMER. We could check the validity of this hypothesis by evaluating the following query[4]. The result should be 0 for O-CUST to be a foreign key.

```
select count(*) from   ORDER
where  O-CUST not in (select CID from CUSTOMER)
```

4 To simplify the presentation, and without loss of generality, we write this query in SQL instead of COBOL.

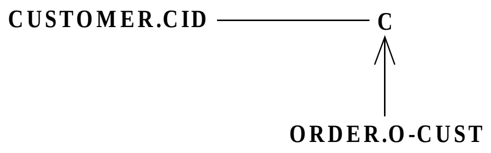


Figure 7. A dataflow diagram.

5. Data Structure Conceptualization

This step extracts a conceptual schema from the logical schema made available at completion of the Data Structure Extraction phase. In short, it consists in interpreting the technical structures in terms of the underlying semantics. After an optional preparation phase, which mainly consists in removing non semantic constructs (such as dead parts and non data constructs) and renaming data objects, two main phases have to be carried out, namely basic conceptualization and normalization (Figure 8).

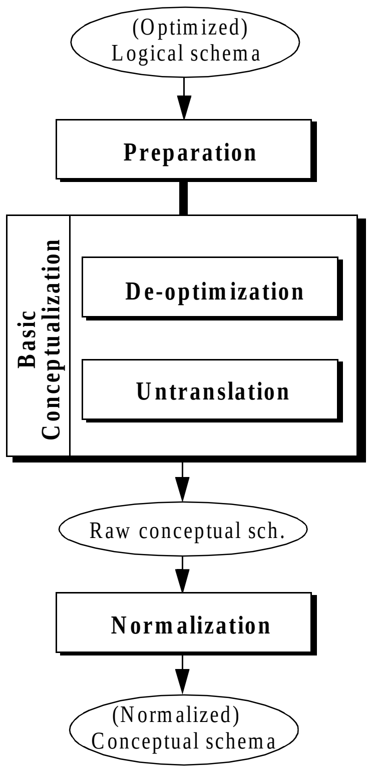


Figure 8. The main processes of the Data Structure Conceptualization phase.

Basic Conceptualization

This process produces a raw conceptual schema through de-optimization and untranslation. It must be noted that the latter must not be seen as two independent processes, but rather as two classes of specific problems, reasoning and techniques. In actual DBRE projects, they form a unique process.

De-optimization

This activity consists in identifying and processing optimization constructs. These constructs were incorporated in the schema for performance reasons. They bear no semantics and can be discarded or transformed without loss of information. Merging and splitting record types, denormalizing, removing redundant structures are the main techniques that are used to clean a schema from its optimization aspects.

Untranslation

Untranslating a logical schema consists in retrieving the source conceptual structure of each implementation construct. One of the best examples is the replacement of a foreign key with the intended relationship type.

Normalization

The raw conceptual schema is reshaped to gain readability, minimality and expressiveness. This process is quite similar to that of forward database design [3].

6. Data Reverse Engineering Tools

There is no commercial specific Data-oriented CARE[5] tools so far. This is not a drawback anyway, since the reverse engineering process shares much, in terms of models and techniques, with standard engineering activities. Unfortunately only limited DBRE functions can be found in current CASE tools such as Power-Designer, AMC-Designor, Rose or Designer 2000. At best, they include elementary parsers for SQL DB, foreign key elicitation under very strong assumptions (PK and FK having same names and types) and some primitive standard foreign key transformations. None can cope with complex reverse engineering projects.

The DB-MAIN CASE environment

This set of tools, developed since 1993, is intended to support the major database application engineering activities, including reverse engineering. Besides standard functions that can be found in most data-oriented CASE environment, it includes several processors dedicated to the processes described in this paper. One of its strongest features is its extensibility, among others through the Voyager 2 language, which allows analysts to develop their own processors to plug into the environment. We will briefly describe some of the functions and processors. More detail can be found in [19] and [14].

Support for the data structure extraction process

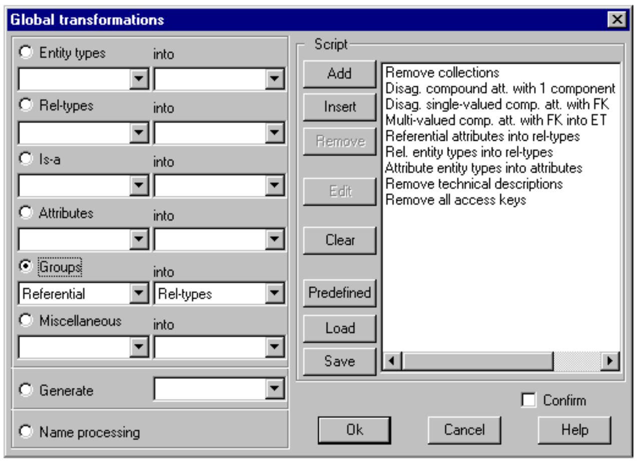
Code parsers are available for SQL, COBOL, CODASYL, RPG and IMS source programs; other parsers can be developed in Voyager 2.

Programs can be analyzed through specific tools: an interactive and programmable text analyzer, a dataflow and dependency graph builder and analyzer, a program slicer.

Schema can be processed through a name processor, a programmable schema analyzer, a programmable foreign key discovery assistant and a schema integrator.

Support for the data structure conceptualization process

This process is based on schema transformation techniques. The environment includes a toolbox of about 30 semantics-preserving schema transformations, a name processor, a conceptual schema integrator and a programmable schema transformation assistant (Figure 9).



Groups: Referential into Rel-types

Remove collections
Disag. compound att. with 1 component
Disag. single-valued comp. att. with FK
Multi-valued comp. att. with FK into ET
Referential attributes into rel-types
Rel. entity types into rel-types
Attribute entity types into attributes
Remove technical descriptions
Remove all access keys

Figure 9. The Global transformation assistant. The script shown is intended to conceptualize COBOL data structures.

7. Effort Quantification

Measuring the effort that will be necessary to complete a reverse engineering project is a most challenging problem. Indeed, this effort directly depends on the goal of the process, on the complexity of the final schema and on the quality of the information sources. These properties can only be estimated at the completion of the project. Anyway, we can suggest some figures for a typical database made up of 100 files/tables and about 2,000 fields/columns[6]. We consider three different goals and we evaluate the effort as the time one skilled analyst needs to complete the project.

1. *Quality assessment*. The objective is to evaluate the quality of a database as a measure of the quality of the whole application. No completeness is required, so that, according to [6], such a task can be completed in **1 week**.

5 Computer-Aided Reverse Engineering

6 By the way, the current champion seems to be SAP internal database, with 16,000-30,000 tables and 200,000+ columns.

2. *Data conversion*. This task, that consists in migrating the contents of a data base to another database, possibly with another schema, is more complex, since it requires the understanding of the structure and of a part of the semantics of the data. However, the elicitation of the integrity constraints can be skipped. We can evaluate the effort to **6 weeks**.
3. *Reengineering*. The objective is to build a new system that is functionally equivalent to the old one, but that is given a better architecture. Now, the whole semantics, including all the integrity constraints, must be elicited[7]. The effort can be evaluated to **26 weeks**[8].

The effort also depends on the quality of the source. In the following table, we consider three situations, and we compare the effort (the figures are tentative).

situation	effort
Well documented, normalized relational DB	C
Undocumented, poorly designed IMS DB	5 x C
Undocumented, poorly designed COBOL files	10 x C

8. Conclusions

Considering the scope of this paper, the best conclusion should discuss the state of the art and draw some recommendations on what remains to be done.

So, what could be considered as available at present time? We can agree on the fact that most problems have been identified, and that many elicitation techniques and heuristics have been proposed in the scientific literature. Many popular CASE tools now include (limited) data reverse engineering functions. Powerful analysis tools have been developed in laboratories, but they have not been integrated in more general environments (that is the case for program understanding tools for instance).

Of course, much remains to be done. On the *psychological* side, sensitizing actions among practitioners are needed. The message must be: data reverse engineering is useful and can be carried out. On the other hand, results are not guaranteed to be 100% complete and the process can prove resource intensive, not only in time and money, but also in preliminary training. As a consequence, methodological material must be developed, such as technical textbooks, tutorials and seminars.

7 Example: recovering about 200 implicit foreign keys in a Part inventory IMS database took 60 working days (with DB-MAIN v3).

8 This ratio has been observed in a project consisting in rebuilding the complete logical schema of a 80-file COBOL application with the help of DB-MAIN v5.

Despite the number of elicitation techniques available, many still need to be evaluated on real applications, and particularly on large-size ones to prove their scalability. They must be refined in order to reduce their noise and silence scores.

Current commercial tools are poor and not scalable with this respect. Proprietary tools are powerful but unavailable. Effort need to be done to build both popular **and** powerful CARE tools.

DRE is an important part of system reverse engineering, but it is only a part. Its integration into large scope reengineering methodologies and tools has not been addressed so far. In particular, developing techniques for reengineering legacy systems into distributed components architectures still need much work, despite some local success records.

Most research work has addressed RDB reverse engineering (Figure 10). However, most needs concern less attractive but more critical problems: redocumenting old COBOL, RPG or Business Basic applications, based on standard files or IMS/CODASYL databases.

Database type	% papers
Standard files	**17.5**
Hierarchical databases (e.g., IMS)	12.5
Shallow databases (Total and Image)	2.5
Network databases (mainly CODASYL)	10.0
Relational databases	**52.5**
OO databases	5.0

Figure 10. Distribution of 40 recent research publications according to the data model.

9. Bibliography

[1] Aiken, P. 1996. *Data Reverse Engineering, Slaying the Legacy Dragon*, McGraw-Hill.

[2] Andersson, M. 1994. Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering, in *Proc. of the 13th Int. Conf. on ER Approach*, Springer-Verlag

[3] Batini, C., Ceri, S., Navathe, S. 1992. *Conceptual Database Design - An Entity-Relationship Approach*, Benjamin/Cummings

[4] Blaha, M.R., Premerlani, W., J. 1995. Observed Idiosyncrasies of Relational Database designs, in *Proc. of the 2nd IEEE Working Conf. on Reverse Engineering*, Toronto, July 1995, IEEE Computer Society Press

[5] Blaha, M., 1998. On Reverse Engineering of Vendor Databases, in *Proc. of the 5th IEEE Working Conf. on Reverse Engineering*, Honolulu, October 1998, IEEE Computer Society Press

[6] Blaha, M., The Case for Reverse Engineering, *IEEE IT Professional*, March-April, 1999

[7] Brodie, M., Stonebraker, M. 1995. *Migrating Legacy Systems*, Morgan Kaufmann.

[8] Campbell, L., Halpin, T. 1994. The reverse engineering of relational databases, in *Proc. 6th Int. Work. on CASE* (1994).

[9] Casanova, M., A., Amaral De Sa. 1984. Mapping uninterpreted Schemes into Entity-Relationship diagrams: two applications to conceptual schema design, in *IBM J. Res. & Develop.*, 28(1)

[10] Chiang, R., H., Barron, T., M., Storey, V., C. 1996. A framework for the design and evaluation of reverse engineering methods for relational databases, *Data and Knowledge Engineering*, Vol. 21, No. 1 (December 1996)

[11] Comyn-Wattiau, I., Akoka, J. 1996. Reverse Engineering of Relational Database Physical Schema, in *Proc. 15th Int. Conf. on Conceptual Modeling (ERA)*, Cottbus, LNCS 1157, Springer Verlag

[12] Davis, K., H., Arora, A., K. 1988. Converting a Relational Database model to an Entity Relationship Model, in *Proc. of Entity-Relationship Approach: a Bridge to the User*, 1988

[13] Davis, K., H. 1996. Combining a Flexible Data Model and Phase Schema Translation in Data Model Reverse Engineering, in *Proc. of the IEEE Working Conf. on Reverse Engineering*, Monterey, Nov. 1996, IEEE Computer Society Press

[14] *The DB-MAIN Database EngineeringCASE Tool (version 5) - Functions Overview*, DB-MAIN Technical manual, December 1999, Institut d'informatique, FUNDP; available at `http://www. info.fundp.ac.be /~dbm/ references.html`

[15] Edwards, H., M., Munro, M. 1995. Deriving a Logical Model for a System Using Recast Method, in *Proc. of the 2nd IEEE WC on Reverse Engineering*, Toronto, IEEE Computer Society Press

[16] Fonkam, M., M., Gray, W., A. 1992. An approach to Eliciting the Semantics of Relational Databases, in *Proc. of 4th Int. Conf. on Advance Information Systems Engineering* - CAiSE'92, pp. 463-480, Springer-Verlag, 1992

[17] Hainaut, J-L., Chandelon M., Tonneau C., Joris M. 1993a. Contribution to a Theory of Database Reverse Engineering, in *Proc. of the IEEE Working Conf. on Reverse Engineering*, Baltimore, May 1993

[18] Hainaut, J-L, Chandelon M., Tonneau C., Joris M. 1993b. Transformational techniques for database reverse engineering, in *Proc. of the 12th Int. Conf. on ER Approach*, Arlington-Dallas, E/R Institute and Springer-Verlag, LNCS

[19] Hainaut, J-L, Roland, D., Hick J-M., Henrard, J., Englebert, V. 1996. Database Reverse Engineering: from Requirements to CARE tools, *Journal of Automated Software Engineering*, Vol. 3, No. 1 (1996).

[20] Hainaut, J.-L., *Database Reverse Engineering*, DB-MAIN Research report, Namur, 1999 (133 p.); available at `http://www.info.fundp.ac.be/~dbm /references.html`

[21] Johannesson, P., Kalman, K. 1990. A Method for Translating Relational Schemas into Conceptual Schemas, in *Proc. of the 8th ERA conference*, Toronto, North-Holland,

[22] Joris, M., Van Hoe, R., Hainaut, J-L., Chandelon M., Tonneau C., Bodart F. et al. 1992. PHENIX: methods and tools for database reverse engineering, in *Proc. 5th Int. Conf. on Software Engineering and Applications*, Toulouse, December 1992, EC2 Publish.

[23] Kalman, K. 1991. Implementation and critique of an algorithm which maps a relational database to a conceptual model, in *Proc. of the CAiSE•91 conference*.

[24] Markowitz, K., M., Makowsky, J., A. 1990. Identifying Extended Entity-Relationship Object Structures in Relational Schemas, *IEEE Trans. on Software Engineering*, Vol. 16, No. 8

[25] Navathe, S., B., Awong, A. 1988. Abstracting Relational and Hierarchical Data with a Semantic Data Model, in *Proc. of Entity-Relationship Approach: a Bridge to the User*

[26] Petit, J-M., Kouloumdjian, J., Bouliaut, J-F., Toumani, F. 1994. Using Queries to Improve Database Reverse Engineering, in *Proc. of the 13th Int. Conf. on ER Approach*, Manchester, Springer-Verlag

[27] Premerlani, W., J., Blaha, M. R. 1993. An Approach for Reverse Engineering of Relational Databases, in *Proc. of the IEEE Working Conf. on Reverse Engineering*, IEEE Computer Society Press

[28] Signore, O, Loffredo, M., Gregori, M., Cima, M. 1994. Reconstruction of ER Schema from Database Applications: a Cognitive Approach, in Proc. of the *13th Int. Conf. on ER Approach*, Manchester, Springer-Verlag

[29] Sneed, H.S. 1996. Object-Oriented COBOL Recycling. *Proc. of the 3rd IEEE Working Conf. on Reverse Engineering*, Monterey, Nov. 1996, IEEE Computer Society Press.

[30] Thiran, Ph., Hainaut, J-L., Bodart, S., Deflorenne, A. 1998. Interoperation of Independent, Heterogeneous and Distributed Databases. Methodology and CASE support: the InterDB Approach. in *Proc. of the Int. Conference on Cooperative Information Systems* (CoopIS-98), New-York, August 1998, IEEE Computer Society Press