CONTRIBUTION TO A THEORY OF DATABASE REVERSE ENGINEERING

J-L. Hainaut, M. Chandelon, C. Tonneau, M. Joris

Institut d'Informatique, University of Namur
rue Grandgagnage, 21 - B-5000 Namur (Belgium)
email : jlh@info.fundp.ac.be - fax : +32 81-724967

Abstract

The paper proposes both a general framework and specific techniques for file and database reverse engineering, i.e. recovering its conceptual schema. The framework relies on a process/product model that matches formal as well as empirical design procedures. Based on the analysis of database design processes, two major phases are defined, namely *Data structure extraction* and *Data structure conceptualization*. For each of them, a set of activities is proposed. Most of these activities can be described as transformation and integration of specifications.

1. INTRODUCTION

Considering the proliferation of comprehensive database design methods and database-oriented CASE tools, database forward engineering appears much more mature than its processing counterpart. This fact is easy to explain. The domain is more restricted and offers less freedom while the requirements are better understood. In addition (and probably consequently) a fairly comprehensive and realistic database theory has now been made available for practitioners, a fact that cannot be claimed for most SE formal approaches.

An increasing number of CASE tools offer some database reverse engineering (DBRE) functionalities (let us mention the Bachman re-engineering toolset only). Though they ignore many of the most difficult aspects of the problem, these tools provide their users with invaluable help to carry out DBRE more effectively [R1].

Surprisingly enough, DBRE has raised little interest in the DB scientific community. By browsing major information sources such as ACM TODS, VLDB and ER conferences proceedings, or Knowledge & Data Engineering, one can hardly collect twenty papers more or less related with DBRE. Let us mention some references :

- RE of standard files : [C1], [N2], [D1]
- RE of IMS databases : [N1], [W1]
- RE of CODASYL databases : [B2]
- RE of relational databases : [C2], [N1], [D2], [S2], [F1]

Most of these studies appear to be limited in scope (most often dedicated to one data model), and are generally based on severely unrealistic assumptions on the quality and completeness of the data structures to reverse engineer. For instance, they suppose that,

- all the conceptual specifications have been translated into data structures and constraints,
- the translation is rather straightforward (no tricky representations),
- the schema has not been deeply restructured for performance objectives or for any other requirements,
- a complete physical schema of the data is available,
- names have been chosen rationally (e.g. a foreign key and the referenced primary key have the same name).

The situation should rapidly evolve, as testified for instance by [B2], the first popular reference that includes several sections dedicated to DBRE. However, a general theory supporting DBRE of real applications has yet to be built. This paper is a contribution to this future theory.

Roughly speaking, reverse engineering (RE) a piece of software consists in reconstructing its functional and technical documentation, starting mainly from the source text of the programs [I1] [H3]. Recovering these specifications is generally intended to convert, restructure, maintain or extend old applications. It is also required when developing a Data Administration function.

In short, RE tries to answer the question : *what are possible specifications of this implementation*[1]. The problem is particularly complex with old and ill-designed applications. In this case, not only no decent documentation (if any) can be relied on, but the lack of systematic methodologies for

[1] A secondary, but also important question is often *how the implementation got to be what it is*, i.e. elicitating the functional and technical requirements together with the reasonings through which they have been satisfied.

designing and maintaining them have led to tricky and obscure code. Therefore, RE has long been recognized as a complex, painful and prone-to-failure activity, in such a way that it is simply not undertaken most of the time, leaving huge amounts of invaluable knowledge buried in the programs, and therefore definitely lost.

In *data-oriented applications*, the complexity can be broken down by considering that the files or databases can be reverse engineered (almost) independently of the procedural parts.

This proposition to split the problem in this way can be supported by the following arguments :

- the semantic distance between the so-called conceptual specifications[2] and the physical implementation is most often narrower for data than for procedural parts;
- the data are generally the most stable part of applications;
- even in very old applications, the *semantic structures* that underlie the file structures are mainly procedure-independent (though the *physical structure* is highly procedure-dependent);
- reverse engineering the procedural part of an application is generally easier when the semantic structure of the data has been elicited.

Therefore, concentrating on reverse engineering the data components of the application first can be more successful than trying to cope with the whole application. Though RE data structures still is a complex task, it appears that the current state of the art provides us with sufficiently powerful concepts and techniques to make this enterprise more realistic.

This paper analyzes the problems of reverse engineering data structures in data-oriented applications. It proposes a systematic and general framework, based on formal techniques and heuristics, for solving the problem of recovering a possible conceptual schema of an existing database. The reader is supposed to be somewhat acquainted with database concepts; reference [E1] is recommended otherwise.

2. DATABASE DESIGN REVISITED

Though database design has been one of the major theoretical and applied research domain in software engineering in the last two decades, and though it can be considered as a now fairly well mastered problem, we are faced here with files and database[3] that have not been built according to well structured methodologies. Tackling the reverse engineering of a database needs a deep understanding of the forward process, i.e. database design, not only according to standard and well formalized methodologies, but above all when no such rigourous methods have been followed. In other words, we intend to grasp the mental rules that make the intuitive behaviour of practitioners. Our approach should not be **normative**, as in forward engineering, where practitioners are told how to work, but rather **descriptive**, since we have to find out how they have worked.

The analysis is based on the following assumptions about database design :

- a database must satisfy a limited set of **requirements** such as : correctness, user acceptance, time performance, space performance, availability, reliability, compliance with a data manager, hardware constraints, security, compatibility with organizational constraints, conformity with a methodological standard, etc. Satisfying each kind of requirements constitutes an identifiable **design problem**.
- solving each of these design problems is the objective of an identifiable **design process**; each process produces **design products**, i.e. specifications of the solution at a given level of abstraction;
- designing any database, whatever the methodology (or absence thereof), requires to solve the same collection of problems; therefore, designing a database consists in carrying out a **limited set of processes**; in some unstructured design approaches, some processes can be by-passed while in others several processes are conducted in parallel.
- each design process is based on specific **concepts**, **techniques** and **reasonings**;
- each design process can be expressed as a **transformation** applied on input products and yielding output products; the transformation consists in adding constructs to the input specifications, removing some constructs, or changing the format of these specifications.

These assumptions are the basis for a *generic model of database design activities* that gives us some useful hints on how to conduct the reverse engineering of an existing database. Indeed, it suggests to **search the description of the database for traces of each specific design process**, instead of trying to rebuild the conceptual schema in a

[2] In the database realm, the abstract, implementation-independent specification of a database is called its *conceptual schema*.

[3] From now on, the term *database* will encompass any permanent, structured, data collection on secondary storage, including standard files.

single step. In addition, it allows for the specification of both non standard and empirical design practices.

In the limited scope of this paper, we shall consider a simplified generic model of database design. Figure 2.1 depicts the organization of the main design processes and the design products. According to this model, the processes that can be carried out and the products they transform and produce are as follows :

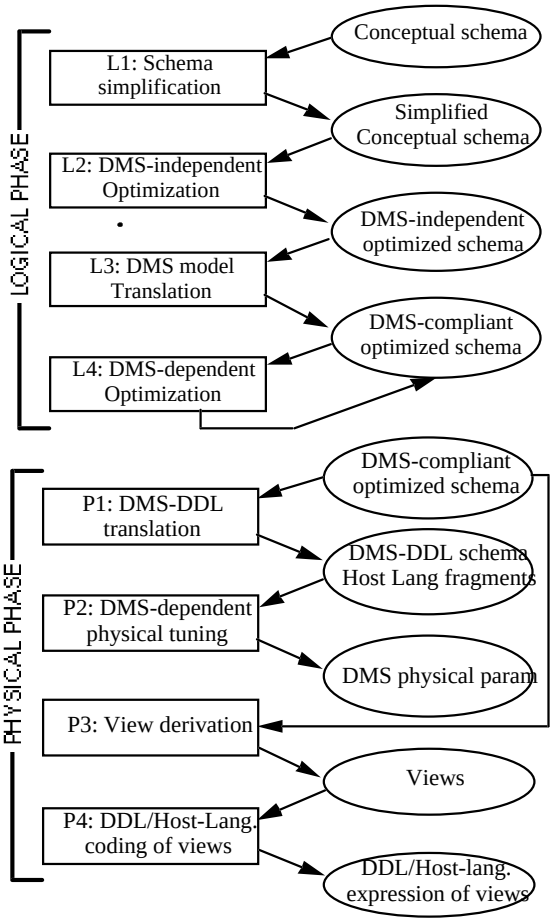


Figure 2.1 - Some important design processes and products of database design. The processes have been classified into the logical and physical phases.

Conceptual phase : user requirements are collected, analyzed and formalized into a conceptual schema. Since it has no impact on reverse engineering, this activity is not shown in figure 2.1.

Logical phase : the schema can be expressed (L1) into a simpler model (e.g. Bachman's model), better suited for optimization reasonings; it can be optimized independently of the target DBMS (L2); it is then translated according to the target data model (L3); at this stage, it can be further optimized according to DMS-dependent rules (L4).

Physical phase : the logical schema is expressed according to the DDL[4] of the DMS and to procedural languages (P1); its physical parameters are set through DMS-dependent physical tuning (P2); the view needed by the application programs are derived (P3) and expressed partly into the view DDL, and partly into the host language.

3. WHAT MAKES *DBRE* SO DIFFICULT ?

The final, *executable description* of the database, i.e. the *DDL/Host language* expression of schemas and views, can be seen as the result of a chain of transformations that have ***degraded*** the origin conceptual schema. Let's reexamine each process of the generic model of database design as far as it introduces some degradation into the conceptual schema.

Process L1 : the conceptual specifications are preserved, but they are expressed with poorer structures, leading to a less concise and readable schema. For instance, n-ary relationship types have been transformed into binary ones, multivalued attributes have been reduced to single-valued ones, or IS-A links are transformed into one-to-one relationship types.

Process L2 : the schema can be restructured according to design requirements concerning access time, distribution, data volume, availability, etc. The conceptual specifications are preserved, but the schema is obscured due to non-semantic restructuration, such as structure splitting or merging, denormalization or structural redundancy.

Process L3 : the data structures are transformed in order to make them compliant with the model of the target DMS. Generally, this process deeply changes the appearance of the schema in such a way that the latter is still less readable. For instance, in standard files (resp. relational DBMS) many-to-many relationship types are transformed into record types (tables) while many-to-one relationship types are transformed into reference fields (foreign key). In a CODASYL schema, a secondary identifier is represented by an indexed singular set. In a TOTAL or IMAGE database, a one-to-many relationship type between two major entity types is translated into a detail record type. Frequently, names have to be converted due to the syntactical limitations of the DMS or host languages.

On the other hand, due to the limited semantic expressiveness of older (and even current) DMS, this

4 DMS stands for Data Management System (including Database Management Systems or DBMS. The Data Description Language (DDL) is the language of the DMS through which the data structures are declared or defined.

translation is seldom complete. It produces two subsets of specifications : the first one being strictly DMS-compliant while the second one includes all the specifications that cannot be taken in charge by the DMS. For instance, referential constraints cannot be processed by standard file managers. In principle, the union of these subsets includes the conceptual specifications.

Process L4 : the schema is restructured to match design criteria such as access time. This restructuring depends on the specific behaviour of the DMS. Just like process L2, it makes the schema less readable. The semantic contents of the DMS-compliant structure are preserved.

Process P1 : only the first subset of specifications collecting strictly DMS-compliant structures can be translated into the DDL of the DMS. The discarded specifications should be either ignored, or translated into languages, systems and procedures that are out of control of the DMS (e.g. host language, user interface manager, human procedures, etc). From now on, the schema of the database generally represents a strict subset only of the conceptual specifications.

Another phenomenon must be pointed out, namely **structure hiding**. When translating a data structure into DMS-DDL, the designer (or programmer) may choose to hide some information, leaving to the host language the duty to recover this information. The most widespread example consists in declaring some subset of fields (or even all the fields) of a record type (or segment, or table) as a single, unstructured, field; recovering the hidden structure can be made by storing the unstructured field values into a host program variable that has been given the adequate structure.

Finally, let's observe that the DMS-DDL schema is not always materialized. Many standard file managers, for instance, doesn't offer any central way to record the structure of files, e.g. in a data dictionary.

```
Initial record structure
     01 CUSTOMER
        02 C-KEY  pic X(14)
           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)

Coded record structure
     01 CUSTOMER
         02 C-KEY  pic X(14)
         02 filler pic X(57)
```

Figure 3.1 - An example of structure hiding. The decomposition of both the key part and the data part are replaced by anonymous data structures in the actual coded structure. Though it can simplify data description and data usage, this frequent technique makes data structure considerably more complex to recover.

Process P2 : works at the internal level, and doesn't modify the schema as it is seen by the programmer.

Process P3 : each view is dedicated to a limited set of application programs (or users). In principle, the set of the views cover all the data structures they derive from.

Process P4 : this process is similar to P1. Each view is expressed into a mixture of DDL texts, host language procedures, screen/report specifications, etc. Here too, the principle of structure hiding can be applied heavily, specially in case of standard file managers. Figure 3.1 is an illustration of this practice. Figure 3.2 shows a common way to translate lost referential integrity constraints into the procedural part of the program.

```
fd F-CUST;
record is CUSTOMER.
01 CUSTOMER.
   02 CNUM pic X(14).
   02 CDATA pic X(57).

fd F-ORD;
record is ORDER.
01 ORDER.
   02 ONUM pic 9(8).
   02 O-DATE pic X(12).
   02 CUST pic X(14).

working storage section
01 CN pic X(14).
01 C.
   02 CNUM pic X(14).
   02 filler pic X(57).
01 O.
   02 ONUM pic 9(8).
   02 O-DATE pic X(12).
   02 CUST pic X(14).

procedure division (in pseudo-code)
    ...
    read CUSTOMER(CNUM = X) into C
    if found(C) then
          O.ONUM := ...
          O.DATE := ...
          O.CUST := C.CNUM
          write O into ORDER
    endif
    ...
```

Figure 3.2 - Expressing non-DMS structural parts (here a referential constraint) by procedural statements. The latter, sometimes centralized into a single file management module, check the non-violation of constraints before data insert, update and delete. These sections are often written according to regular patterns such as that presented here.

Observation : in poor DMS, like most standard file managers, **the result of process P4 is the only information that is still available** about the implemented data structures. This fact ***plus*** structure hiding make the reverse engineering of standard file an exercise particularly challenging.

Let's draw some conclusions from this analysis :

- producing an executable schema of a database can be seen as the step-by-step transformation of its conceptual schema.
- each transformation introduces some sort of degradation in the schema that makes it less complete, simple, intuitive and readable.
- the kind of degradation depends on the objective of the design process.

4. GROSS ARCHITECTURE OF A *DBRE* METHODOLOGY

Let's first observe that reverse engineering a database is concerned with the design decisions that have been taken during the logical and physical phases only. The proposed approach is based on playing backward these two phases, starting from the results of the latter one. Grossly speaking, the RE analyst is faced with the following problem :

- given the DDL/host language expression of existing data structures (global schema and/or views),
- given known operational requirements (e.g. the DMS, performance requirements, etc),

Ø find a possible conceptual schema that could lead to these data structures.

The process can be split into two main phases, rather independent from each other :

Ø retrieve the existing data structures from their DDL/host language expression, and

Ø retrieve a possible conceptual schema that defines the semantics that underlies these data structures.

The first process is the reverse of the physical phase, and has been named *Data structure extraction*. The second one is the reverse of the logical phase, and has been named *Data structure conceptualization*. In a first approach, we can consider that they are conducted sequentially. According to the analysis developed above, the gross organization of the method can be sketched as in Figure 4.1.

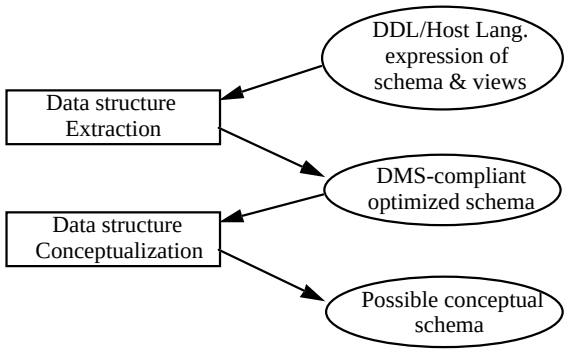


Figure 4.1 - Gross organization of database reverse engineering activity.

In low level DMS (e.g. standard file managers), the starting point generally is made up of the DDL/Host language expression of the views. In higher level DMS, the input information generally consists of the DDL expression of the global schema. In both cases, these expressions should be augmented with the translation of discarded specifications. In some organizations, the data structure descriptions are available through a data dictionary system, making the first process a bit easier.

5. DATA STRUCTURE MODELLING FOR *DBRE*

Before going into deeper details, we need supporting models to express data structures at the different levels of abstraction. Due to the great flexibility that must be provided in conducting the RE processes, we propose a unique, layered, data model. Through this unique model, both a DMS-compliant optimized schema, and a conceptual schema (together with schemas at all the intermediate levels) will be expressed in a uniform way. This feature will allow us to use very general transformational processes, whatever the abstract level of the products involved, and to describe a great variety of reverse engineering behaviours or strategies[5].

This model is based on the Entity-Relationship model [C3] [B2]. It is intended first to express **conceptual structures**[6] by means of the following concepts :

- entity type (ET) and n-ary relationship type (RT),
- ET- and RT-attribute,
- ET-, RT- and attribute identifier,

---

5 For instance those which allow some parts of the schema being conceptualized while others are still at the physical level.

6 This model is a bit more complex than usual. This is due to the semantic richness of the technical constructs that are available in operational DMS, and that are not always used in strict top-down database design methodologies.

- integrity constraints, such as inclusion, exclusion, redundancy, implication,
- etc.

In logical and physical schemas, these concepts are to describe technical structures as well. For instance, an entity type will describe a relational table in an SQL database, a record type in a COBOL file or in a CODASYL database and a segment type in an IMS database. Similarly, an attribute will be the abstraction of a field or of a column, and a relationship type will be the abstraction of a CODASYL set type, IMS parent/child relationship, TOTAL / IMAGE path or ADABAS file coupling.

In addition, this model has been given constructs allowing the definition of files, access paths, access keys (abstraction of indices, calc keys, etc), field redefinition, physical position, etc, in order to express more technical data structures, such as those found in DMS-compliant optimized schemas for instance, and to support reasoning at the physical level. Finally, some procedural concepts that are relevant when dealing with file and record processing are also described in the model. Let's only mention the notions of *application*, *source file*, *module*, *variable*, *transfer instruction* (`MOVE`, `CALL USING`, `READ INTO`, etc), *condition name*, etc.

6. DATA STRUCTURE EXTRACTION

The data structures of the DMS/Host language optimized schema, are found out by analysing the source code of the schemas and programs. The result of this process is a (hopefully) complete, formalized, model of the physical data structures, as they are perceived by users and programmers, according to the DMS model. For some DMS, the analysis is rather straightforward (e.g. CODASYL, IMS or relational), while for others, it requires considerable work and knowledge (e.g. standard files).

The problem is twofold :

- retrieve the explicit and implicit (hidden) data structures that are under the control of the DMS,
- retrieve the discarded specifications (generally integrity constraints ignored by the DMS).

6.1 Retrieving the DMS data structures

The difficulty of the first problem is dependent on the DMS. In true DBMS, the description of the global DMS-compliant schema is generally available, either as a DDL text, or in a data dictionary (e.g. *system catalog tables*). In lower-level DMS, such as standard file managers, the problem can be much more complex. Indeed, the global schema, generally file and record structures, is not under the control of the DMS, and is only available in the (generally lost or obsolete) documentation of the application. The only description available is made up of file and record descriptions included in the source programs. Since these descriptions can be, and generally are, partial descriptions only, reverse engineering the data structures used by a source program produces one view of the global schema only.

Two difficulties must be solved when carrying out this process : (1) an application program uses several files, a subset only of which concern the database; print, output, temporary, update or sort files must be recognized and discarded[7]; (2) due to the structure hiding habits of many programmers, the actual structure of a record type or of a field can only be elicited by analyzing how and where records/fields are used; this means for example examining the control flow of the procedural parts of the program, as illustrated in figure 6.1, or analyzing the entry forms and the output reports/forms.

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

working storage section
   01 IN-CUST
      02 ZIP-CODE pic X(8)
      02 SER-NUM  pic 9(6)
      02 C-DATA   pic X(57)

   01 EXPLODE1
      02 NAME     pic X(15)
      02 ADDRESS  pic X(30)
      02 ACCOUNT  pic 9(12)

procedure division
   ...
 read CUSTOMER into IN-CUST.
   ...
 move C-DATA of IN-CUST into EXPLODE1.
   ...
```

Figure 6.1 - Hidden structure elicitation. The original record decomposition is recovered through data flow analysis.

6.2 Retrieving discarded specifications

This process is mainly based on the analysis of the procedural parts of the applications, be they program sections or triggered actions associated with user interface forms or with database events. These procedures mainly check integrity constraints and compute derived data. Their structure is generally simple and standard, and can be recognized easily, provided we can locate them in the huge

[7] Unless these files may give hints on the schema when they store data from the database.

amount of source code. Among the main constraints the texts have to be searched for, let's mention the referential constraints (in standard files and in relational databases), the identifiers (in sequential files and tables, in network databases), one-to-one relationship types (when many-to-one only are available), exclusive structures and redundancy constraints.

In some circumstances, the data themselves have to be analyzed. Indeed, evidence of uniqueness properties (identifiers), referential integrity, fine-grained field decomposition or value domain, can be found out by careful examination of the data.

6.3 Integrating views

When the previous processes have recovered views of the data structures only, e.g. in low-level DMS, two strategies are possible :

- integrate these views to obtain a global schema, then conceptualize it;
- conceptualize each view, then integrate the conceptual views obtained in this way.

The first strategy allows to reduce as early as possible the number of schemas to be conceptualized (collecting more than 100 views is not uncommon in large applications). In addition, powerful heuristics can be used at this level, based on position and length of data fields in the records, multi-record type files or the use of compile-time statements (e.g. COBOL `copy` or C `include`). However, integrating physical views can prove complex when they include many performance-oriented constructs such as redundancies, denormalized structures and other tricks. The second strategy implies conceptualizing a larger number of (often similar, if not identical) views, and dropping useful hints from the physical structures. However, view integration is easier, since it corresponds to processes that are now standard[8] [B1], [B3], [S1].

7. DATA STRUCTURE CONCEPTUALIZATION

The objective of this process is to discard technical constructs from the DMS-compliant schema, to reduce DBMS-dependent constructs, to eliminate performance-oriented data redundancies, to make hidden conceptual data structures explicit, and to produce a clear, normalized and natural conceptual schema.

7.1 Schema cleaning and renaming

The schema may include constructs that do not pertain to the database itself. Structures for transient data, state variables of programs, as well as dead parts must be detected and eliminated. This process requires not only domain knowledge, but also information on the structures of the programs. In addition, the name of the data structures may need translation for various reasons : weakness of the DMS syntax conventions, reserved words (DMS, host language, local standard), multi-lingual naming, heterogeneous and undisciplined naming conventions, etc.

7.2 Elimination of DMS-specific optimization constructs

The schema is examined for optimization constructs that are specific to the origin DMS[9]. A deep knowledge in the physical behaviour of the DMS is required. Some examples : field padding for address alignment, record splitting when the size is greater than page size, grouping frequently accessed records in the same database space stored in a high-speed device, defining non-information-bearing set types in CODASYL databases, etc. These constructs must be recognized and discarded.

7.3 *Un-translating* the DMS-compliant schema

Producing a DMS-compliant schema implies translating the data structures that are not supported by the DMS. Detecting such transformed structures, and replacing them by their conceptual origins lead to a higher-level schema. A good knowledge of the forward transformation rules generally used when translating into the DMS model is necessary. The biggest problem is that there is generally no 1-1 mapping between the conceptual structures and their DMS-compliant translation. A conceptual construct can be translated into several DMS structures, while several different conceptual constructs can be translated into the same DMS structure.

7.4 Elimination of DMS-independent optimi-zation constructs

Some optimization practices are valid whatever the target DMS. Examples : merging/splitting record types, derived attributes, denormalization. Recognizing them allows one to discard them.

7.5 Expressing the schema in a higher-level model

Eliminating optimization-oriented structures and retrieving the conceptual origin of each construct of the DMS schema

8 There are currently no literature on integrating physical data structures.

9 In all generality, performance and technical efficiency are only examples of requirements that can lead to schema restructuring. Availability, security, data modularity, distribution, etc, are other examples of requirements that will shape the final schema. Being aware of these requirements should ease the reverse engineering process.

can produce a schema that is still awkward and unclear. By restructuring this schema, it is possible to make it more readable and more natural. Replacing binary structures by n-ary ones, extracting complicated compound multivalued attributes to replace them by entity types, defining generic entity type by factoring similar semantic properties of a set of entity types, generating specific entity types from optional, exclusive attributes and roles, are some examples of transformations that can help to meet these objectives.

7.6 Integrating schemas

When the views extracted from the source programs have been conceptualized, they should be merged once they have reached this state. Schema integration will also occur when reverse engineering an information system consisting of more than one database, or a heterogeneous database (such as an IMS database + VSAM files).

8. *DBRE* TECHNIQUES

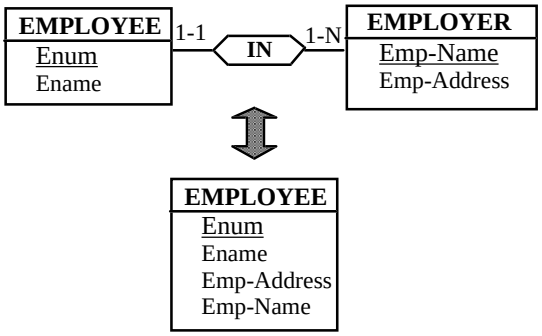
Several major DBRE processes are based on a limited set of common techniques. We shall describe shortly three of them, namely *schema transformation*, *redundancy elimination* and *schema integration*.

8.1 Schema transformation

Schema transformation is the basic tool in many database design activities. An in-depth discussion of the concept can be found in [H1] and [H2]. Grossly speaking, a transformation consists in replacing a data structure with another one which has some sort of equivalence with the former. The most important equivalence that is sought is semantic equivalence. In this case, both schemas express exactly the same semantics; in addition such a transformation is said to be reversible, i.e. it exists another, inverse, transformation that transforms the final schema into the former one.

In most cases, the final structure satisfies a criterion the former doesn't meet. DMS compliance, space or time efficiency, normalization are such criteria.

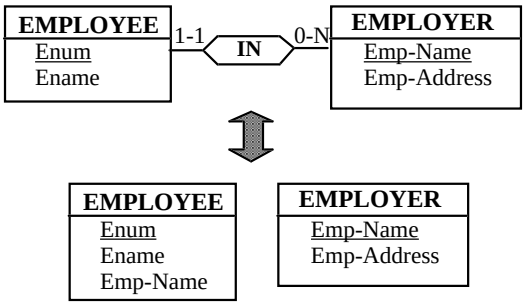
In the database forward engineering model we rely on, the final schema is obtained by successive transformations of the conceptual schema. When transformations guaranteeing semantic equivalence have been used, reverse engineering their resulting schema can be done by using the inverse transformation.



Emp-Name --> Emp-Address

Figure 8.1 - Read top-down, the transformation denormalizes the schema, leading to better access performances, or decreasing the number of entity types for instance. It is a typical design transformation that can be used in activities L2 and L4. Read bottom-up this transformation eradicates a transitive FD, and therefore normalizes the attribute structure of EMPLOYEE. It is a typical reverse engineering transformation.

We shall illustrate the concept with an example of transformation that eliminates a transitive functional dependency that holds into the attributes of an entity type (Figure 8.1), and with another example that tends to make a schema *more relational* (Figure 8.2). Both transformations ensure semantic equivalence.



EMPLOYEE.Emp-Name in EMPLOYER.Emp-Nam

Figure 8.2 - Read top-down, this transformation allows the representation of many-to-one relationship types with reference attributes, a construct that is compatible with relational databases and standard files (used in activity L3). Read bottom-up, it allows the explicitation of relationship types in a reverse engineering activity.

It can be shown that a fairly small number of standard transformations can explain how most optimized DMS-compliant schemas encountered in practice have been obtained. Understanding their mechanism, and how they relate with optimization and translation reasonings is essential for reverse engineering complex schemas.

8.2 Data redundancy elimination

Introducing data redundancies in a database schema is very common. The main objectives are better access performances, higher availability or recovery (they are

used in processes L2 and L4). Recognizing redundant structures, and understanding their fundamental mechanisms is important in the reverse engineering activities.

There are two basic techniques for defining redundancies in a schema :

- through **structural redundancy**, a new object type B is added into the schema in such a way that instances of B can be computed from instances of other object types of the schema. Examples : attribute `Total-Amount` in entity type `ORDER`, attribute `Number-of-Employees` in entity type `EMPLOYER`.
- **denormalization** consists in grouping independent fact types in such a way that their instances are available simultaneously. This construct reduces the number of aggregates (entity or relationship type) and may decrease the access time. See example in Figure 8.1.

8.3 Schema integration

Schema (or view) integration is a design domain that studies the merging of specifications, i.e. schemas, whose real worlds may overlap. This merging is not a pure addition of these source schemas since each fact/object of the real world must be represented only once in the resulting schema. Given a collection of source schemas, several integration strategies have been proposed : merging two schemas at a time or all the schemas in parallel, producing a new schema or augmenting one of the source schema. Binary integration is generally carried out in four steps :

- preparation of the schema (optional) : restructuration of some schemas in order to make the merging easier; in some techniques, it allows to automate the next two steps;
- correspondence : an object type in one schema is related to an object type in the other schema to state the similarity of the real world objects/facts they describe. The kind of this relationship is stated : they are the same, one is included in the other one, they have a common generic type, etc;
- merging : the object types in correspondence are merged, together with their relationships with the other object types of the schemas;
- restructuration (optional) : the final schema is refined in order to make it simpler and more readable.

In reverse engineering, schema integration can occur at different levels. At the conceptual level (merging conceptualized views), the problem is fairly standard. At lower levels, such as integrating DMS-compliant views, the problem can be somewhat more complex, because the schemas include non-semantic constructs.

9. SPECIFIC *DBRE* METHODOLOGIES

The principles that have been presented so far are not dedicated to specific RE approaches, nor to specific DMS. With this respect, they must be perceived as a generic framework in which such specialized approaches can be defined. It is fairly easy to state the dependency of each RE process on specific contextual aspects. For instance,

- processes 6.1, 6.2, 7.1 depend on corporate programming standards,
- processes 6.3, 7.6 depend on the number and similarity of the views,
- processes 6.1, 6.2, 6.3, 7.1, 7.2, 7.3 depend on the DMS,
- processes 7.2, 7.4 depend on optimization requirements,
- process 7.5 depends on corporate analysis standards.

Specializing these processes according to these criteria provides us with a specific DBRE method. It should be noted that this specialization can rely mainly on the genericity of such components as the unique data model and of the schema transformations. For instance, specialization according to the DMS can be defined by specializing the data model (it can be limited to the structures known by the DMS), and by selecting the schema transformations that could have been used by the database designer for translating into this DMS. Such methodological specialization has been studied in [H2] for DB forward engineering.

10. CONCLUSIONS

Except in oversimplistic, or accidently favourable situations, reverse engineering a database from source DDL/Host language texts is a complex task that still needs in-depth research. Understanding formal and empirical design methodologies, together with their underlying techniques and reasonings, is a considerable asset in this process, mainly because it helps to identify the main design processes that shape the final schema by including specific requirements. A careful analysis of that *shape* can give hints as to the conceptual/technical/organizational origin of the observed data structures. The paper establishes a general framework for understanding practical (i.e. mainly intuitive and non formal) design methodologies and to analyse the possible ways a conceptual schema could have been transformed into the observed schema.

It must be clear that reverse engineering cannot be fully automated process. Indeed, it must integrate not only formal knowledge on database modeling and design, technical knowledge on the implementation tools, but also knowledge on how programmers program(med), how designers design(ed), as individuals (through their personal behaviour) and as members of an organization (following possible methodological standards). In addition,

knowledge on the application domain, together with information from other sources (obsolete and incomplete documentation, data dictionary, file contents, etc) are generally used to support the source text analysis.

The PHENIX project, from which some of the principles presented hereabove have been borrowed, has developed a specific method for standard (COBOL) file reverse engineering. Processing of relational and CODASYL structures is under investigation. The reader will find in [J1] further details on the PHENIX project, and particularly on the expert-system that has been developed.

11. CREDIT

Parts of the material on which this paper is based are results of the PHENIX research project. PHENIX is a four-year (1989-1993) industry-university research project developed jointly by the FUNDP (University of Namur) and the BIKIT (Babbage Institute for Knowledge and Infor-mation Technology, University of Ghent), and is supported by a consortium of 14 industrials (Barco, BBL, Bell, BIM, E2S, Glaverbel, Groupe S, METSI, Provinces Réunies, Siemens-Nixdorf, Solvay, Telinfo, Sidmar, Warmoes) and by IRSIA/IWOLN, a Belgian Research support agency (contract n° 5421). The objective of the project is to develop an expert-system approach to database reverse-engineer-ing. The staff of the University of Namur comprises M. Chandelon, prof. F. Bodart, prof. J-L Hainaut, M. Joris, B. Mignon and C. Tonneau. The staff of the BIKIT comprises E. Cardon, J. D'Hayer, F. Osaer, P. Tisseghem, prof. Vandamme, prof. Vanwormhoudt, P. Verscheure, R. Van Hoe.

12. REFERENCES

[B1] Batini, C., Lenzerini, M., Navathe, S., B, *A comparative Analysis of Methodologies for Database Schema Integration*, in ACM Computing Survey, Vol. 15, No 4, pp. 323-364, December, 1986

[B2] Batini, C., Ceri, S., Navathe, S., B., *Conceptual Database Design*, Benjamin/Cummings, 1992

[B3] Bouzheghoub, M., Comyn-Wattiau, I., *View Integration by Semantic Unification and Transformation of Data Structures*, in Proc. of 9th Entity-Relationship Approach, 1990

[C1] Casanova, M., Amarel de Sa, J., *Designing Entity Relationship Schemas for Conventional Information Systems*, in Proc. of Entity-Relationship Approach, pp. 265-278, 1983

[C2] Casanova, M., A., Amaral De Sa, *Mapping uninterpreted Schemes into Entity-Relationship diagrams : two applications to conceptual schema design*, in IBM J. Res. & Develop., Vol. 28, No 1, January, 1984

[C3] Chen, P., P., *The Entity-Relationship Model - Towards a Unified View of Data*, in ACM TODS, Vol. 1, No 1, pp. 9-36, , 1976

[D1] Davis, K., H., Adarsh, K., A., *A Methodology for Translating a Conventional File System into an Entity-Relationship Model*, in Proc. of Entity-Relationship Approach, Octobre, 1985

[D2] Davis, K., H., Arora, A., K., *Converting a Relational Database model to an Entity Relationship Model*, in Proc. of Entity-Relationship Approach : a Bridge to the User, 1988

[E1] Elmasri, R., Navathe, S., *Fundamentals of Database Systems*, Benjamin/Cummings, 1989

[F1] Fonkam, M., M., Gray, W., A., *An approach to Eliciting the Semantics of Relational Databases*, in Proc. of Entity-Relationship Approach - ER'92, pp. 463-480, October, 1992

[H1] Hainaut, J-L., *Entity-generating Schema Transformation for Entity-Relationship Models*, in Proc. of Entity-Relationship Approach, 1991

[H2] Hainaut, J-L., Cadelli, M., Decuyper, B., Marchand, O., *Database CASE Tool Architecture : Principles for Flexible Design Strategies*, in Proc. of the 4th Int. Conf. on Advanced Information System Engineering (CAiSE-92), Manchester, May 1992, Springer-Verlag, LNCS, 1992

[H3] *Software Reuse and Reverse Engineering in Practice*, Hall, P., A., V. (Ed.), Chapman&Hall, 1992

[I1] *Special issue on Reverse Engineering*, IEEE Software, January, 1990

[J1] Joris, M., Van Hoe, R., Hainaut, J-L., Chandelon M., Tonneau C., Bodart F. et al., *PHENIX : methods and tools for database reverse engineering*, in Proc. 5th Int. Conf. on Software Engineering and Applications, Toulouse, 7-11 December, 1992

[N1] Navathe, S., B., Awong, A., *Abstracting Relational and Hierarchical Data with a Semantic Data Model*, in Proc. of Entity-Relationship Approach : a Bridge to the User, 1988

[N2] Nilsson,E., G., *The Translation of COBOL Data Structure to an Entity-Relationship Type Conceptual Schema*, in Proc. of Entity-Relationship Approach, October, 1985

[R1] Rock-Evans, R., *Reverse Engineering : Markets, Methods and Tools*, OVUM report, 1990

[S1] Spaccapietra, S., Parent, C., *View Integration : A Step Forward in Solving Structural Conflicts*, Res. Report , EPFL, Lausanne (CH), IEEE Trans. on Knowledge and Data Engineering, October, 1992

[S2] Springsteel, F., N., Kou, C., *Reverse Data Engineering of E-R designed Relational schemas*, in Proc.

of Databases, Parallel Architectures and their Applications, March, 1990

[W1] Winans, J., Davis, K., H., *Software Reverse Engineering from a Currently Existing IMS Database to an Entity-Relationship Model*, in Proc. of Entity-Relationship Approach : the Core of Conceptual Modelling, pp. 345-360, October, 1990