Proceedings of the 5th International Conference
on Software Engineering and its Applications
Toulouse (France), December 1992

PHENIX : Methods and Tools for Database Reverse Engineering

Joris, M.[1], Van Hoe[2], R., Hainaut, J-L[1],
Cardon, E.[2], Chandelon, M.[1], Tonneau, C.[1], Verscheure, P.[2]
Bodart, F.[1], Vandamme, F.[2], Vanwormhoudt, M.[2]

1 Institut d'Informatique
Facultés Universitaires N-D de la Paix
rue Grandgagnage, 21
B-5000 Namur (Belgium)
fax : +32 81-72.49.67 or +32 81-23.03.91
email : jlh@info.fundp.ac.be

2 BIKIT
University of Gent
Plateaustraat, 22
B-9000 Gent (Belgium)
fax : +32 91-64.41.97
email : phenix@episto.rug.ac.be

Abstract

The paper presents methodological principles concerning the problem of recovering the semantic structures of actual file and database of old and poorly documented applications. This problem, called *Database Reverse Engineering*, is related to the standard database design paradigm, revisited in order to explain and understand informal and empirical design behaviours. A general method is proposed, based on two major phases, namely data structure extraction and data structure conceptualization. An expert assistant has been implemented to help professionals to practice this method.

Keywords : reverse engineering, database design, CASE tools, AI techniques

Résumé

L'article présente les principes méthodologiques relatifs au recouvrement des structures sémantiques des fichiers et bases de données d'applications anciennes et mal documentées. Ce problème, connu sous le terme de rétro-ingénierie de bases de données, est resitué dans le cadre des démarches standards de conception de bases de données, démarches réexaminées dans le but de comprendre et d'expliquer les comportements de conception informels et empiriques. On propose une méthode générale, structurée en deux phases principales, l'extraction des structures de données et leur conceptualisation. Un système expert d'assistance basé sur ces principes a été réalisé.

Mots clés : retro-ingénierie, conception de base de données, ateliers de génie logiciel, techniques d'AI

1. INTRODUCTION

The problem

Reverse engineering a software component consists in recovering its functional and technical description, starting mainly from the source text of the programs. Recovering these specifications is generally intended to convert, restructure, maintain or extend old applications [14], [18], [13]. It is also required when developing a Data Administration function. This paper tackles the problem of reverse engineering file and database structures[1].

The essence of database reverse engineering can be simply stated as follows :

> *considering a given collection of databases, of which operational (i.e. machine readable) descriptions, programs that manipulate their contents, interfaces (screens, reports, etc) that presents them, and possibly their contents themselves, are available, retrieve a possible conceptual schema of which these databases can be a correct implementation.*

The difficulty of the process is that, in most cases, the most up-to-date documentation of the data structures is the program source code itself. Generally, it is the only documentation that remains.

Database reverse engineering is only a part of application program reverse engineering [18], an activity that 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 burried in the programs, and therefore definitely lost. However, 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.

Splitting the problem in this way can be supported by the following arguments :

- the semantic distance between the conceptual specifications and the physical implementation is most often shorter 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;
- reverse engineering the procedural part of an application will be 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 tackle the whole application in one step. Though reverse engineering data structures still is a complex task, it appears that the current state of the art in database design provides us with sufficiently powerful theories and techniques to make this entreprise more realistic.

State of the art

Some work has been done during the 80's on data structures reverse engineering, but surprisingly not as much as could have been expected. The problem of COBOL files is tackled in [5], [17], [8], that of IMS databases in [16], [22], CODASYL databases in [1]], while [6], [16], [9], [21] are dedicated to the problem of relational structures. All of them target an entity-relationship expression of the conceptual structures.

A more ambitious endeavour is the REDO Esprit project dedicated to the development of a methodological approach and tools for software maintenance. The data-remodelling function handles the COBOL file structures only. COBOL program are parsed and information from the I-O and FILE sections is stored in the system description database (SDDB), the REDO repository that contains an abstract representation of the source texts. This information is then processed (through rule-based techniques) to produce, either automatically or interactively, an entity-relationship schema that can be further modified by the designer. Links between the conceptual specifications and their COBOL origin are maintained.

The best known commercial database reverse engineering tool clearly is the Bachman re-engineering toolset (Bachman Information Systems Inc, USA). This CASE tool offers both forward and reverse engineering functions in an integrated environment. The reverse engineering modules extract data structure definitions from standard file systems, DB2, DL/1 (IMS) and IDMS (CODASYL) databases automatically and load them in the repository of the workstation. These structures are converted and merged into a Bachman data model that can be edited by the user. The process is assisted by a limited rule-based *co-pilot*. The recovered data

---

1 From now on, we shall call ***database***, any structured collection of data that is perceived as a whole by one or several applications. This definition encompasses *true* databases, managed by DBMS, standard files, such as COBOL files, and even in-core data structures.

model can then be processed by the forward engineering modules to produce DB2 schemas for instance. Among the other available tools that offer reverse engineering functions, let's mention SOFTORG (SZAMALK, Hungary), processing COBOL, PL/1 and assembler data structures, SYNON/2 (SYNON Inc., USA) dedicated to IBM mid-range systems migration and AISLE (Software System Design, USA) for ADA environments. These environments offer functions to parse source texts and to recover some kind of computer-independent data schema.

All these proposals are based on a somewhat simplistic view of the problem according to which the available source code is a straighforward and as complete as possible expression of the source conceptual schema, be it actual or virtual. For instance these proposals share some common hypotheses such as :

- all the knowledge on the data structures is to be found in their declaration statements (and not in the procedural sections for instance);
- names are fully significant and consistent;
- no technical tricks have been used in order to optimize the data structures;
- the data structures have not evolved and have suffered no time degradation.

Therefore, DDL[2] text parsing is straightforward and automatic, and the semantic interpretation of the elicitated data structures is immediate thanks to a limited set of simple rules.

Field experiments show that the actual problems are far more complex. Indeed, it appears that actual programming practices tend to produce obscure code as far as data structures are concerned. Structure hiding (see below), naming inconsistencies, integrity control spreading and duplicating, tricky structure optimization, integration of requirements evolution, multiple viewing of the data, redundancy, are some common phenomena that make data structures recovering and understanding much harder than expected. The problem is particularly sharp for standard file management systems, but it may occur with true database management systems as well.

The PHENIX project

PHENIX is a four-year (1989-1993) industry-university research project developed jointly by the FUNDP (Facultés Universitaires Notre-Dame de la Paix, Namur) and the BIKIT (Babbage Institute for Knowledge and Information Technology, University of Ghent), and supported by a consortium of 14 industrials[3] and by IRSIA/IWOLN, a Belgian Research support agency (contract n° 5421).

Its scope is database reverse engineering through A.I. techniques, and its objectives are to understand the problem of reverse engineering, to propose general models and methods for database reverse engineering, and to develop a prototype expert-system to assist professionals in this activity.

It proposes a deeper analysis of the reverse engineering activities that tries to take into account the weaknesses of the current technology. This analysis is based on two major sources of knowledge. The first source is the theoretical and technical state of the art in database forward engineering, i.e. database design. This realm is now well mastered, and provides us with adequate formal knowledge either for the reverse engineering processes, or to understand how databases have been designed. The second source of knowledge is through real-world case studies. The industrial partners have provided the two laboratories with actual applications to reverse engineer. Six researchers spent from two to six months in recovering the conceptual schema of the data structures of these applications. During these experiments, they collected invaluable information on the problems that actually occur in the real world, on how to solve them, and on how reverse engineers behave and reason during this activity (e.g. what heuristics do they use to find out the identifier of a file). This knowledge has been formalized and has been used as raw material for a specific method for database reverse engineering. The expert-system has been designed to help reverse engineers to carry out tedious activities (such as intelligent source text browsing) and to guide them in knowledge-based activities.

Organization of the paper

The paper is organized as follows. Section 2 proposes a revised analysis of the database design activity in order to understand how a conceptual schema has been translated into executable descriptions. Section 3 develops a general method for database reverse engineering, while section 4 describes some important techniques. Section 5 is dedicated to the PHENIX CASE expert-system.

---

2 Data Definition Language : that part of programming or database languages that is used to declare data structures.

3 Barco, BBL, Bell, BIM, E2S, Glaverbel, Groupe S, METSI, Provinces Réunies, Siemens-Nixdorf, Solvay, Telinfo, Sidmar, Warmoes.

The reader is supposed to have sufficient knowledge in database and in database design (otherwise, see [Elmasri, Navathe, *Fundamental of Database Systems*, Benjamin/Cummings,1989] and [1] resp.).

2. DATABASE FORWARD ENGINEERING REVISITED

It is argued that data structure reverse engineering cannot be efficiently done if both *how* and *why* the DDL text has been produced is ignored. When this text results from a strict method supported by a comprehensive CASE tool, the task is fairly easy because the generation rules are simple, and the initiative of the designer is limited to the conceptual phase. However, most older applications (and, unfortunately, many current ones as well) have been designed empirically, i.e. without any methodological concern. Therefore, we are forced to re-examine the database design problem with a particular emphasis on how analysts and programmers *actually behave(d)*, and not how they should (have) behave(d).
We consider that producing operational data structures consists in solving, explicitly or not, a limited set of design problems. Each problem is solved by a (actual or fictive) design process based on specific concepts, techniques and reasonings. Both standard methods and empirical design try to solve these problems. However, in the latter case, the solving processes are generally implicit. Understanding these problems and these processes allows us to understand how an arbitrary conceptual schema has been transformed into an operational schema. Since empirical design implies a great deal of initiative and expertise, the standard database design model must be refined.
The standard design model suggests the following phases [1] :

Conceptual analysis
: Produces a clear, readable, complete, non-redundant and normalized formal representation of the static aspects of the universe of discourse.

Logical design
: Produces the logical schema, a model-dependent, but DBMS-independent expression of the conceptual schema. Some optimization can be carried out on this schema (optimized logical schema).

Physical design
: Improves the logical schema as far as performance criteria are concerned, and makes it operational (DDL expression). Generally, this is made by tuning the DBMS physical parameters. Note that some conceptual specifications may be left out of the DDL expression, and that they have to be coded in the application programs themselves.

User's view derivation
: Generates the DDL expression of views or sub-schemas, i.e. of the limited perception of the schema that will be available to each user (e.g. application programs).

Empirical design behaviours are more complex. In particular, optimization can occur during logical design, schema restructuring can be used in physical design, view derivation may imply hiding the structure of a field or of a record type.
Here follows a short list of practical problems that have to be solved in designing data-intensive applications. It can be seen as refinement of the standard framework with special emphasis on specific requirements.

1. **Improving time performances and storage space**. Data structures are splitted or merged. Merging structures will shorten the access paths between structures, splitting them will allow to define different storage and access strategies for each of the resulting parts. De-normalization and data redundancies are introduced by duplicating information in order to reduce access times.
2. **Name conversion**. Conceptual names are modified, to fulfill not only the naming rules of a specific DBMS, but also possible corporate standards or programming tricks.
3. **Translation of conceptual structures** into the DBMS model. Expressing the conceptual structures into DBMS constructs uses a much greater variety of techniques than is suggested in standard methods. This is particularly true with standard Cobol files. In addition, there is no commonly agreed upon way to control lost integrity constraints (e.g. maintaining the referential integrity in a set of COBOL files). Therefore, procedural sections managing the lost structures are spread and duplicated throughout the programs.
4. **Physical schema coding**. A DDL expression is available in most DBMS. In some cases, details of the logical schema are hidden. Such is the case for some CODASYL systems, in which the schema-DDL does not allow compound data items while the sub-schema-DDL allows them. In low-level data managers, such as standard file systems, the very notion of DDL schema does not exist.
5. **View derivation**. In standard file systems, there are no way to explicitly define views. In this case, we can call *view* the description of the data structures that are known by a program.

6. **View coding**. As suggested in section 4, some details from the logical schema can be coded in a view instead of in the DDL text. In standard file systems, views can be coded as record structures. However, current programming practices tend to hide the view structures. The so-called *structure-hiding*, illustrated in figure 1 is a common practice. The file record structure is partly hidden, but it can be retrieved (and therefore recovered during reverse engineering) by observing the data flow in the programs.

```
FD CUST.
   01 C-REC.
      02 CUST-ID.
         03 CUST-NUM PIC X(10).
         03 NAME PIC X(40).
      02 filler PIC X(95).
   ...
WORKING-STORAGE SECTION.
   01 CUS-ADD
      02 filler pic X(50).
      02 ADDRESS.
         03 STREET PIC X(40).
         03 NB     PIC X(5).
         03 CITY   PIC X(50).
   ...
PROCEDURE DIVISION.
   ...
   READ CUST.
   ...
   MOVE C-REC INTO CUST-ADD.
   ...
```

Figure 1 *- Example of COBOL structure hiding.*

We are now provided with a more realistic overview of the database design activities that are actually practiced. Therefore, we are ready to define a general architecture of what could be a realistic view of database reverse engineering.

3. GENERAL METHOD FOR DATABASE REVERSE ENGINEERING

Within this context, reverse engineering a physical database schema (expressed as a DDL text for instance) consists in recovering the origin conceptual schema, or more realistically, to elaborate one of the possible conceptual schemas that could have been implemented into this physical schema. In short, it can be considered as *undoing* the activities defined in section 2. Analyzing the source code for retrieving the executed actions can be compared with a detective job. There are evidences, but certainties are rather scarce.

When examining the design process, an important milestone seems to emerge, namely the **optimized logical schema**. Indeed, on the one hand, this schema is the complete (possibly optimized) translation of the conceptual schema in terms of the concepts of the DBMS model, and on the other hand, it has not been coded yet.

Let's define a reverse engineering procedure that exhibits the concept of optimized logical schema. Two processes are put forward :

- ***data structure extraction***. This activity consists in recovering the database structures in terms of the DBMS model. These structures make up the optimized DBMS logical schema.
- ***data structure conceptualization***. Through this activity, the meaning of the data structures are made explicit by translating them into a conceptual schema.

Before developing these processes, we will define the model in which the schemas will be expressed.

3.1 A model for data structures specification

The model used to record the data specifications during reverse engineering is unique but generic : it models the various data structures, whatever their abstraction level[4] (coded, physical, logical, conceptual). We have chosen a variant of the entity-relationship model [7] that has been extended to the concepts needed at all the levels of the database life cycle considered in reverse engineering. The main concepts are briefly sketched herebelow :

- entity type, relationship type (n-ary, possible attributes), roles (possibly multi-entity-types) and attributes (possibly compound, optional and/or multivalued).
- generalization/specialization of entity types.
- attribute typing concepts : domain, format, logical and physical lengths, start and end positions, intervals, special values.
- constraints : cardinalities of a role, cardinality of an attribute, identifiers, referential constraints, etc.
- access structures : access keys, sort keys, relative keys, pointers.
- programming structures : source text files, programs, logical data files, physical data files, variables, reports, transfer instructions, calls, logical file uses.

3.2 Data structure extraction

The goal of the data structure extraction process is to make the DBMS-dependent data structures explicit. Section 2 has shown, particularly for standard file systems, that the description of significant structures can be hidden, spread and duplicated throughout the text of several programs. For richer DBMS, which have a centralized repository of data specifications, the data structure extraction process can be much simpler and can be reduced to some translation rules. However, the problem of recovering the concepts that the DBMS is unable to cope with still remains. One frequent example is referential integrity in some relational schemas.

In this paper, we have concentrated on standard (COBOL) file reverse engineering since this problem is both the most vital in companies and administrations, and the most challenging in the software engineering field. According to the field experiments that have been conducted, the following procedure can be suggested.

- **Collecting information sources**
  Collect all the sources of the application programs that can be relevant.
- **Building the general structure of the sources**
  Production of a flowchart by a gross analysis of the program texts, exhibiting their calling relations, their logical files and the flows between them. This flowchart will contain the file exchanges between programs and the calls between them, providing a global view of the applications and a better understanding of its behaviour.
- **Source selection**
  Selection of the most pertinent source files, i.e. typically the kernel of the data processings. The programs, if any, dedicated to off-line integrity checkings, will also be needed to retrieve data integrity constraints[5].
- **Physical concepts extraction**
  Extraction of the main concepts[6] : entity types, attributes, typing concepts, logical files, access structures.
- **Structure refinement**
  Extraction of alternative descriptions : for all data structures, and particularly for those which seem to be grossly defined, trace transfer-instructions to/from these structures in order to detect additional and possible more detailed descriptions (Figure 1).
- **Elicitation of integrity constraints**
  This mainly concerns identifying and referential constraints. Although this problem is very complex (a predicate can be coded in various ways in an algorithm), careful pattern analysis in the procedural sections can give strong evidences. Let's give some suggestions about this :
  - Programmers tend to use systematic patterns to express similar situations. For instance, all the referential constraints are generally coded in the same way throughout the programs of the same

---

4 The principle of a unique, generic model to cover the complete design life cycle is discussed in [12].

5 Note that in some architectures, procedural sections can be attached to user interface components or database triggers. Though the analysis may be easier in this case they can be processed in the same way.

6 As a consequence of the unique representation model, some entity-relationship concepts have a particular interpretation at the physical level. A record type is represented by a (physical) entity type and a data field by a (physical) attribute.

author. So, finding out this pattern (there are not so many of them anyway) allows to make this kind of constraint explicit.

- Error-handling sections are rather easy to find. They give essential information on file manipulation exceptions, and particularly integrity constraints violations. Finding where these sections are called from gives clear indications on the concerned constraints.
- Naming conventions are important. Names such as CUS-ID, ORDER-REF, PROD-CODE suggest identification of major entity types. This information gives evidences that have to be validated through procedural section analysis.

- **Intra-view redundancy reduction**
  So far, the processes described above give one schema for each source text. This schema may (and often will) contain description redundancies. For instance, a file copy requires opening the input file and the output file. These two files will be described in the schema, though they bear the same semantics. Therefore, this schema needs to be cleaned up by merging similar descriptions and by discarding identical ones[7].
- **Inter-view redundancy reduction**
  Each schema is one view of the logical schema only. All the views elaborated in this way need to be integrated (merged) into a unique logical schema. In a 6 data files application that has been processed, up to 120 views had to be integrated. An alternative method is to conceptualize each logical view first, then to merge their conceptual expressions (see below).

3.3 Data structure conceptualization

This process builds a possible conceptual schema as an abstraction of the semantics that is burried into the logical schema obtained through data structure extraction. Knowing by which manipulations the conceptual schema can be translated and transformed into a logical schema, we can propose reverse engineering activities that try to undo the effect of these manipulations.

- **DBMS un-translation**
  The structures which are compliant to the DBMS are transformed into a higher abstract level, to obtain a more concise and more readable expression. One of the most frequent transformation consists in replacing reference attributes + referential constraint by a one-to-many relationship type. This process requires knowledge on the transformation techniques that have been used to produce the logical schema.
- **Re-conceptualization of names**
  The names found in the code are replaced with more meaningful names. This requires knowledge on programming standard, on the constraints of the DBMS, but also some domain knowledge.
- **De-optimization restructuration**
  The splitting/merging transformations used to optimize the schema have to be detected and reversed. Un-normalized structures have to be detected and transformed to eliminate redundancies. The structural redundancies introduced for improving performances must be removed. However, data redundancy sometimes implies also multiple views (two redundant data structures can have different descriptions). So removing data redundancy will have to be coupled with the integration of multiple views.
- **Schema merging**
  If more than one schema has been conceptualized, be they related to one database or to several distinct databases, they must be merged into a unique conceptual schema.
- **Schema restructuring**
  The resulting schema can be condensed and improved in order to make it more readable and clearer. For example, some entity types can be transformed into many-to-many relationship types and IS-A structures can be defined.

4. SPECIFIC TECHNIQUES

The activities described in section 3 are based on a small set of specific techniques. Two of them are of particular importance, namely **schema transformation** and **schema integration**. Due to their broad scope, they are used in database design as well and have received considerable attention both at theoretical and practical levels [1]. However, their application to reverse engineering still need some extension and adaptation.

---

7 Note that discarding a description must be done carefully. Indeed, two descriptions can be identical while they are related to two different concepts, such as CUSTOMERS and PROSPECTS.

4.1 Schema transformation

Transforming a schema consists in applying structural modifications to this schema, keeping (as much as possible) its semantics unchanged. An acceptable definition of the term "semantics" is here "the facts expressed by the data structures". Therefore, the syntax of the representation is modified while its semantics is kept unchanged. A formal discussion of this concept can be found in [11].

If a transformation ensures semantics preservation, it is said *reversible* and can be applied in both directions. This fact is obviously useful for the reverse engineering : when the trace of applying a transformation has been detected, applying its inverse recovers the origin structure.

As it appears from section 2, transformations are basic tools in database design in the following activities : schema translation into DBMS model and schema optimization. In database reverse engineering, transformations will be used in the following activities : DBMS un-translation, de-optimization restructuration and schema restructurating.

The concept will be illustrated by two important transformations that are typical to the production of COBOL file structures from conceptual schemas. The first transformation (figure 2) replaces a dependent entity type by a multivalued attribute. Since it is reversible, the transformation can be applied both from left to right and right to left. In terms of reverse engineering, it states the conditions in which a multivalued attribute can be replaced, without semantic loss, by an entity type.

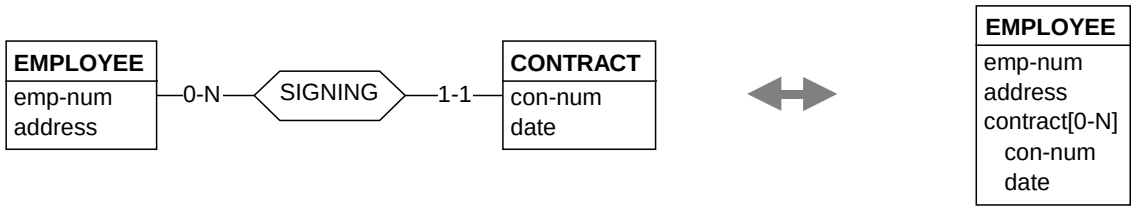

Figure 2 - *Transformation of an entity type into a multivalued attribute, and conversely.*

Figure 3 illustrates another way to get rid of a one-to-many relationship type. The latter is replaced by a reference attribute + a referential constraint. In terms of reverse engineering, it states how to make some relationship types explicit.

The PHENIX method and tool provide some 20 basic transformations that can be used to untranslate and de-optimize logical schemas.

The availability of the transformational toolset does not solve the problem of *when and where* to apply them. Each transformation is accompanied by two set of rules. The first one is purely technical; it expresses the preconditions the schema **must satisfy** in order to be transformed. The second one tries to describe application heuristics and gives some of the conditions under which the transformation **may be used**. It therefore suggests situations when applying it can be useful.

4.2 Schema integration

Schema integration is also a topic that has been intensively worked out in the database field, but generally in a different context. Most studies [2] [4] [20] tackle the problem of deriving a common conceptual view (schema) from a collection of partial, incomplete and conflicting user's views. In the PHENIX method and tool, schema integration can occur at several steps, namely *intra-view redundancy reduction*, *inter-view redundancy reduction* and *schema merging*. There are two essential characteristics that make the problem specific : the views are known to describe the same database, and the views may include access constructs (e.g. indices) and technical structures.

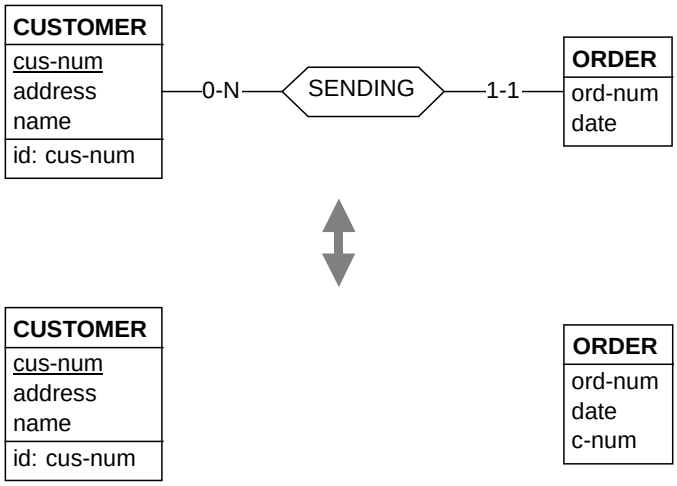

Figure 3 - *Transformation of a relationship type into a reference attribute, and conversely.*

The PHENIX integration strategy is made up of two main steps, namely correspondence definition and view reduction.

1. Correspondence definition

A correspondence between constructs A and B (generally from distinct schemas) means that the sets of real-world objects that A and B denote are identical at any time. In short, A and B have the same semantics. Finding the correspondences is the core of the integration problem. This task cannot be fully automated and requires important domain knowledge. However, it can be assisted thanks to detection heuristics based for instance on name similarities, starting position and length, structural context (other related elements). Figure 4 illustrates two views and correspondences between their elements.

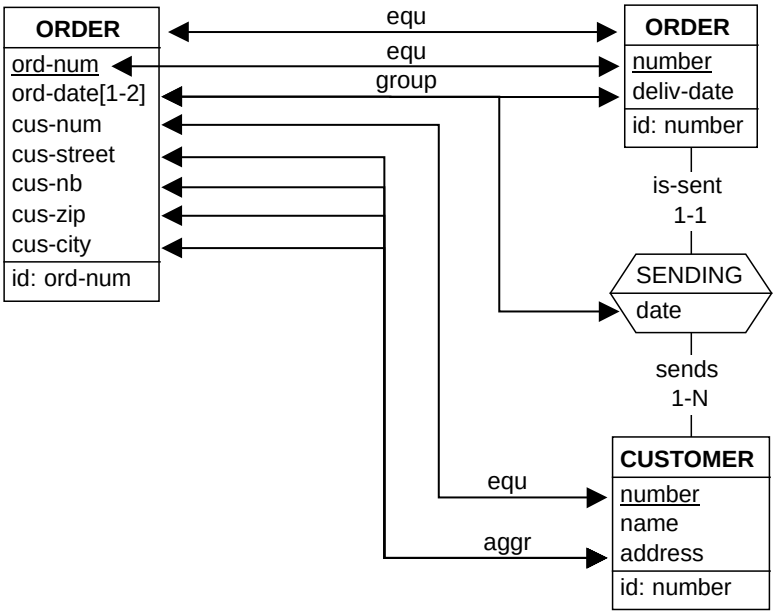

Figure 4 - *Correspondences between the elements of two views.*

2. View reduction

Elements that correspond are reduced to one element only. The nature (attribute, entity type, relationship type) of the origin elements can be different. For instance, an entity type and an attribute can be said to have the same semantics. Figure 5 shows the schema that results from the integration of the views of figure 4.

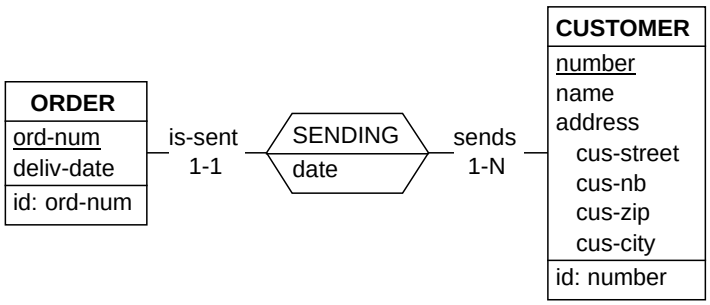

Figure 5 - *Result of the integration process.*

The reduction technique is close to that suggested in [20], and uses an internal representation of the schemas as semantic networks. It allows integrating elements of different nature. It can also process not only the equality correspondence, but also the IS-A correspondence (the semantics of a structure is included into the one of another structure, such as PERSON and EMPLOYEE). Furthermore, correspondences between sets of data structures can also be defined, with reference to the aggregation or grouping of these sets [3].

5. THE PHENIX CARE TOOL

The objective of the CARE (Computed Aided Reverse Engineering) system is to provide a tool which supports the database reverse engineering method developed in the context of the Phenix project. This implies that the tool was designed as an assistant rather than a toolbox, providing methodological and strategical guidance to a reverse engineer. It was also important that the system tackle the problem of reverse engineering in its whole complexity, and not only provide an automatic simplified tool. The tool is generic in that it meets various goals of reverse engineering and it is not dependent on a specific DBMS. Last but not least, in the specification and implementation of the tool both expert system techniques and more classical methods were applied.

5.1 The functions

The Phenix tool offers the following major functionalities.

1. Extraction.

The extraction process is responsible for parsing the source texts and loading them in the central repository of the system. Three levels of automation are provided in the extraction module.

The main extraction is executed at the beginning of a reverse engineering session in order to extract the basic concepts (e.g., physical entity types) automatically. An incremental extraction can be requested when additional information on a given data structure is needed. In that case a local extraction process is carried out. A manual extraction is also provided since the user can create additional concepts (e.g., integrity constraints), guided by pattern searching in the original source texts.

2. Integration

The integration process can be executed at the schema-level, i.e. the integration of two schemas, or at the level of the basic data structures, e.g. the integration of two entity types. Three suggestion modes are available. The user can know the two structures to be integrated, or (s)he can supply a structure and the tool proposes corresponding structures to be integrated. The tool can also suggest couples of structures to be integrated.

3. Transformation

A set of about 20 transformations which allow to obtain more conceptual structures are available to the user. A transformation can either be requested by the reverse engineer, or suggested by the CARE tool.

4. Name Processing and Management

The user is provided with several name processing functionalities. The user can remove prefixes or suffixes and replace parts of the names appearing in a schema The system can suggest a correct name for new data structures, which is useful during transformation or integration.

5. Semantic Enrichment

The user can perform several semantic enrichment, such as creating, deleting, modifying data structures (e.g., relationship types), defining an IS-A hierarchy on existing data structures, etc.

6. Source Management

At the start of a reverse engineering session, the source management process assists the user in choosing the most important source files by grouping them into schemas on the basis of consistency checking and displaying source text statistics. This process is supported by call and perform graph generators and source text viewers.

7. Version Management.

A save/save-as functionality makes it possible to store different versions of a reverse engineering session. During a session a schema state-tree editor is active which allows multiple undo and the possibility to explore different hypotheses from a certain state on.

8. Editors

Several specialized editors for the main objects in the central repository are available, such as a graphical schema editor, entity type editor, etc.

5.3 Architecture of the Phenix CARE Tool

The functions of the PHENIX CARE tool, as described above, reflect user's needs in terms of main activities and specific techniques (sections 3 and 4). On the other hand, the problem of when and where achieving these activities or applying these techniques in complex situations (i.e. strategies) remains to be solved. The architecture of the PHENIX tool is designed to make it a CARE assistant rather than a mere passive toolbox. The overall behaviour of the system is controlled through a blackboard organization [10] that is well suited to implement reverse engineering strategies based on the heuristics collected during field experiments.

The general architecture of the Phenix CARE tool is depicted in figure 6. As a knowledge-based system [15], the PHENIX CARE tool includes :

- A knowledge base collecting structural knowledge (the *Object Base*, or repository) and problem-solving knowledge (*RE process base* made of the functions described above).
- An inference engine, rule-oriented, and providing forward and backward chaining reasoning.
- A graphical user interface.

As a blackboard-oriented system, the inference engine is encapsulated and driven by a general control module which a strategy module is coupled to.

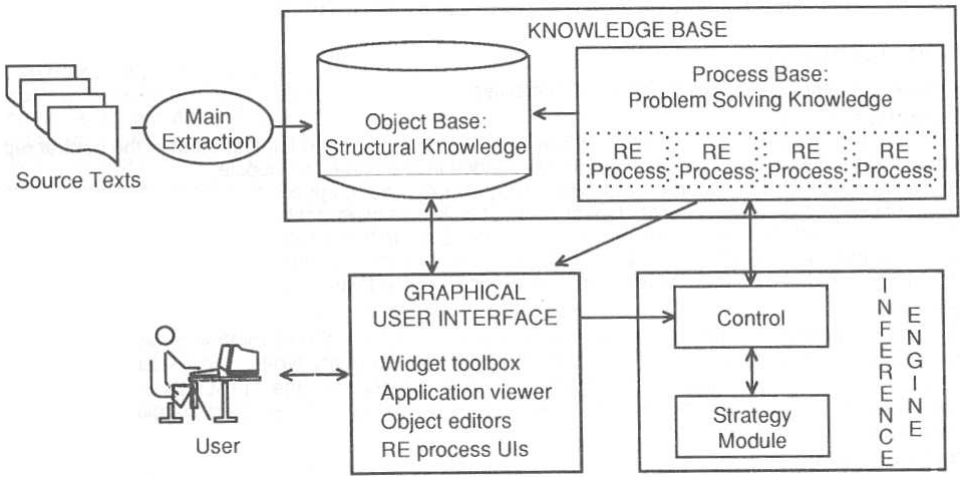

Figure 6 *- Functional architecture of the PHENIX CARE environment*

Implementation aspects

The PHENIX CARE environment has been developed with the frame-based SMECI expert system shell. The user interface is developed with AIDA/MASAI. Both tools are based on Le-Lisp, and are distributed by ILOG.

6. CONCLUSION

By merging theoretical knowledge from the database field with extensive field practice, a general methodological framework of database reverse engineering has been defined. This framework tries to capture actual design and programming behaviour (in particular those which focuse on performance issues) in order to trace the decisions into the resulting source texts of the programs. This knowledge helps in understanding the rationale that stand behind operational technical data structures, and in abstracting the semantic aspects of these structures into a conceptual schema.

Despite the way it has been presented, this framework is not limited to describe sequential methods only. Indeed, experiments suggest that reverse engineering naturally induces parallel activities, not only in each main phase (extraction and conceptualization), but also between them. For instance, a final *Schema restructuration* may suggest going back to the source text to check the existence of an integrity constraint through program pattern searching. In addition the current schema may include parts that still are physical, while others have already been conceptualized. These situations are quite different from what is standard in forward engineering. Hence the need of very flexible methods and tools.

The current efforts concentrate on the integration of intelligent help services in the tool. For instance the implementation of heuristics that have been discovered during field experiments are under investigation.

7. REFERENCES

[1] Batini, C., Ceri, S., Navathe, S., B., *Conceptual Database Design*, Benjamin/Cummings, 1991

[2] 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

[3] Brodie, M., *On modelling behavioural semantics of databases*, Proc. VLDB conf. Cannes, 1981, pp 32-42.

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

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

[6] 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

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

[8] 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

[9] 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

[10] Englemore, R., Morgan, T., *Blackboard Systems*, Addison-Wesley, 1988

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

[12] 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

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

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

[15] Luger, G., F., Stubblefield, W., A., *Artificial Intelligence and the Design of Expert Systems*, Benjamin/Cummings, 1989

[16] 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

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

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

[19] Smith & Smith, *Database Abstractions : Aggregation and Generalization*, ACM Transactions on Database Systems, June 1977, Vol. 2, N. 2, pp. 105-133.

[20] Spaccapietra, S., Parent, C., *View Integration : A Step Forward in Solving Structural Conflicts*, Res. Report , EPFL, Lausanne (CH), August 1990. To appear in IEEE Trans. on Knowledge and Data Engineering, October, 1992

[21] 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

[22] 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