DATABASE REVERSE ENGINEERING
Models, techniques and strategies

Jean-Luc HAINAUT
Institut d'Informatique, University of Namur,
rue Grandgagnage, 21 - B-5000 Namur (Belgium)

0. Foreword

This text is a short presentation of a tutorial that was presented at the 10th International Conference on the Entity-Relationship Approach, held in San Mateo (California), in October 1991.

Part of the material the lecture is related with the PHENIX research project developed jointly by the University of Namur[1] and the BIKIT[2]. The objective of the project is to develop an expert-system approach to database reverse engineering. This project is supported by IRSIA, a Belgian public research agency, and by industrial partners[3].

1. Motivation

Reverse engineering a piece of software consists in reconstructing its functional and technical documentation, starting mainly from the source text of the programs. 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, reverse engineering tries to answer the question : *what could be a possible specification of this implementation*. The problem is particularly complex with old and ill-designed applications. In this case, not only no documentation (if any) can be relied on, but the lack of systematic methodologies for 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.

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;

---

1 Institut d'Informatique : M. Chandelon, prof. F. Bodart, prof. J-L Hainaut, M. Joris, B. Mignon, C. Tonneau

2 Babbage Institute for Knowledge and Information Technology (associated with the Ghent University) : E. Cardon, F. Osaer, prof. J. Vandamme, R. Van Hoe, prof. M. Vanwormhoudt, P. Verscheure

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

- 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 can be more successful than trying to tackle 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 theories and techniques to make this entreprise more realistic.

The lecture analyzes in detail the problems of reverse engineering data structures in data-oriented applications. It proposes formal techniques and heuristics for solving the problem of recovering a possible conceptual schema of an existing database.

This document presents a short overview of some material that will be developed in the conference.

2. Understanding the forward engineering of data structures

Though database design has been one of the major theoretical and applied research domain of the last two decades, and though it can be considered as a fairly well mastered problem, we are faced here with files and databases[4] that have not been built according to well structured methodologies. Tackling the reverse engineering of a database requires 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.

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

- a database must satisfy a limited set of **requirements** such as : correctness, time performance, space performance, availability, reliability, compliance with a data manager, compatibility with organizational constraints, etc. Satisfying each kind of requirement constitutes an identifiable **problem**.
- solving each of these 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 solving the same kinds of problems; therefore, designing a database consists in carrying out a **limited set of processes**; in some unstructured design approaches, some processes can be bypassed 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 giving output products; the transformation consists in adding, removing and changing the format of the input specifications;

These assumptions are the basis for a *theoretical generic model of database design activities* that gives us some useful hints on how to conduct the reverse engineering of an existing

4 From now on, the term *database* will refer to any permanent, structured, data collection on secondary storage, including standard files.

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 single step.

In the limited scope of this lecture, we shall consider a simplified generic model of database design. Figure 2.1 gives the organization of the main design processes, while Figure 2.2 shows the complementary view of the design products.

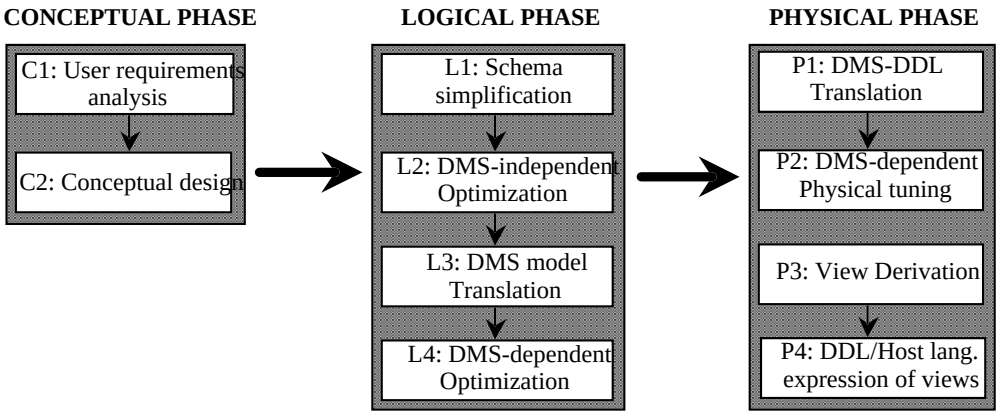

Figure 2.1 - *Some important design processes of database design. The processes have been classified into the conceptual, logical and physical phases.*

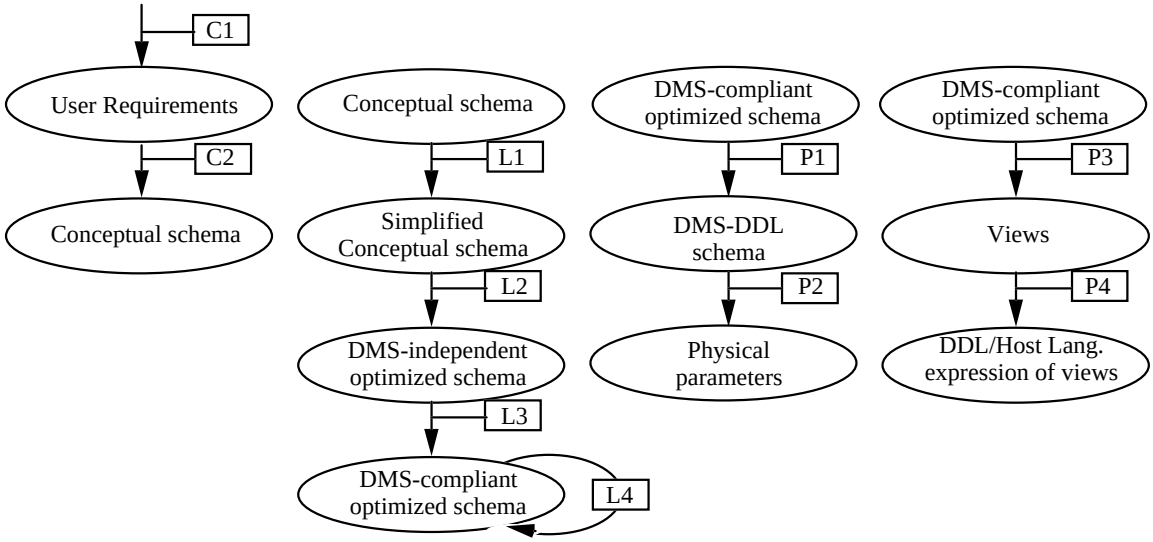

Figure 2.2 - *Some important design products developed during database design. The processes are identified according to the conventions of Figure 2.1.*

According to this model, the processes that can be carried out and the products they transform and produce are as follows :

Conceptual phase : user requirements are collected, analyzed (C1) and formalized (C2) into a conceptual schema.

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 first optimized independently of

the target DBMS (L2); it is then translated according 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 of the DMS (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. Where, why and how has a conceptual schema been degraded

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 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.

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 (e.g. conversion E-R --> relational). The schema is still less readable. On the other hand, due to the limited semantic expressiveness of older (and even current) DMS, this 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. As for process L2, it makes the schema less readable.

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 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 of the conceptual specifications. Another phenomenon must be pointed out, namely **structure hiding**. When translating a data structure into the 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.

Let's also observe that the DMS-DDL schema is not always materialized. Many standard file managers, for instance, do not offer any central way to memorize the structure of files.

Process P2 : works at a lower 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.
Let's make an important 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 files a particularly challenging exercice.

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. Database reverse engineering - A general approach

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),
- 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 :
- retrieve the existing data structures from their DDL/host language expression, and
- retrieve a possible conceptual schema that defines the semantics that underlies these data structures.

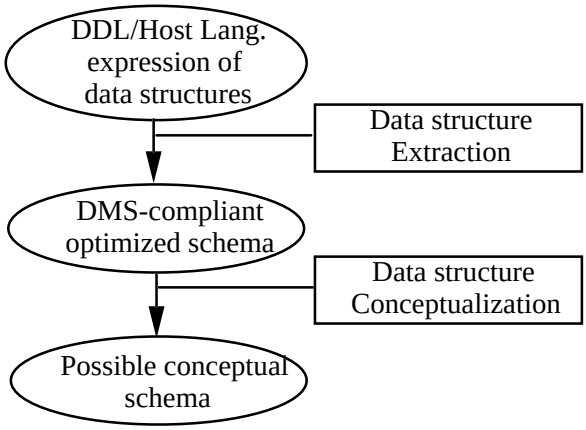

Figure 4.1 - *Gross organization of database reverse engineering activity.*

The first process is the reverse of the physical phase, and can be named *Data structure extraction*. The second one is the reverse of the logical phase, and can be 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.

In low level DMS (e.g. standard file managers), the starting point is generally 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.

5. Data structure modelling for reverse engineering

Before going into deeper details we need supporting models to express data structures at the different level 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.

This model is based on the Entity-Relationship model. It is intended first to express **conceptual structures**[5] by means of the following concepts :
- entity type (ET) and n-ary relationship type (RT)[6],
- ET- and RT-attribute[2],
- ET-, RT- and attribute identifier,

---

5 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.

6 According to the schema level, an entity type will be interpreted differently. For instance, it will describe a relational table in an SQL database, a record type in COBOL file or in a CODASYL database or a segment type in an IMS database. Similarly, an attribute will be the abstraction of a field, column, etc, in a DMS schema, and a relationship type will be the abstraction of a CODASYL set type, IMS parent/child relationships, or an ADABAS file coupling.

- inter- and intra- ET, RT and attribute integrity constraints, such as inclusion, exclusion, redundancy, implication,
- etc.

In addition, this model has been given constructs allowing the definition of access paths, access keys (abstraction of indices, calc keys, etc), file, etc, in order to express more technical data structures, such as those found in DMS-compliant optimized schemas for instance.

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 complete, formalized, model of the 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 :

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

Retrieving the DMS data structures

The difficulty of the first problem depends 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. In lower-level DMS, such as standard file managers, the problem can be more complex. Indeed, the global schema, generally file and record structures, is not under the control of the DMS. 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 a view of the global schema.

Two difficulties must be solved when carrying out this process : (1) an application program uses several files, only a subset 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 or analyzing the entry forms and the output reports/forms.

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. These

7 However, these files can give interesting hints on the schema when they store data from the database.

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 amount of source code. Among the main contraints the texts have to be searched for, let's mention the referential contraints (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), and redundancy constraints.

Integrating views

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

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

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.

Elimination of DMS-specific optimization constructs

The schema is examined for optimization constructs that are specific to the origin DMS[8]. A deep knowledge of 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.

***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 origin leads 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.

---

8 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.

Elimination of DMS-independent optimization constructs[5]

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

Expressing the schema in a higher-level model

Eliminating optimization-oriented structures and retrieving the conceptual origin of each construct of the DMS schema can produce a schema that still is akward 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 a 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.

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. General techniques

8.1 Schema transformation

Schema transformation is the basic tool in many database design activities. A general discussion of the concept can be found in [HAINAUT,91]. Roughly 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. there 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 garanteeing semantic equivalence have been used, reverse engineering their resulting schema can be done by using the inverse 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.

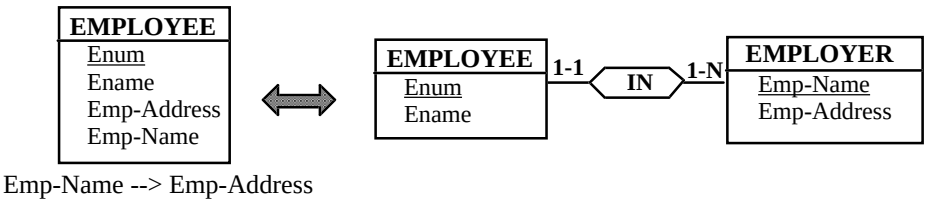

Figure 8.1 - *Read from right to left, the transformation denormalizes the schema, leading to better access performances, or decreasing the number of entity types for instance. It is typically a design transformation that can be used in activities L2 and L4. Read from left to right, this transformation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

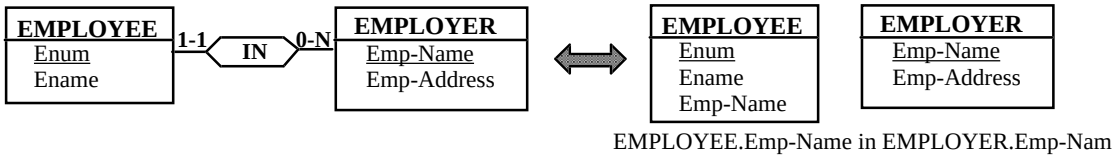

Figure 8.2 - *Read from left to right, this transformation allows the representation of many-to-one relationship types with reference attributes, a construct that is compatible with relational d_atabases and standard files (used in activity L3). Read from right to left, it allows the explicitation of relationship types in a reverse engineering activity.*

It can be shown that a fairly small number of standard transformations can explain most DMS-compliant schemas encoutered in practice. Understanding them, and relating each of them with the model of the target DMS is essential for reverse engineering complex schemas.

8.2 Data redundancy

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 (ET or RT) as well. Example : see 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 N schemas, several integration strategies have been proposed : merging two schemas at a time or all the schema in parallel, producing a new schema or augmenting one of the source schema. Integration is generally carried out in four steps (in case of binary integration) :

- preparation of the schema (optional) : restructuration of some schema in order to make the merging easier; in some techniques, it allows to automate the next two steps;
- correspondence : an object in one schema is related to an object 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 object, etc;
- merging : the objects in correspondence are merged, together with their relationships with the other objects 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. Applications

Many of the problems analyzed and of the techniques presented above are applicable whatever the DMS. However, they can be the basis of specific methods, each dedicated to a particular DMS.

These methods are elaborated through an analysis of the main features of the design process. In particular, the problem of generating DMS-compliant schemas (process L3) and optimizing them (L4) for a selected set of popular DMS will be examined. Rules and methods for reverse engineering schemas expressed in these DMS will be derived.

10. Conclusions

Except in oversimplistic, or accidentally favorable situations, reverse engineering a database from source DDL/Host language texts is a complex task that still needs in-depth research. Understanding the design methodologies, together with their underlying techniques and reasonings, is a considerable asset in this process, mainly because it helps identify the main design processes that have shaped 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 lecture establishes a general framework for understanding practical (i.e. mainly intuitive and non formal) design methodologies and to analyze the possible ways a conceptual schema could have been transformed into the observed schema.

It must be clear that reverse engineering cannot be a 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, file contents, data dictionary, etc) are generally used to support the source text analysis.

11. References

Reverse engineering

ROCK,90 Rock-Evans, R., Hales, K., *Reverse Enginering : Markets, Methods and Tools*, OVUM report, 1990

Database reverse engineering

CASANOVA,84a 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

DAVIS,85 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

NILSSON,85 Nilsson,E., G., *The Translation of COBOL Data Structure to an Entity-Relationship Type Conceptual Schema*, in Proc. of Entity-Relationship Approach, October, 1985

SPRING,90 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

WINANS,90 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

Schema integration

BATINI,83 Batini, C., Lenzerini, M., Moscarini, M., *View integration, in Methodology and tools for data base design*, Ceri, S., (Ed.)North-Holland, 1983

BATINI,86 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

BOUZEGHOUB,90 Bouzhegoub, M., Comyn, I., *View Integration by Semantic Unification and Transformation of Data Structures*, in Proc. of Entity-Relationship Approach : the Core of Conceptual Modelling, pp. 413-430, October, 1990

JARDINE,89 Jardine, D., A., Yazid, S., *Integration of Information Submodels*, in Proc. of Information System Concepts : An In-depth Analysis, pp. 247-267, October, 1989

MOTRO,87 Motro, *Superviews : Virtual Integration of Multiple Databases*, in IEEE Trans. on Soft. Eng., Vol. SE-13, No 7, July, 1987

NAVATHE,84 Navathe, S., B., Sashidar, T., Elmasri, R., A., *Relationship Merging in Schema Integration*, in Proc. of Very Large Databases, 1986

SPACCA,90a Spaccapietra, S., Parent, C., *View Integration : A Step Forward in Solving Structural Conflicts*, Res. Report , EPFL, Lausanne (CH), August, 1990

SPACCA,91 Spaccapietra, S., Parent, C., Dupont, Y., *Automating Heterogeneous Schema Integration*, EPFL, Lausanne (CH), February, 1991

Schema transformation

BERT,85 Bert, M., N., al., *The logical design in the DATAID project : the EASYMAP system*, in Computer-Aided Database Design : the DATAID Project, Albano, al., (Ed.)North-Holland, 1985

BOUZEGHOUB,90 Bouzhegoub, M., Comyn, I., *View Integration by Semantic Unification and Transformation of Data Structures*, in Proc. of Entity-Relationship Approach : the Core of Conceptual Modelling, pp. 413-430, October, 1990

DATRI,84 D'Atri, A., Sacca', D., *Equivalence and Mapping of Database Schemes*, in Proc. of Very Large Databases, pp. 187-195, August, 1984

GIRAUDIN,85 Giraudin, J-P., Delobel, C., Dardailler, P., *Eléments de construction d'un système expert pour la modélisation progressive d'une base de données*, in Proc. of Journées Bases de Données Avancées, Mars, 1985

HAINAUT,81 Hainaut, J-L., *Theoretical and practical tools for data base design*, in Proc. of Very Large Databases, pp. 216-224, September, 1981

HAINAUT,90b Hainaut, J-L., *Entity-Relationship models : formal specification and comparison - Tutorial*, in Proc. of Entity-Relationship Approach : the Core of Conceptual Modelling, pp. 433-444, 1991

HAINAUT,91a Hainaut, J-L., *Entity-generating Schema Transformation for Entity-Relationship Models*, in Proc. of the 10th Conf. on Entity-Relationship Approach, 1991

KOBAYASHI,86 Kobayashi, I., *Losslessness and Semantic Correctness of Database Schema Transformation : another look of Schema Equivalence*, in Information Systems, Vol. 11, No 1, pp. 41-59, January, 1986

KOZACZYNSKY,87 Kozaczynsky, Lilien, *An extended Entity-Relationship (E2R) database specification and its automatic verification and transformation*, in Proc. of Entity-Relationship Approach, 1987

LING,89 Ling, T., W., *External schemas of Entity-Relationship based DBMS*, in Proc. of Entity-Relationship Approach : a Bridge to the User, 1989

REINER,86 Reiner, D., Brown, G., Friedell, M., Lehman, J., McKee, R., Rheingans, P., Rosenthal, A., *A Database Designer's Worbench*, in Proc. of Entity-Relationship Approach, 1986

ROSENTHAL,88 Rosenthal, A., Reiner, D., *Theoretically sound transformations for practical database design*, in Proc. of Entity-Relationship Approach, 1988

SCHOLL,90 Scholl, M., H., Schek, H-J., *A Relational Object Model*, in Proc. of ICDT'90, pp. 89-105, Dec, 1990

Database design

BRAGGER,84 Brägger, R., P., Dudler, A., Rebsamen, J., Zehnder, C., A., *GAMBIT : An Interactive Database Design Tool for Data Structures, Integrity Constraints, and Transaction*, in Proc. of Data Engineering, pp. 399-407,1984

DATAID,83 *Methodology and tools for data base design*, Ceri, S., (Ed.), North-Holland, 1983

DATAID,85 *Computer-Aided Database Design : the DATAID Project*, Albano, al., (Ed.), North-Holland, 1985

ELMASRI,89 Elmasri, R., A., Navathe, S., B., *Fundamentals of Database Systems*, Benjamin/Cummings, 1989

LYNGBAEK,86 Lyngbaek, P., Kent, W., *A Data Modeling Methodology for the Design and Implementation of Information Systems*, in Proc. of Object-Oriented Database Systems, pp. 6-17, 1986

OLLE,82 *Information Systems Design Methodologies : a Comparative Review*, Olle, W., Sol, H., Verrijn-Stuart, A., (Ed.), North-Holland, 1982

REINER,86 Reiner, D., Brown, G., Friedell, M., Lehman, J., McKee, R., Rheingans, P., Rosenthal, A., *A Database Designer's Worbench*, in Proc. of Entity-Relationship Approach, 1986

SCHIEL,84 Schiel, U., Furtado, A., L., Neuhold, E., J., Casanova, M., A., *Towards Multi-level and Modular Conceptual Schema Specifications*, in Information Systems, Vol. 9, No 1, pp. 43-57, , 1984

TEOREY,86 Teorey, T., J., Yang, D., Fry, J., P., *A Logical Design Methodology for Relational Databases using the Extended Entity-Relationship Model*, in ACM Comp. Surveys, Vol. 18, 2, No , pp. 197-222, 1986

ULLMAN,89 Ullman, J., D., *Principles of Data- and Knowledge-base Systems* (Vol I & II), Computer Science Press, 1989