SCHEMA TRANSFORMATION TECHNIQUES FOR DATABASE REVERSE ENGINEERING

J-L. Hainaut, C. Tonneau, M. Joris, M. Chandelon[1],

ABSTRACT - The paper presents a DBMS-independent database reverse engineering (DBRE) methodology based on a generic process model and on transformation techniques. DBRE is proposed as a two-phase process consisting in recovering the DBMS-dependent data structures (data structure extraction) then in recovering their semantics (data structure conceptualization). The second phase, that is strongly linked with the logical design phase of current database design methodologies, can be performed by application of a selected set of standard schema restructuring techniques, or schema transformations. The paper illustrates the methodology by applying it to various DBRE processes : removing optimization structures, *untranslating* Relational, COBOL, CODASYL, TOTAL/IMAGE and IMS database as well as file structures, and finally conceptual normalization.

KEYWORDS : database design, dabase reverse engineering, schema transformation, schema equivalence

1. INTRODUCTION

Reverse engineering a piece of software consists, among others, in recovering or reconstructing its functional and technical specifications, starting mainly from the source text of the programs [17] [16]. Recovering these specifications is generally intended to redocument, convert, restructure, maintain or extend old applications. It is also required when developing a Data Administration function that has to know and record the description of all the data existing in the organization The problem is particularly complex with existing, ill-designed applications. In this case, not only no decent 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. 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 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 structure* that underlies the file structures is mainly procedure-independent (though their *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 is most often 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.

The literature proposes systematic approaches for database schema recovering, but not as many as could be expected when considering the current needs in industry. Let us mention [2], [25], [4] for standard files, [23], [34] for IMS databases, [1] for CODASYL databases and [3], [23], [5], [30], [7], [26] for relational databases. Even some recent popular text books include sections dedicated to DBRE [1].

Most of these studies, however, appear to be limited in scope, and are generally based on assumptions on the quality and completeness of the source data structures to reverse engineer that cannot be relied on in many practical situations. For instance, they often suppose that,

- all the conceptual specifications have been translated into data structures and constraints,
- the translation is rather straightforward (no tricky representations); [26] is one of the only proposals that process some non trivial 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).

In many proposals, it appears that the only databases processable are those that have been obtained by a rigourous database design method. This condition cannot be assumed for many operational databases, particularly for the

[1]Institut d'Informatique, Univ. of Namur, rue Grandgagnage, 21, B-5000 Namur (Belgium) - jlhainaut@info.fundp.ac.be

oldest ones. Moreover, these proposals are most often dedicated to one data model and do not attempt to elaborate techniques and reasonings common to several models, leaving the question of a general DBRE approach still unanswered.

Concerning the CASE tools, an increasing number of products offer DBRE functionalities (let us only mention the Bachman re-engineering toolset). 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 [28].

This paper presents some aspects of a rigourous framework in which DBRE can be studied with greater generality. In particular, this framework encompasses all the standard database models currently in use and it can cope with empirically designed databases. First, the paper develop a schema specification model that is able to describe both conceptual and technical data structures (Section 2). Section 3 analyses the concept of schema transformation and describes some representative techniques. Section 4 proposes to revisit the database design activity as a sequence of transformations in order to understand how a conceptual schema can be translated into executable descriptions. In section 5, a generic model of database reverse engineering is proposed. This model relies heavily on schema transformation techniques. Then the paper describes schema transformations to cope with optimization of data structures (section 6). Section 7 examines how conceptual structures can be translated according to the specific model of a data management system (DMS[2]) and proposes techniques to recover the origin structures from relational, COBOL, CODASYL, TOTAL/IMAGE and IMS schemas. Section 8 proposes schema transformations dedicated to conceptual normalization.

2. SCHEMA SPECIFICATION MODEL

Database design and reverse engineering are concerned with building, converting and transforming database schemas at different levels of abstraction. Elaborating DMS-independent and even methodology-independent techniques and reasonings that support these activities requires the availability of a set of models to express all these schemas. Due to the transformational approach adopted in this presentation, and due to the large scope of the proposal that encompasses all the traditional levels of abstraction, it has been found essential to base it on a unique schema specification model. This model and its transformational operators are intended,

- to support forward as well as reverse engineering,
- to express conceptual, logical and physical schemas, as well as their manipulation,
- to support any DMS model and the production and manipulation of their schemas.

In short, conceptual schemas as well as physical schemas are expressed into a unique, generic, *extended entity-relationship model*. To make the presentation clearer, we shall distinguish two layers of features in the model, comprising the conceptual and technical features respectively. Here follows a brief description of the concepts of the model. A more comprehensive specification can be found in [13]; formal foundations of most aspects of the model are proposed in [10].

The **conceptual layer** consists of the features of the standard entity-relationship model with some extensions. It includes the following concepts, most of them being illustrated in Fig. 2.1 :

- *entity type*, comprising any number (including zero) of attributes;
- *IS-A hierarchy* (Fig. 8.3),
- *relationship type* (called *rel-type* from now on), comprising two or more roles and any number of attributes; a role is taken by one or several entity types (multi-ET role), and is given a cardinality constraint `[min-max]` that states the minimum and maximum number of relationships in which any entity takes this role;
- an *attribute* is either atomic or compound; an atomic attribute has a domain of values; each attribute is given a cardinality constraint[3] `[min-max]` stating how many values can be associated with its parent object (i.e. entity type, rel-type, compoud attribute); a multivalued attribute (cardinality `max` > 1) can be pure (set of values), bag (multiset of values) or list (indexed set or multiset);
- an entity type can have any number of (candidate) *identifiers*, including zero; an identifier is made of attributes and/or roles (i.e. connected entity types)[4]; one of the identifiers can be declared primary;
- a rel-type has at least one identifier made of roles and/or attributes; any role with cardinality `[min-1]` is an identifier; when no identifier is specified or can be deduced, then all the roles of the rel-type form its identifier;

---

2 A DMS can be either a File Management System (FMS) or a Database Management System (DBMS).

3 Note that this constraint allows for the specification of optional/mandatory attributes as well as single-valued/ multivalued attributes. In addition using this constraint for both roles and attributes stresses their duality and simplifies greatly many transformations.

4 In [1], identifiers are called *internal* when they comprise attributes only, *external* when they include roles only, and *mixed* when they are made of both.

- a pure or list multivalued attribute can be given an identifier which is a subset of its (grand-) children components;
- integrity constraints can be associated with these constructs; let us mention inclusion constraint (Fig. 3.7), referential constraint[5] (Fig. 3.8), redundancy constraint (Fig. 6.1), exclusion constraint, coexistence group (A group of attributes and/or roles of an entity type whose values are simultaneously present or absent.) (Fig. 8.4) and functional/multivalued dependency (Fig. 3.2 and 3.6).

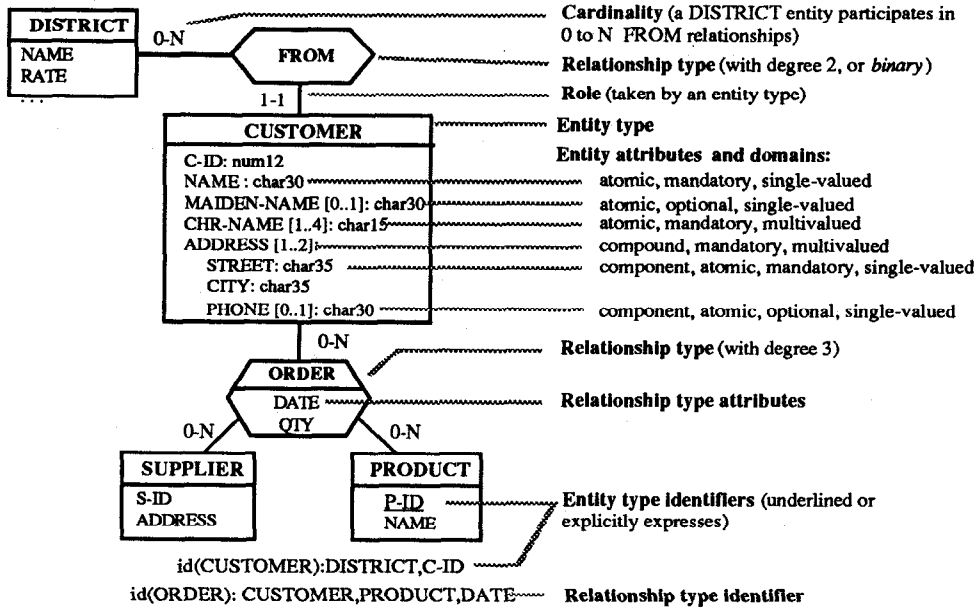

Figure 2.1 Graphical representation of some conceptual constructs.

The **technical layer** includes constructs that pertain to the description of logical and physical data structures (Figure 2.2). These features are additions to the conceptual layer described hereabove. Some of them are described briefly.

- a file is a repository of entities; at this level, an entity is the abstraction of a record, segment or row;
- an access key is a goup of attributes with which an access mechanism is associated; it is the abstraction of value-based access mechanisms such as indexes, hash files, etc;
- an access path is an access mechanism allowing the navigation through rel-types; it allows to specify set types (CODASYL) and parent-child relationships (IMS) for instance;
- the entities in a file, the entities linked with an entity through a rel-type and list multivalued attribute values can be ordered according to insertion time or to a sort-key;
- an attribute has a physical length, depending on its encoding scheme, and a physical position in its parent object. Two attributes can share the same physical position.

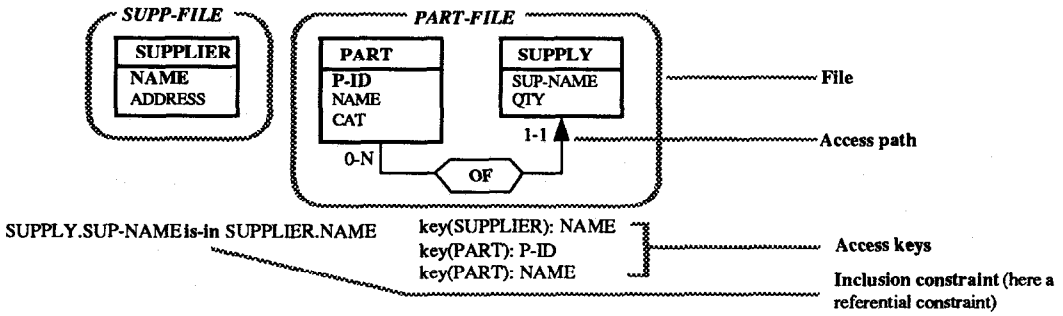

Figure 2.2 Graphical representation of some technical constructs together with inclusion constraints

[5] This is a special case of inclusion constraint that is not based on the concept of primary key; it only requires the presence of a (candidate) identifier in the target entity type. This allows a greater simplicity in the reverse engineering untranslation algorithms as compared with proposals such as [23] and [7].

This model must be considered as a generic specification model that can be specialized into a great variety of submodels, for instance according to the design levels or design product classes of a standard or user-defined multilevel design method, or according to the target DBMS. Specializing the generic model into a specific submodel can be done by stating the set of rules the schema must satisfy. Some examples can be found in [13], including the rule sets that defines standard ER model and ORACLE v.5 model.

3. SCHEMA TRANSFORMATION

3.1 PRINCIPLES

Schema transformation is a ubiquitous concept in database design. Proving the equivalence of schemas [19], refining a conceptual schema [1] [24], integrating two partial schemas [1], producing a DBMS-compliant schema from a conceptual schema [27], restructuring a physical schema [22] [1], DB reverse engineering [12] [14], are basic design activities that can be carried out by carefully chosen schema transformations . However, it has not been often studied as an design tool of its own. Let's only mention [19], [20], [29], [11] as some proposals in this direction. However, things seem to change at the practice level, for instance in text books. [1] is one of the first popular references that present schema transformation as a basic concept in database design. In the NIAM world, [24] includes a chapter on conceptual schema transformations. Though developing this concept and its formalization is beyond the scope of this document, we shall summarize the material presented in [11] to sketch the main definitions and properties that will be important from the methodological viewpoint.

A **transformation** Σ consists of mappings **T** and **t** :

- **T** is a *structural mapping* that replaces source construct C in schema S by construct C'; C' is the target of C through T, and is noted **C' = T(C)**. In fact, C and C' are classes of constructs that can be defined by structural predicates. T is therefore defined by a *minimal precondition* ***P*** that any construct C must satisfy in order to be transformed by T, and a *maximal postcondition* ***Q*** that T(C) satisfies. T can therefore be written **T = <P,Q>**. P and Q are pattern-matching predicates that identify the components and the properties of C and T(C), and more specifically : the components of C that are preserved in T(C), the components of C that are discarded from T(C), the components of T(C) that do not exist in C. T is the *syntax* of the transformation.
- **t** is an *instance mapping* that states how to produce the T(C) instance that corresponds to any instance of C. If c is an instance of C, then c' is the corresponding instance of T(C), noted **c' = t(c)**. t is the *semantics* of the transformation.

According to the context, Σ will be noted either **<T,t>** or **<P,Q,t>**.

3.2 REVERSIBILITY

The notion of *reversibility* is an important characteristic of a transformation. If a transformation is reversible, then the source and the target schemas have the same descriptive power, and describe the same universe of discourse, although with a different presentation (syntax). Reversible transformations are also called *semantics-preserving*.

A transformation $\Sigma 1$ = <P1,Q1,t1> = <T1,t1> is *reversible, iff* there exists a transformation $\Sigma 2$ = <P2,Q2,t2> = <T2,t2> such that, for any construct C, and any instance c of C,

$$P1(C) \Rightarrow [T2(T1(C))=C] \text{ and } [t2(t1(c)=c]$$

$\Sigma 2$ is the inverse of $\Sigma 1$, but not conversely. For instance, an arbitrary instance c' of T(C) may not satisfy the property c'=t1(t2(c')). All the transformations that will be presented and discussed in this paper are reversible.

If $\Sigma 2$ is reversible as well, then $\Sigma 1$ (as well as $\Sigma 2$) is called *symmetrically reversible*. In this case, $\Sigma 2$ = <P1,Q1,t2> , and both transformations can be defined through the unique notation Σ = <P,Q,t1,t2>. Σ is called an *SR-transformation*. This is the most desirable form of transformations, and most techniques that will be described in this paper have this *SR-property*[6]. However, in database design, and particularly at the implementation level, non fully reversible transformations are often used due to the unavailability of SR-transformations. There exist two ways to *degrade* an SR-transformation into a simply reversible one :

---

[6] [1] gives another definition of reversibility in terms of information equivalence of source and target schemas : "schemas S1 and S2 have the same information content (or are *equivalent*) if for each query Q that can be be expressed on S1, there is a query Q' that can be expressed on S2 giving the same answer, and vice versa." It is easy to prove that SR-transformations are mappings between schemas that have the same information content according to this definition.

generalization and specialization. *Generalization* consists in choosing a target construct that describes more general instances than those allowed by the source schema. Representing a one-to-one rel-type by a one-to-many rel-type is common practice in CODASYL and IMS schemas for example. *Specialization* consists in choosing a target construct that offers more properties than required by the source construct. For instance, a multivalued attribute will generally be represented by a list attribute in COBOL and CODASYL data structures (Fig. 7.5).

3.3 NOTATION

A PROLOG-like expression of P and Q is suggested in [13]. However, we shall use a more readable expression of the transformation in which generic versions of C and T(C) are represented through ER graphical convention.

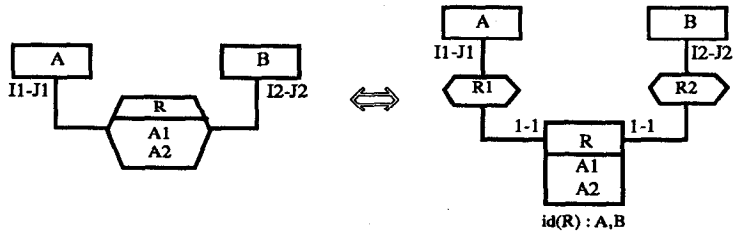

***Figure 3.1** Representation of structural mapping T1 & T2 of a typical SR-transformation.*

Figure 3.1 illustrates this representation by describing how a *many-to-many* rel-type that has attributes (i.e. the C construct) can be replaced by a new entity type that inherits these attributes, by two *one-to-many* rel-types, and by an identifier (i.e. construct T(C)). The schema shows how all the constructs and properties of the source schema are precisely defined : names and cardinality constraints for instance. It identifies also the constructs that are kept (entity types A,B), that are discarded (rel-type R and its attributes), and that are new (entity type R, rel-types R1 and R1, attributes A1 and A2) (ee [11] for more details). This transformation is generic since names A, R, A1, .., A2, R1, R2, I1, J1, I2, J2 must be replaced by actual values (CUSTOMER, ORDER,QTY,.., 0, 1) in order to get a specific transformation on an actual schema. Expression of instance mapping t depends on the model in which the schemas are expressed. In the relational model, algebraic or calculus expressions will be used. In ER models, we could use ad hoc algebra or ER-SQL. In DMS models, procedural expression will be adequate. From now on, we shall ignore the t part of the transformations.
More precisely, the notation will be as follows.

C ⇒ C' : describes how construct C is transformed into construct C' = T(C). Transformation <T,..> is reversible, but its inverse is not.

C ⇐ C' : describes how construct C' is transformed into construct C = T(C'), and therefore how C' *may be* (without garantee) the target of C. Transformation <T,..> is reversible, but its inverse is not. Replacing C by C' will be carried out in heuristics but should be done very carefully since C' is more constrainted (i.e. bears more semantics) than C.

C ⇔ C' : describes how construct C is transformed into construct C' = T(C). Transformation <T,..> is symmetrically reversible. The preferred reading of this notation is from left to right. When reading from right to left is meant, we shall use the qualifier *reverse*.

3.4 EXAMPLES OF PRACTICAL SR-TRANSFORMATIONS

It can be proved that most schema transformations that are used in practical design activities (including DBRE) derive from three generic transformations, namely *Project-Join*, *Extension* and *Identifier substitution* that are described in [11] and [15]. They will be discussed informally through some of their most useful ER specializations, presented as neutral techniques independently of the objectives they may satisfy and of the activities in which they may be useful : forward/reverse, conceptual/logical/physical. Their axiomatization will be presented in a future paper.

Project-Join transformations

The relational project-join decomposition [ULLMAN,88] can be applied to ER structures as well. Indeed, they can be given a relational expression that allows them to inherit the concepts and reasonings of the relational theory [10]. Here follows an example of the P/J transformation applied to an unnormalized rel-type (3.2) The attribute/role components (E1, E1, E3, A1, A2, A3) of R are decomposed into the two subsets (**E1, E3**, A1) and (**E1, E3**, E2, A2, A3) following the functional dependency E1,E3 ⟶ A1. The first subset produces rel-type R1

while the second one produces rel-type R2.

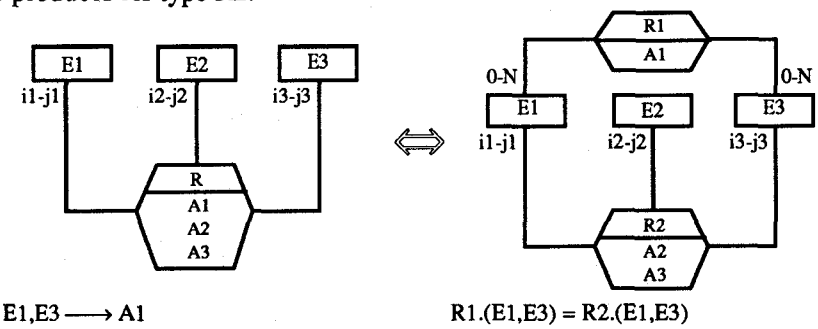

***Figure 3.2** - Project-Join transformation of an unnormalized rel-type*

Extension transformations

This transformation, which has been described in [11] has a very broad field of application. Grossly speaking, the extension transformation consists in extracting a subset of attributes and/or roles of a rel-type or of an entity type, and in replacing them by a new *surrogate* entity type. This new entity type is linked to the components it replaces. In 3.3, rel-type R has components[7] (`A1, A2, A3, E1, E2, E3`). Let us consider the subset (`E2, E3, A2`), extract them from R, and replace them by new entity type `X` (R becomes S). Entity type `X` is linked to `E1` (via `T1`), `E2` (via `T2`) and `A2` (as a proper attribute).

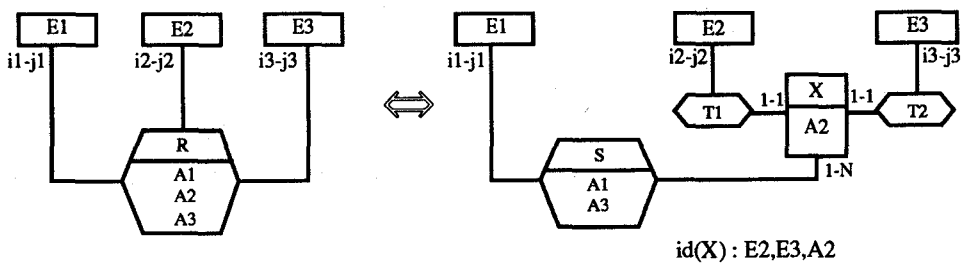

***Figure 3.3** - Extension transformation of a rel-type by explicit representation of some of its components*

Representing a rel-type with an entity type is a popular version of this transformation (3.4). It can be applied to N-ary, many-to-many, one-to-many, one-to-one or recursive rel-types as well. Rel-type `R` disappears since it is completely represented by `X`.

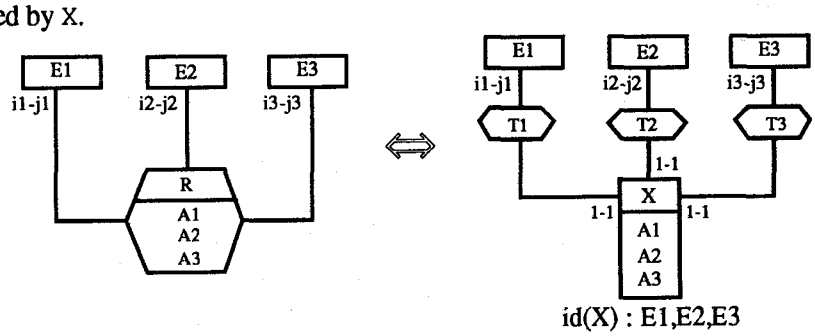

***Figure 3.4** - Extension transformation of a rel-type by explicit representation of all its components*

Transforming an attribute into an entity type is another standard application. It comes in two flavours, namely **instance representation** (3.5(a)), in which each instance of A2 in each E entity is represented by an EA2 entity, and **value representation** (3.5(b)), in which each value of A2, whatever the number of E entities in which it appears, is represented by an EA2 entity.

---

7 The components of a rel-type comprise its roles and its attributes.

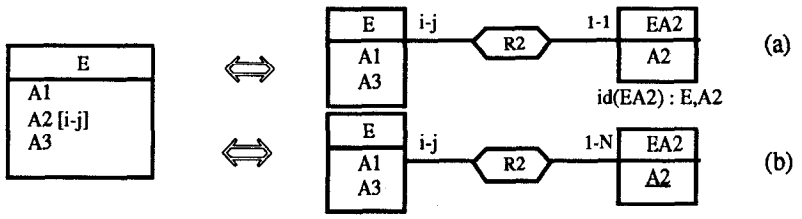

***Figure 3.5** - Extension transformation of an entity type by (a) explicit representation of the **instances** of one of its attributes, and (b) explicit representation of the **values** of one of its attributes.*

A last example is the normalization of an entity type (3.6). Components (`R1.A, B2, B3`)[8] are extracted from entity type B.

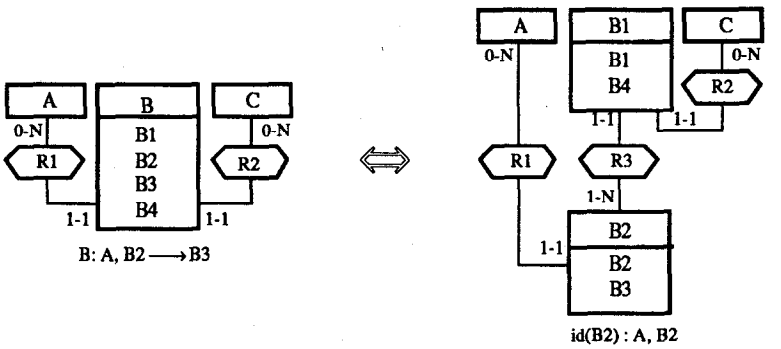

***Figure 3.6** - Extension transformation of an unnormalized entity type*

Identifier substitution

A role of a rel-type is replaced by an identifier of the participating entity type [9]. In 3.7, role S.C[9] has been replaced by the identifier of C, namely A and B. S can be further transformed through a P/J transformation, leading to the technique proposed in 7.12. A more general version, developed in [15] allows S to be many-to-many or N-ary.

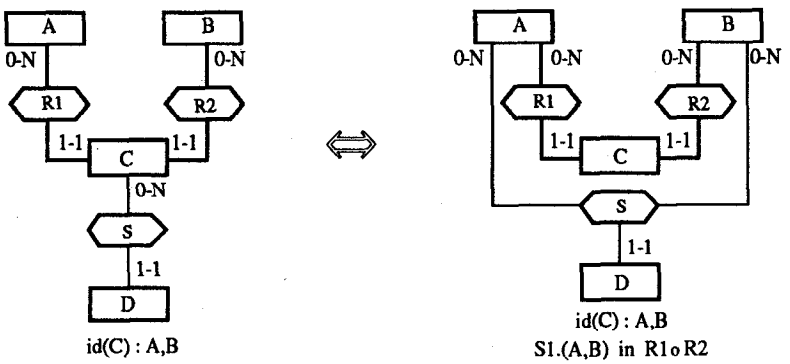

***Figure 3.7** - Role S.C is replaced by the identifier (A,B) of C.*

The best known application of identifier substitution is that of the *foreign key* (3.8), which can be either multivalued (cardinality J > 1) or single-valued (cardinality J = 1).

---

8 The components of an entity type are its attributes and the other roles of binary rel-types in which it takes a role with cardinality i-1.

9 I.e. role C of rel-type S. The default name of a role is that of the participating entity type.

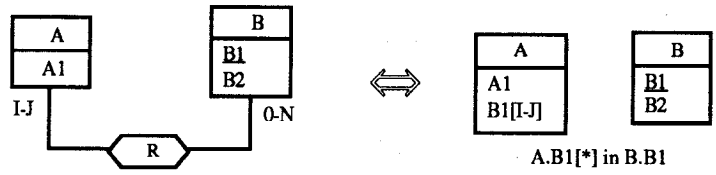

Figure 3.8 - The traditional representation of a rel-type by a foreign key : role R.B is replaced by id B1 of B.

4. DATABASE DESIGN REVISITED

Database design is a process that *transforms* users requirements (as well as technical requirements) into, a.o., database schemas. Traditionally, database design has been described as made up of a sequence of four specific processes[10], namely CONCEPTUAL DESIGN, LOGICAL DESIGN, PHYSICAL DESIGN and VIEW DESIGN [31] [1].

- The CONCEPTUAL DESIGN process is aimed at producing the *conceptual schema*, a computer-independent representation of the database. Among others, this representation is to be given such desirable qualities as normality, minimality, clarity, that can be considered as provided through a specific sub-process called CONCEPTUAL NORMALIZATION.
- Through the LOGICAL DESIGN process, the conceptual schema is transformed into a *DMS-compliant optimized logical schema* with the following two characteristics : (1) it follows the data model of the chosen DMS, (2) it satisfies operational criteria such as space and time performance[11]. This process can be considered as being refined, at least conceptually, into three subprocesses : schema simplification, schema optimization and DMS-translation.
  - SCHEMA SIMPLIFICATION replaces *advanced constructs* such as IS-A hierarchies, N-ary rel-types by equivalent basic constructs. The Bachman Data Structure Diagram is a popular target model of this subprocess. The simplified conceptual schema is perceived as an adequate medium for logical reasonings when traditional DMS are considered.
  - SCHEMA OPTIMIZATION modifies the schema in order to give it better performances. Three families of techniques are commonly used, namely unnormalization, structural redundancy and restructuration (e.g. vertical and horizontal fragmentation).
  - SCHEMA TRANSLATION converts the schema into data structures that are compliant with the model of the DMS. The result consists into two complementary parts : the DMS constructs, that can be controlled by the DMS, and the non-DMS constructs, most often integrity constraints, that will be ignored by the DMS.
- The PHYSICAL DESIGN process translates the DMS constructs of this schema into a DMS-DDL text, and the non-DMS constructs into, e.g., procedural sections or local variables of the application programs. In addition, through physical tuning, technical parameters are set and physical constructs are built, such as indexes and clusters.
- The VIEW DESIGN process builds the external views required by users and programs, and translates them in DMS-DDL and application programs fragments in the same way as for the global schema.

Most of these processes can be in turn refined, directly or indirectly, into lower-level processes or operators that are primitive schema transformations as defined in section 3[12]. As an illustration, the following procedure, based on semantics-preserving transformations, can be a simply strategy for the SCHEMA TRANSLATION instantiated for standard relational structures.

1. *for each non-functional rel-type R, do : transform R into an entity type (apply 3.4);*
2. *while compound or multivalued attributes exist, do :*
   for each attribute A that is both single-valued and compound, and that depends directly on an entity type, do : disaggregate A (apply 7.6 (reverse));
   for each attribute A that is multivalued, and that depends directly on an entity type, do : transform A by instance representation (apply 3.5(a));
3. *until no rel-type can be transformed, do :*
   for each rel-type R, do : if 3.8 is applicable, then transform R into reference attribute (apply 3.8);

---

10 A process/product model for database design description has been proposed in [13].

11 There are other operational requirements such as *privacy*, *modularity* or *stability* against future evolution. However, the latter criteria will be ignored in this paper for simplicity.

12 This transformational top-down refinement model is explicitly advocated in [1], [9], [13] and [29] for instance.

4. *until no rel-types remain, repeat : for each rel-type R, do :*
   add an artificial attribute A to E, and make A the identifier of E;
   do 3;
5. *make each identifier and each foreign key an access key;*
6. *remove each access key that is a prefix of another access key;*

Conclusion. Most of the processes that have been described above can be ultimately defined as sequences of schema transformations. Since the logical and physical design processes are intended to preserve the contents of the conceptual schema, they are themselves semantics-preserving. Therefore, they must be based on SR-transformations. This conclusion is important when considering the DBRE problem. Indeed, the schema transformations on which DB design relies can be used as a sound basis for reverse engineering.

5. DATABASE REVERSE ENGINEERING

Described as a design process, DBRE transforms input products, that mainly consist of source code texts, into schemas of the database. Two schemas are of particular interest, namely the *logical schema of the database* according to the DMS model, and a possible *conceptual schema.* Concerning the final objective of DBRE, one or both of these schemas may be needed. The conceptual schema provides a high level description of the information contents needed for DMS conversion, interoperability or data administration for instance. The DMS schema is necessary for accessing or converting the data. The main input products consist in DMS-DDL description of the global schema and of the views, either in text format, or as data dictionary contents. Essential information, for instance on untranslated integrity constraints, can be found in the source text of the application programs (procedure and data structures), in screen layout and procedural components of the user interface and in database checking procedures (e.g. triggers, check clauses). The physical schema may yield useful hints concerning the logical schema (indexes suggesting foreign keys, join-based clusters, etc). File/database contents analysis may provide strong hints, or validate hypotheses, on the presence of such constructs as identifiers, foreign keys, field layout and value domains.

In [18] and [14], a general two-step procedure is proposed for reverse engineering any database or file collection. It will be briefly summarized in this section. The procedure suggests that DBRE be conducted as the execution of two processes (figure 5.1), namely DATA STRUCTURE EXTRACTION and DATA STRUCTURE CONCEPTUALIZATION. This organization is strongly advocated by three observations. First, each process yields one of the output schemas of DBRE, secondly they are the exact reverse of DB design processes, namely PHYSICAL DESIGN and LOGICAL DESIGN and thirdly, they are supported by different concepts, reasonings and techniques

- DATA STRUCTURE EXTRACTION produces a complete description of the data structures according to the model of the DMS, e.g. COBOL file structures, CODASYL schema, relational schema, etc. In addition, the non-DMS parts of the schema have been elicited from, a.o. the procedural parts of the applications or database contents. According to the DMS, the process can be more or less easy. For instance, the DMS part of COBOL data structures can be difficult to discover while CODASYL or relational schemas can be analysed quite easily. DATA STRUCTURE EXTRACTION appears as the inverse of the PHYSICAL DESIGN forward process. Due to the scope of this paper, we shall not develop this process any further (see [12] [18] or [14]).
- DATA STRUCTURE CONCEPTUALIZATION tries to make the semantics of the logical schema explicit by recovering the intention of the optimized DMS data structures. This process is the reverse of the LOGICAL DESIGN forward process, and can be decomposed into three subprocesses, namely *schema de-optimization, DMS-untranslation* and *conceptual normalization.*
  - SCHEMA DE-OPTIMIZATION detects and removes the non-semantic constructs from the logical schema, and particu-larly the optimization structures. This process appears as the inverse of the SCHEMA OPTIMIZATION forward process.
  - DMS-UNTRANSLATION detects DMS-compliant constructs and replaces them by their DMS-independent equivalent. This process appears as the inverse of the SCHEMA TRANSLATION forward process.
  - CONCEPTUAL NORMALIZATION has the same objectives as its forward engineering counterpart such as minimality and clarity. In particular, it is intended to recover the high-level structures transformed by the SCHEMA SIMPLIFICATION.

According to this approach, each DBRE process is either a forward process or the inverse of a forward process. Consequently, they can be based on basic schema transformations that are either forward transformations or the inverse thereof.

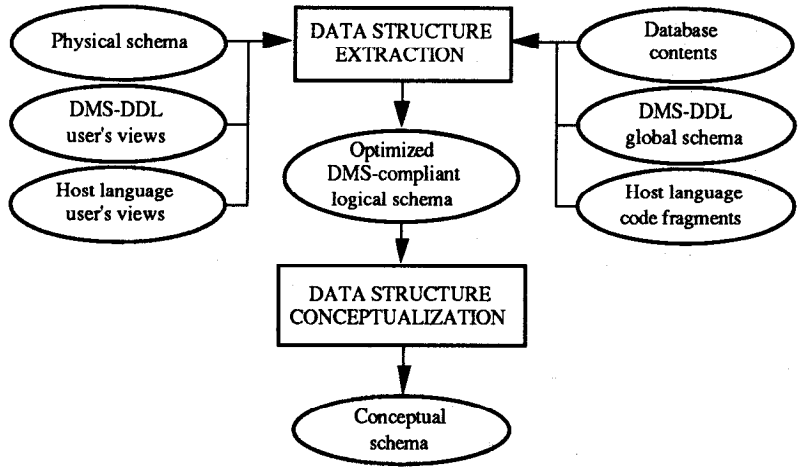

***Figure 5.1** - A global strategy for database reverse engineering.*

The following sections propose a collection of reversible schema transformations that can form the kernel of a database reverse engineering toolset. Chapter 6 examines how to discard and transform optimization constructs. Chapter 7 proposes some techniques to *untranslate* COBOL structures, relational schemas, CODASYL schemas, TOTAL/IMAGE schemas and IMS schemas. Chapter 8 discusses some conceptual normalization techniques. The problem of how to apply these transformations in each of these processes, i.e. the process strategies, is not adressed in this paper.

6. *DE-OPTIMIZATION* TRANSFORMATIONS

Both DMS-dependent and DMS-independent optimization processes will be considered as a whole, since they make use of the same set of transformations, though possibly through different strategies. There are three major families of optimization techniques based on schema transformations, namely *unnormalization*, *structural redundancy* and *restructuration*. They must be precisely understood in order to reverse their effect on a schema. In particular, some of them are more specifically fitted for some DMS than for others.

6.1 UNNORMALIZATION REVERSE TRANSFORMATIONS

The most common unnormalization technique consists in merging two entity types linked by a one-to-many rel-type into a single entity type. This technique allows obtaining the information of both entity types in one logical access, thereby decreasing the access time. On the other hand, it induces redundancies. An unnormalized structure is detected in entity type B by the fact that the determinant of a dependency F is not an identifier of B[13]. Reversing this transformation consists in *normalizing* the entity type by segregating its components according to 3.6. This technique has been discussed in [11].

6.2 STRUCTURAL REDUNDANCY REVERSE TRANSFORMATIONS

Structural redundancy techniques consist in adding new constructs in a schema such that their instances can be computed from instances of other constructs. Attribute duplication, rel-type composition and agregate representation are some examples of common optimization structural redundancies. These transformations are (trivially) symmetrically reversible since they merely add derivable constructs without modifying the source constructs. The reverse transformation consists in removing the redundant constructs. The main problem is to detect the redundancy constraint that states the equivalence of construct. Figure 6.1 depicts the elimination of a composed rel-type and of a duplicate attribute.

6.3 RESTRUCTURATION REVERSE TRANSFORMATIONS

Restructuration consists in replacing a construct by other constructs in such a way that the resulting schema yield better performance. These techniques introduce no redundancy.
Four popular techniques are vertical partitioning and horizontal partitioning of entity types as well as their inverse.
Horizontal partitioning consists in replacing entity type A by entity types A1 and A2 of identical structure in

---

13 Discovering this abnormal FD pertains to the DATA STRUCTURE EXTRACTION process.

such a way that the population of A is partitioned into those of A1 and A2. This technique yield smaller data sets, smaller indexes and allows for better space allocation and management (e.g. backup and distribution). Its inverse, i.e. **horizontal merging**, decreases the number of entity types and makes physical clustering easier. **Vertical partitioning** of entity type A partitions its attribute/role components into two (or more) entity types A1 and A2, linked by a one-to-one rel-type. This partitioning is driven by processing considerations : components that are used simultaneously are grouped into a specific entity type. This decreases the physical length of A entities, and improves access time. **Vertical merging** is the inverse technique (8.1). It merges two entity types that are linked by a one-to-one rel-type (possibly recovered from a foreign key) in order to avoid double access to get related entities. Horizontal partitioning can be applied to rel-types as well. For instance the population of *many-to-many* rel-type R can be split into *one-to-many* rel-type R1 that collects selected instances, while *many-to-many* rel-type R2 collects the others. The implementation of R1 will be more efficient than that of R2.
Another frequent restructuration consists in replacing some one-to-many rel-types by foreign keys, even in DBMS supporting such rel-types (IMS, CODASYL). The argument can be to physically split the database into independent fragments that are easier to distribute, to recover, to backup.

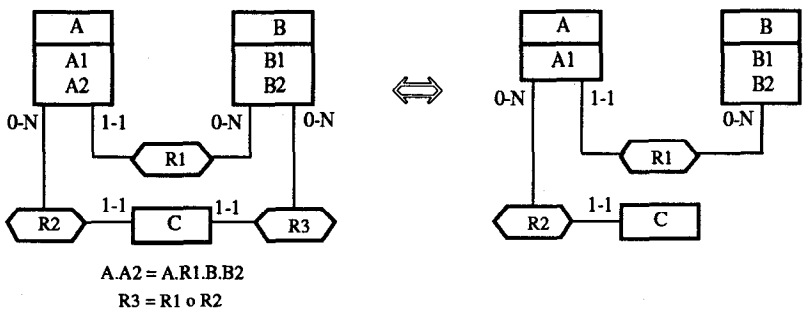

***Figure 6.1** - A2 has been recognized (typically in the DATA STRUCTURE EXTRACTION phase) as a duplicate attribute and R3 as the composition of R1 and R2. They are removed.*

7. DMS-UNTRANSLATION TRANSFORMATIONS

This is the process that is the most dependent on the DMS. Consequently, the transformations will be classified according to the most popular DMS, namely standard file managers (e.g. COBOL), Relational DBMS, CODASYL DBMS, TOTAL/IMAGE DBMS and IMS DBMS. Not all the possible, nor even all the useful transformations can be described in this paper (see [15] for instance). Therefore only some of the most needed techniques will be discussed. As will be shown, transformations used for recovering some DBMS schemas can be used for other models as well. Though these techniques (and some other specific ones) can be used to process object-oriented structures, this topic has been discarded in this paper.

The reader will probably be surprised that processing of some popular constructs have been ignored in this section. Such is the case for *many-to-many* rel-types, *N-ary* (N > 2) rel-types, *attribute* entity-types and *IS-A* hierarchies, that are generally considered in most proposals. In fact these problems have been discarded since they are common to all DMS and there is no general agreement on whether they must be recovered or not. Therefore they will be addressed in the CONCEPTUAL NORMALIZATION process.

Due to their SR-property, most transformations that will be presented can be read from left to right, in which case they provide reverse engineering techniques, and from right to left, in which case they represent their forward design counterpart.

7.1 COBOL file structures untranslation

The COBOL data model imposes few constraints on attribute structures. The most important one concerns multivalued attributes, which can be represented through *list attributes* only. However, some programmers adopt relational-oriented representations such as *concatenation* or *instantiation*; addressed in figures 7.3 and 7.4. In addition, optional attributes are not explicitly represented except as multivalued attributes (.. `occurs 1 depending` ...). This being said, attribute representation will be ignored.
The absence of explicit rel-type representation is a more challenging constraint that can be coped with through techniques such as the following. The most popular representation is through foreign keys[14], that can be

[14] Representing a many-to-many rel-type by a relationship record type is not explicitly mentioned. Indeed, this structure can be recovered in two steps, first replacing the two foreign keys by one-to-many rel-types, as proposed in 7.1 and 7.7, then

multivalued (7.1). The referential constraint is a precondition that can prove difficult to assert in the Data strcture extraction phase when the DMS ignores it. Detecting it will require a careful analysis of the procedural sections of the programs, of the screen definitions or of the file contents. A useful evidence is that many foreign keys are secondary indexes (`alternate keys`) as well. In is worth to notice that the transformation requires B1 being an identifier of B, be it primary or candidate. Most current proposals are limited to primary identifiers.

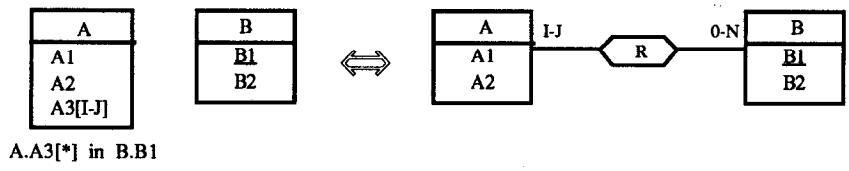

Figure 7.1 *- Representation of a foreign key (A3) by a rel-type.*

Another technique consists in implementing a one-to-many rel-type R between entity types A and B by integrating B entities as instances of a multivalued, compound, attribute of A. Recovering the origin constructs R and B can be done by applying transformation 3.5(a), where attribute A2 (possibly compound) of E is transformed into entity type EA2.

Finally, a one-to-many rel-type can be represented by a sorted multi-record-type, sequential or indexed file (7.2). The keys are structured in such a way that an A instance is followed by its associated B instances in the file sequence. The transformation is not reversible unless a referential constraint from B.B1.B11 to A.A1.A11 can be proved, for instance by file contents examination. However, this physical pattern is sufficiently frequent to make its rel-type origin strongly probable.

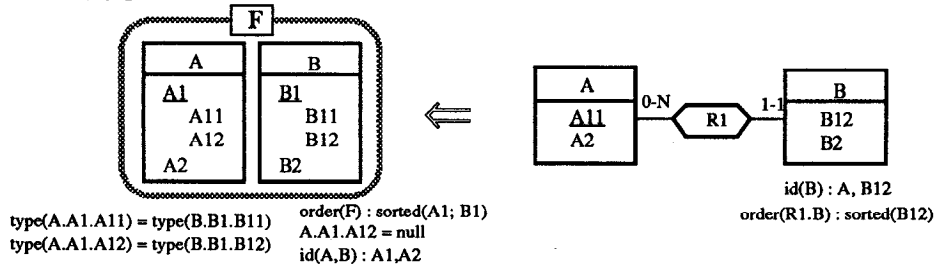

Figure 7.2 *- Interpretation of a multi-record-type file as a rel-type.*

7.2 RELATIONAL schema untranslation

As far as data structures are concerned, the relational model is particularly poor : single-valued and atomic attributes, no rel-types. Therefore, the main problem is to detect representations of multivalued attributes, compound attributes and rel-types.

A **multivalued attribute** can be represented by a distinct table including a foreign key referencing the main table. This pattern is processed by first resolving the foreign key (7.7), then by conceptual restructuring (3.5(a), reverse).

Two other representation techniques are also frequent, though less elegant, namely *instantiation* and *concatenation.*

The trace of an *instantiation transformation* can be detected by a structure of serial attributes, i.e. a sequence of attributes with the same type and length, and whose names present syntactical (EXP1, EXP2, etc) or semantical (JANUARY, FEBRUARY, etc) similarities. Figure 7.3 illustrates a simple transformation that is adequate for syntactical name similarity. When the uniqueness of their values can be proved, A2 should be declared a *multivalued attribute.*

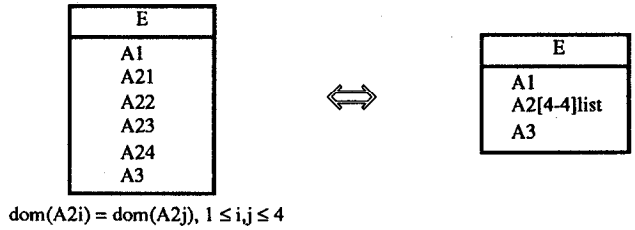

Figure 7.3 *- Representation of homogeneous serial attributes by a multivalued list attribute.*

---

applying a conceptual restructuration such as proposed in 3.1 if needed.

The *concatenation representation* of a multivalued attribute consists in replacing the set of values by their concatenation, expressed as a single-valued attribute. Its domain appears as possibly made of a repeated simple domain (7.4). Its detection generally requires the analysis of the procedural code, of the screens layouts or of the table contents.

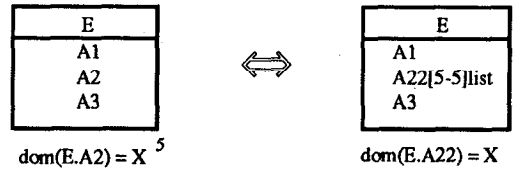

***Figure 7.4** - Representation of a concatenated attribute by a multivalued attribute.*

List attributes may be considered undesirable, and pertaining to the implementation level. Therefore transformation 7.5 can be applied when possible. Unfortunately, it is not reversible since a list structure does not enforce uniqueness and it adds an order relation to the set of values.

***Figure 7.5** - In some cases, a list attribute can be the representation of a multivalued attribute.*

A **compound attribute** can be represented by concatenation (in a way similar to 7.4), by attribute extraction (3.5(a) or 3.5(b)), or by ungrouping, to mention the most frequent techniques. Ungrouping can be detected by the presence of a sequence of heterogeneous attributes whose names suggest a semantic correlation, for instance through a common prefix. Recovering this gouping is straightforward (7.6).

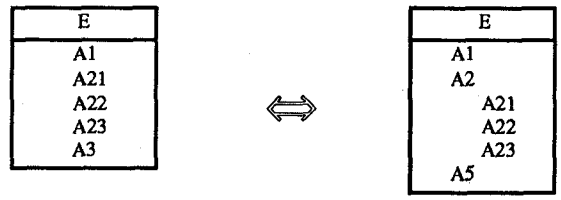

***Figure 7.6** - Representation of heterogeneous serial attributes by a compound attribute.*

Most **one-to-many rel-types** (and one-to-one as well) are represented by single-valued foreign keys (7.7). Some RDBMS provide an explicit representation of them through the `foreign key` clause. Their validation and management through `check` clauses and `trigger` mechanisms are less straightforward but still easy to detect, at least visually. Sometimes, there is no declarative hints, and the situation is similar to that of COBOL files. For instance, a common heuristics is that many foreign keys are supported by indexes, or are used to defined join-based views.

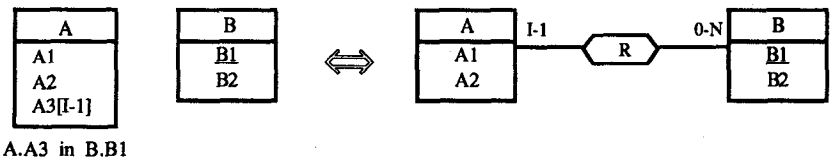

***Figure 7.7** - Representation of a single-valued foreign key by a one-to-many rel-type.*

7.3 CODASYL schema untranslation

Among the logical models considered in this paper, the CODASYL DBTG model is the closest to the ER model. As far as conceptual structures are concerned, the main restrictions apply on rel-types (one-to-many and non-recursive) and on identifiers (one absolute id through `location calc`; one relative id per rel-type through `duplicates not` in the `member` clause). Therefore, non-binary rel-types, many-to-many rel-types, one-to-one rel-types, recursive rel-types, secondary absolute identifiers, identifiers with more than one role, have been transformed. **Non-binary** and **many-to-many rel-types** will be considered as the target of conceptual normalization (3.1 and 3.4, reverse), and will be ignored in this section. A **one-to-one rel-type** is implemented

either by a one-to-many rel-type, or by a foreign key. Evidence of the first technique can be found through procedural code analysis and data analysis. Processing the second technique is similar to what has been suggested for COBOL and relational models. A **recursive rel-type** can be represented by an entity type and two one-to-many or one-to-one rel-types. Recovering such a rel-type falls in the conceptual restructuring techniques (3.1). It can also be represented by a foreign key, as for COBOL and relational models (7.1 and 7.7).

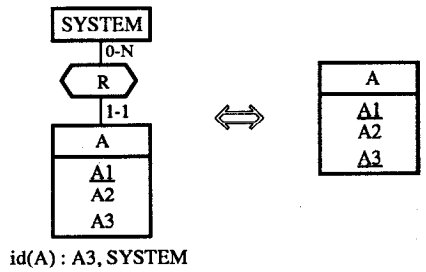

Figure 7.8 - Recovering a secondary all-attributes identifier.

An entity type with **K all-attributes identifiers** will be inserted into (K-1) SYSTEM rel-types, i.e. rel-types whose (0-N) role is played by the SYSTEM entity type. Each of the (K-1) secondary identifiers is declared local within one of each such SYSTEM rel-type (7.8). Another technique consists in extracting the attributes of the identifier to transform them into an entity type, linked to the main entity type through a one-to-one rel-type. This technique can be detected as an *attribute entity type* in the Conceptual Normalization process (3.5(a) and 3.5(b)).

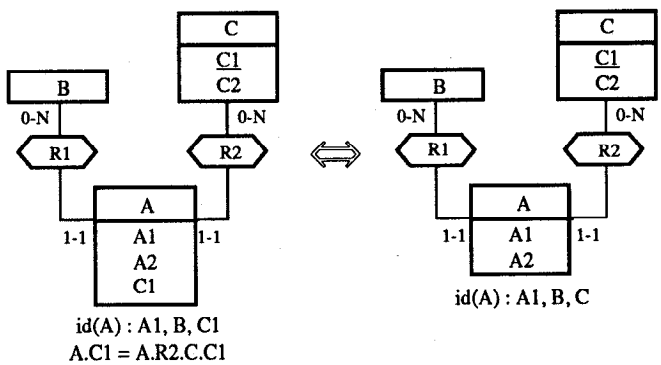

Figure 7.9 - Recovering a complex identifier (with more than one role component).

External and mixed identifiers that include **more than one role component** cannot be expressed as such. Either this identifier is discarded from the schema, and processed by procedural sections, or all the role components but one are replaced by foreign keys. The latter technique can be reversed as proposed in 7.9. Elicitating the referential constraint requires procedural text or database contents analysis. The schema may keep the source rel-type, according to the principles of *non-information bearing sets* as proposed in the 70's [21]. In such situations, some DBMS offer a trick (an option of the *set selection* clause) through which the referential constraint is automatically maintained.

IDS-1 (Honeywell-Bull), that uses a popular variant of the CODASYL model (in fact it is the pre-CODASYL model), requires similar rules, with additional constraints. The same can be said of MDBS-3 and 4 (MDBS) and SIBAS (Norsk-Data). ADABAS (Software AG) structures are often assimilated to CODASYL model [32]. However, this similarity is rather loose, and this model requires specific translation rules that are out of the scope of this paper.

7.4 TOTAL/IMAGE schema untranslation

The TOTAL DBMS (CINCOM), and its clone IMAGE (HP), propose a logical model that is generally classified as network [32]. However, it seems to fit best as a variant of the hierarchical model as far as design techniques are concerned. This model offers two kinds of entity types, namely the **parent** entity types (*master data set*), and the **child** entity types (*variable entry data set*). In addition one-to-many rel-types can be defined between entity types; each rel-type defines access-paths from the parent entities to chidren entities. A parent has single-valued, mandatory and atomic attributes, one of them being its identifier and access key; it can be origin (*one* side) of a rel-type. A child is the target (*many* side) of at least one rel-type. A child entity is the target of at least one rel-type instance (the

others can be optional). Among its attributes, there is a copy of the identifier value of each of its parent. These copies behave like redundant *foreign keys* that allow accessing the parent entities. A child has no identifier. A TOTAL/IMAGE schema is a two-level hierarchy - sometimes called a *shallow* structure - in which level 1 comprises parents only while level 2 is made of children only. In this model, the problems that occur when translating ER to TOTAL/IMAGE are numerous : expressing complex attributes, non-functional rel-types, one-to-one rel-types, recursive rel-types, parents that are target of some rel-types, children that are source of some rel-types (and in particular hierarchies with more than two levels and non-hierarchical schemas), entity types with more than one identifier or with secondary access keys (IMAGE has a feature for the latter), just to mention the most important. Systematic translation of some of these constructs has been proposed in [9].
In reverse engineering TOTAL/IMAGE schemas, the redundant foreign keys are detected without problem since they are explicitly declared; they can be discarded without loss. **Compound attributes** are most often processed the same way as in relational schemas. A **multivalued attribute** can be detected either the same way as in relational schemas or as a single-attribute, single-parent, child entity type (7.10).

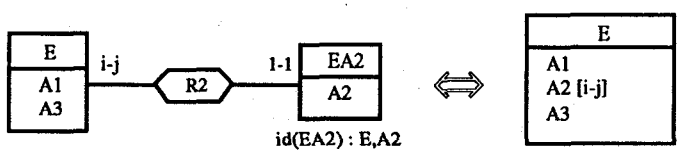

***Figure 7.10** - Recovering a multivalued attribute. If the id(EA2) precondition can be dropped, then A2 is a bag attribute.*

One-to-one rel-types can be processed as in CODASYL schemas. Non-compliant **one-to-many rel-types**, for instance recursive rel-types or rel-types between two parent entity types, are most often expressed as relationship entity types (7.11). They can be recovered provided R1 can be proved one-to-one, by analyzing either the procedural code of the programs or the database contents. Note that some one-to-many rel-types may be expressed implicitely as foreign keys, just like in relational schemas, and are therefore more difficult to detect. As before, recovering many-to-many or higher-degree rel-types is considered as of conceptual normalization concern.

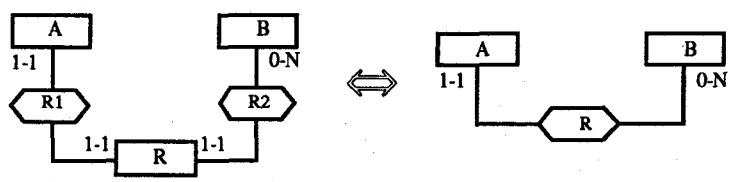

***Figure 7.11** - Recovering a one-to-many rel-type.*

The *2-level hierarchy* constraint implies eliminating (1) **non-hierarchical constructs**, such as circuits, and (2) **deep hierarchies**. Technique 7.11 (reverse) is often used to move A one level up w.r.t. B (R is a common child of A and B, which are therefore at the same level). Conversely, interpreting child entity type R as a one-to-many rel-type between A and B will automatically recover the origin B-A hierarchy. However, other techniques can be used, such as that which is described in 7.12, based on the identifier substitution transformation (3.7). It requires detecting the inclusion constraint that states that any (`A`, `B`) instance obtained from a `D` entity must identify a `C` entity (the notation `S1 o S2` expresses the relational composition of `S1` and `S2`). One-to-many rel-type elimination through foreign key and entity type merging techniques are also often observed.

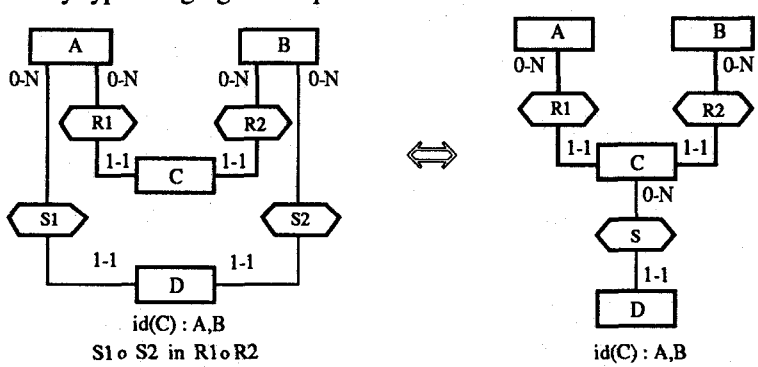

***Figure 7.12** - Recovering a 3-level hierarchy from a 2-level hierarchy.*

7.5 IMS schema untranslation

The IMS (DL/1) model proposes to structure a schema as a collection of entity types (*segment types*) linked by *one-to-many* rel-types, that fall into two classes, physical and logical [6]. The *one* side of a rel-type is a physical or logical parent while the *many* side is a physical or logical child.Ignoring logical rel-types, the schema reduces to a forest, i.e. a collection of arborescences (or *physical DB's*). The root of each arborescence is called a root entity type (*root segment*). Each root can have one identifier, that is an access-key and can be a sort key as well. It consists of one attribute. Each rel-type defines an access path, from the parent to the child only. A child entity type can have an identifier made of its parent + one local attribute. This identifier is not an access key. Attributes are mandatory, single-valued and atomic. However, compound attributes can be simulated by defining overlapping attributes through common physical positions.

Most IMS schemas are built with the latter constructs[15]. However, two additional features can be used, namely logical rel-types and secondary indexes. A logical rel-type represents an access-path from a (logical) child entity type to a parent one. A logical rel-type can be defined between any two entity types provided some constraints are satisfied : an entity type can have only one logical parent, a logical child must be a physical child, a logical child cannot be a logical parent, the physical parent of a logical child cannot be a logical child, etc. When bi-directional access paths are needed, IMS proposes to define two, inverse, logical rel-type structures (the *pairing* technique). A secondary index is an access key based on any attribute of the database, whatever its entity type. Surprisingly enough, logical rel-types and secondary indexes are considered in the IMS world as intimidating constructs which are difficult and dangerous to use. Even recent references [8], though insisting on their harmlessness, suggest to avoid them whenever possible, for instance by replacing rel-types by foreign keys controlled manually. These constraints define clearly the problems that will appear when translating an ER schema into an IMS structure : compound and multivalued attributes, entity types with several identifiers, one-to-one, many-to-many or recursive rel-types, circuits, entity types with more than two parents, complex identifiers, etc. Additional problems have to be solved when no logical rel-types are used.

Most of these problems have already been discussed and solved in the TOTAL/IMAGE section, or can be considered as relevant to conceptual normalization (e.g. many-to-many rel-types). We shall concentrate on recovering non-compliant **one-to-many rel-types**. Such a rel-type can be transformed, for instance, into a foreign key (manually controlled) as in relational schemas, by merging its entity types (producing a possibly unnormalized structure), or into a *relationship* entity type. Recovering the source rel-type from application of latter technique is described in 7.11. However, to make the process clearer, processing a typical IMS substructure is depicted in 7.13, where entity types F and G have been considered as one-to-many rel-type representations. Once again, the main difficulty is to detect the *one-to-one cardinality* of R5 and R7. R6 and R8 (or R5 and R7) can be replaced by foreign keys, possibly supported by secondary indexes.

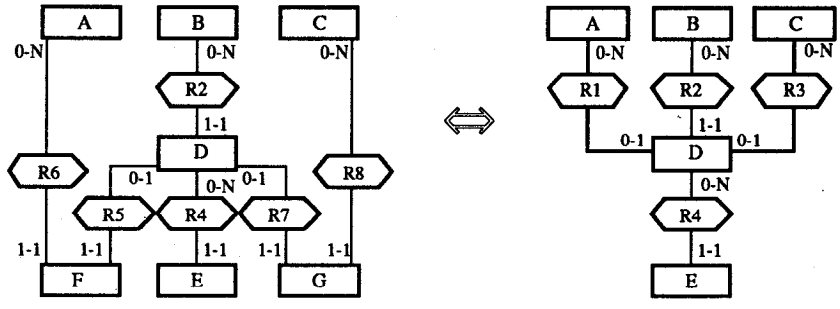

Figure 7.13 - Recovering one-to-many rel-types.

Note that duplicate structures resulting from *pairing* can be resolved easily since they are explicitly declared in the DL/1 schema. The redundant logical children can be merged in a preliminary step.

8. CONCEPTUAL NORMALIZATION TRANSFORMATIONS

These transformations are aimed to make higher-level semantic constructs explicit. Whether such expressions are desirable is a matter of methodological standard and of personal taste. For instance, a method that is based on a binary, functional ER model (e.g. the Bachman's model) will accept the conceptual schema obtained so far. More powerful models will require the expression of e.g. IS-A relations when relevant. In addition, the final conceptual schema is supposed to be as readable and concise as possible, though these properties basically are subjective. We

15 As well as SYSTEM-2000 databases (MRI / SAS) and FOCUS files (Information Builders)

shall mention some standard transformations that are of interest when refining a conceptual schema. This list is of course far from complete.

A **relationship entity type**, i.e. an entity type whose aim obviously is to relate two or more entity types, will be transformed into a rel-type (3.1 reverse and 3.4 reverse). This technique typically produces many-to-many and N-ary rel-types, and rel-types with attributes.

An **attribute entity type** has one attribute only, and is linked to one other entity type A only. It can be interpreted as nothing more than an attribute of A, possibly multivalued (3.5(a) reverse and 3.5(b) reverse).

A **one-to-one rel-type** may express the connection between fragments B1 and B2 of a unique entity type B (vertical partitioning). These fragments can be merged into this entity type (8.1).

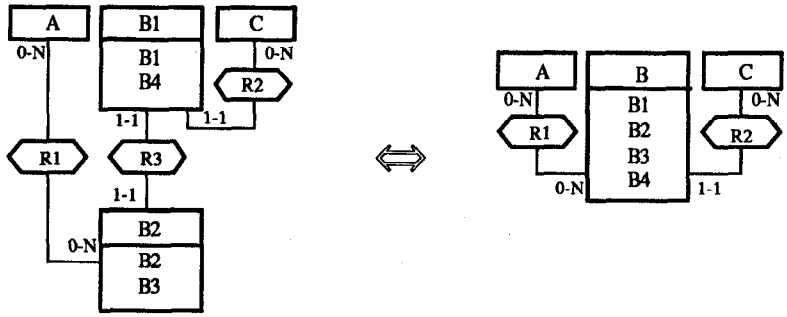

***Figure 8.1** - Merging ET. If the cardinality of R3.B1 is [0-1], constraint "*`coexist(B):B2,B3,R1.B`*" must be added to B.*

Conversely, an entity type that appears as comprising **too many attributes** and roles can suggest a decomposition into fragments linked by *one-to-one* rel-types (Figure 8.1 reverse).

A **N-ary rel-type that has a role with cardinality 1-1** can be decomposed into binary, one-to-many rel-types through a P/J transformation (8.2).

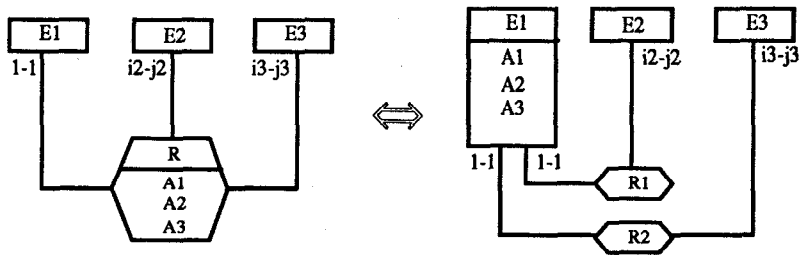

***Figure 8.2** - Decomposition of a rel-type through a [1-1] role. Processing of a [0-1] role would induce the coexistence constraint "*`coexist(E1):A1,A2,A3,R1.E1,R2.E1`*".*

Entity types that seem to have some **attributes and roles in common** can be made the subtypes of a common supertype that inherits the common characteristics (8.3).

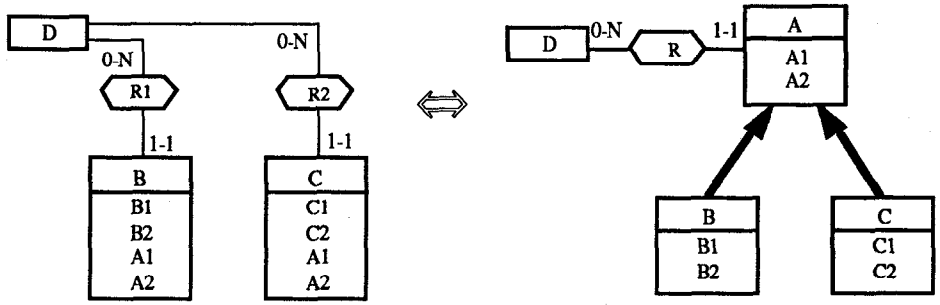

***Figure 8.3** - Defining a super-type. In fact, the transformation is a bit more complex since it requires conditions on the semantic equivalence of (R1,R2), (B.A1,C.A1) and (B.A2,C.A2)*

An entity type that has **subsets of optional, coexistent, attributes and roles** can be given subtypes, each of them inheriting one of these subsets (8.4). An entity type that has one subset of optional attributes and roles can also be examined for such a transformation.

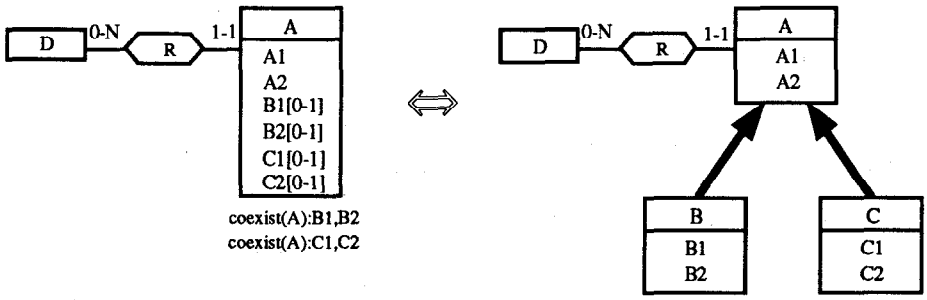

Figure 8.4 - Defining subtypes from optional, coexistent, subsets of attributes/roles.

One or several **one-to-one rel-types**[16] that concern a common entity type A may also express a specialization relation in which A is the supertype. These rel-types are replaced by IS-A relations (8.5).

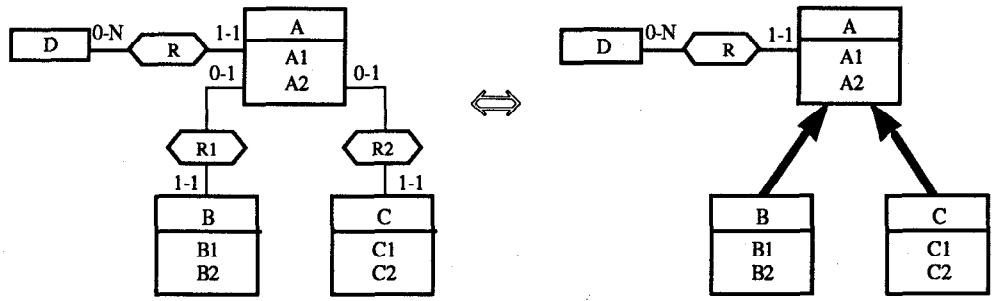

Figure 8.5 - Defining IS-A relations.

Note that the last three techniques derive from the three basic expressions of an IS-A hierarchy into the plain ER model (see [1] for instance). They can be specialized in order to make them cover the situations of total and/or exclusive subtypes.

9. CONCLUSIONS

The principles that have been presented in this paper form the baselines on which flexible, DMS-independent DBRE methodologies can be built. Indeed, it appears from the analysis developed so far that, (1) all database schemas can be expressed within the unique schema representation model, whetever their underlying data model and their abstraction level; (2) recovering the schemas of a database requires performing the same processes, whatever the DMS; (3) all the DMS-specific processes can be based on the same toolset of transformation techniques.

Considering the limited scope of this paper, only some theoretical aspects of DBRE have be presented. Practicing DBRE on actual applications shows that this formal framework provides invaluable guidelines in an otherwise highly unstructured process. On the other hand, practical experiments also shows that this framework must be made more flexible, and that support for less formal processes have to be provided as well. Let us mention some examples.

- In some cases, the precondition of a transformation cannot be completely recovered. For instance, proving the inclusion constraint of a foreign key can be complex, if not impossible, in some standard file manager or older RDBMS. Therefore, the *probable* R-transformation will be used instead of the deterministic SR-transformation. This results in replacing deterministic reasonings by heuristics, therefore introducing an *uncertainty* factor.
- Ideally, an construct found in a schema has been introduced in order to satisfy functional or most probably technical requirements. Today, ignoring these requirements and the rationales underlying this introduction, makes it difficult to interprete this construct, and to discover the semantics (or absence thereof) it may represent. In particular, an observed data construct can often be the target of more than one source construct through different transformations. Determining which one must be considered is not always that easy.
- In complex applications, the whole process cannot be as linear as suggested in section 5. It appears instead as an iterative process strongly based on learning mechanisms. Having recovered some conceptual structures may suggest going back to the source texts to search them for additional information. A consequence is that a database

16 It is worth noting that a one-to-one rel-type can have three interpretations : true one-to-one rel-type, IS-A relation representation and vertical splitting.

schema in a DBRE process generally is not as homogeneous as in forward engineering. Indeed, some parts of a schema can be already conceptualized, while others still contain technical constructs to be conceptualized.

- Data structure extraction is more complex that generally presented in the literature, and requires processes and techniques that are completely ignored in current CASE technology. Let us mention three specific problems only.
  - *structure hiding* is a common programming practice that consists in implementing parts of the logical data structures in local variables structures of the application programs instead with DMS constructs. Compound field, or list of contiguous fields, represented as a large anonymous field in the DDL schema (*filler* in COBOL) is a popular example that can make the recovering process very difficult, since it requires a semantic analysis of the procedural parts of the programs [14].
  - parts of the schema, mainly untranslatable integrity constraints, have not been translated into the DDL schema. Here too, a semantic analysis of the procedural parts of the programs is needed [14].
  - in standard data structures, such as COBOL files, each application program does not include the description of the files, but a view of them. The complete description of the files is generally unavailable, and must be rebuilt from the collection of such views through schema integration techniques that differ significantly from traditional techniques [18].

Anyway, the transformational approach is considered as a favorable framework for software production automation. Indeed, the transformation techniques can be completely formalized, and therefore translated into restructuration algorithms that can be the kernel of a generic CASE tool [13]. For instance, two CASE tools have been built on transformational toolboxes that implement the techniques presented in this paper, namely TRAMIS/Master [HAINAUT,92c] for database forward engineering and PHENIX [18] for database reverse engineering. In addition, expressing schema transformations in a predicative way allows logic-based processing of important engineering activities. For instance, it is fairly straighforward to design heuristics for both forward and reverse engineering that make use of the precondition part of the transformations. This approach has been sketched in [13].

10. REFERENCES

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

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

[3] 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

[4] 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

[5] 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

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

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

[8] Geller, J., R., *IMS, Administration, Programming and Data Base Design*, Wiley, 1989

[9] Hainaut, J-L., *Theoretical and practical tools for data base design*, in Proc. of the Very Large Databases Conf., pp. 216-224, September, 1981

[10] Hainaut, J.-L., *A Generic Entity-Relationship Model*, in Proc. of the IFIP WG 8.1 Conf. on *Information System Concepts: an in-depth analysis*, North-Holland, 1989

[11] Hainaut, J-L., *Entity-generating Schema Transformation for Entity-Relationship Models*, in Proc. of the 10th Entity-Relationship Approach, San Mateo (CA), 1991

[12] Hainaut, J-L, *Database Reverse Engineering, Models, Techniques and Strategies*, in Preproc. of the 10th Conf. on Entity-Relationship Approach, San Mateo (CA), 1991

[13] 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

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

[15] Hainaut, J-L, *Schema Transformation for Database Engineering - Theoretical Elements*, Research report, Institut d'Informatique, FUNDP, Namur, May, 1993

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

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

[18] 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 Soft. Eng. and Applications, Toulouse, 7-11 Dec., 1992

[19] 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

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

[21] Metaxides, A., *"Information bearing" and "non-information bearing" sets*, in Data Base Description, IFIP TC2 Work. Conf., Douqué & Nijssen (Ed.), pp363-368, North-Holland, 1975

[22] Navathe, S., B., *Schema Analysis for Database Restructuring*, in ACM TODS, Vol.5, No.2, June 1980

[23] 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

[24] Nijssen, G., M., Halpin, T., A., *Conceptual Schema and Relational Database Design*, Prentice-Hall, 1989

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

[26] Premerlani, W., J., Blaha, M.R., *An Approach for Reverse Engineering of Relational Databases*, in Proc. of the IEEE Working Conf. on Reverse Engineering, Baltimore, May 1993

[27] 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

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

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

[30] 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

[31] Teorey, T. J., *Database Modeling and Design*, Morgan Kaufmann, 1990

[32] Tsichritzis, D., C., Lochovsky, F., H., Data Base Management Systems, Academic Press, 1977

[33] Ullman, J., D., *Principles of Data- and Knowledge-base Systems*, Computer Science Press, 1989

[34] 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, pp. 345-360, Oct., 1990