Abstract
In this chapter, we develop a transformational framework in which many database engineering processes can be modelled in a precise way, and in which properties such as semantics preservation and propagation can be studied rigorously. Indeed, the transformational paradigm is particularly suited to database schema manipulation and translation, that are the basis of such processes as schema normalization and optimization, model translation, reverse engineering, database integration and federation or database migration. The presentation first develops a theoretical framework based on a rich, wide spectrum specification model. Then, it describes how more complex transformations can be built through predicate-based filtering and composition. Finally, it analyzes two major engineering activities, namely database design and reverse engineering, modelled as goal-oriented schema transformations.

1. MOTIVATION AND INTRODUCTION

Modelling software design as the systematic transformation of formal specifications into efficient programs, and building CASE tools that support it, has long been considered one of the ultimate goals of software engineering. For instance, Balzer (1981) and Fikas (1985) consider that the process of developing a program [can be] formalized as a set of correctness-
preserving transformations [...] aimed to compilable and efficient program production. In this context, according to Partsch (1983),

*a transformation is a relation between two program schemes P and P’ (a program scheme is the [parameterized] representation of a class of related programs; a program of this class is obtained by instantiating the scheme parameters). It is said to be correct if a certain semantic relation holds between P and P’.*

These definitions still hold for database schemas, which are special kinds of abstract program schemes. The concept of transformation is particularly attractive in this realm, though it has not often been made explicit (for instance as a user tool) in current CASE tools. A (schema) transformation is most generally considered to be an operator by which a data structure S1 (possibly empty) is replaced by another structure S2 (possibly empty) which may have some sort of equivalence with S1. Some transformations change the information contents of the source schema, particularly in schema building (adding an entity type or an attribute) and in schema evolution (removing a constraint or extending a relationship type). Others preserve it and will be called semantics-preserving or reversible. Among them, we will find those which just change the nature of a schema object, such as transforming an entity type into an relationship type or extracting a set of attributes as an independent entity type.

Transformations that are proved to preserve the correctness of the original specifications have been proposed in practically all the activities related to schema engineering: schema normalization (Rauh, 1995), DBMS\(^3\) schema translation (Hainaut, 1993b; Rosenthal, 1988), schema integration (Batini, 1992; McBrien, 2003), schema equivalence (D’Atri, 1984; Jajodia, 1983; Kobayashi, 1986; Lien, 1982), data conversion (Navathe, 1980; Estévenart, 2003), reverse engineering (Bolois, 1994; Casanova, 1984) (Hainaut, 1993; Hainaut, 1993b), schema optimization (Hainaut, 1993b; Halpin, 1995) database interoperability (McBrien, 2003; Thiiran, 2001) and others. The reader will find in (Hainaut, 1995) an illustration of numerous application domains of schema transformations.

The goal of this chapter is to develop and illustrate a general framework for database transformations in which all the processes mentioned above can be formalized and analyzed in a uniform way. In Section 2, we present a wide spectrum formalism in which all the information/data models currently used can be specified, and on which a set of basic transformational operators is defined. We also study the important property of semantics-preservation of these operators. Section 3 explains how higher-level transformations can be built through three mechanisms, from mere composition to complex model-driven transformation. The database design process is revisited in Section 4, and given a transformational interpretation. The same exercise is carried out in Section 5 for database reverse engineering. Section 6 concludes the chapter.

---

3. Database Management System
2. SCHEMA TRANSFORMATION BASICS

This section describes a general transformational theory that will be used as the basis for modelling database engineering processes. First, we discuss some preliminary issues concerning the way such theories can be developed. Then, we define a wide-spectrum model from which operational models (i.e., those which are of interest for practitioners) can be derived. The next sections are dedicated to the concept of transformation, to its semantics-preservation property, and to the means to prove it. Finally, some important basic transformations are described.

**Warning.** In the database world, a general formalism in which database specifications can be built is called a model. The specification of a database expressed in such a model is called a schema.

2.1 Developing transformational theories

Developing a general purpose transformational theory requires deciding on the specification formalism, i.e., the model, in which the schemas are expressed and on the set of transformational operators. A schema can be defined as a set of constructs (entity types, attributes, keys, indexes, etc.) borrowed from a definite model whose role is to state which constructs can be used, according to which assembly rules, in order to build valid schemas. For simplicity, the concept of entity type is called a construct of the ERA model, while entity type CUSTOMER is a construct of a specific schema. They are given the same name, though the latter is an instance of the former.

Though some dedicated theories rely on couples of models, such as those which are intended to produce relational schemas from ERA schemas, the most interesting theories are based on a single formalism. Such a formalism defines the reference model on which the operators are built. According to its generality and its abstraction level, this model defines the scope of the theory, that can address a more or less wide spectrum of processes. For instance, building a theory on the relational model will allow us to describe, and to reason on, the transformation of relational schemas into other relational schemas. The 1NF normalization theory is a popular example. Another example would be a transformational theory based on the ORM (Object-Role model) that would provide techniques for transforming (normalizing, optimizing) conceptual schemas into other schemas of the same abstraction level (de Troyer 1993; Proper 1998). The hard challenge is to choose a unique model that can address not only intra-model transformations, but inter-model operators, such as ORM-to-relational conversion.

To identify such models, let us consider a set of models $\Gamma$ that includes, among others, all the operational formalisms that are of interest for a community of practitioners, whatever the un-

---

4. Entity-relationship-attribute model. The UML class model is a variant of the ERA model.
5. 1NF, or First Normal Form designates the class of relations defined on simple domains (which are neither relations nor powersets). By contrast, a non 1NF relation is said to be in N1NF, or NF2 for short.
derlying paradigm, the age and the abstraction level of these formalisms. For instance, in a large company whose information system relies on many databases (be they based on legacy or modern technologies) that have been designed and maintained by several teams, this set is likely to include several variants of the ERA model, UML class diagrams, several relational models (e.g., Oracle 5 to 10 and DB2 UDB), the object-relational model, the IDMS and IMS models and of course the standard file structure model on which many legacy applications have been developed.

Let us also consider the transitive inclusion relation "≤" such that $M \leq M'$, where $M \neq M'$ and $M, M' \in \Gamma$, means that all the constructs of $M$ also appear in $M'$. For instance, if $M$ denotes the standard relational model and $M'$ the object-relational model, then $M \leq M'$ holds, since each schema expressed in $M$ is a valid schema according to model $M'$.

Now, we consider a model $M^*$ in $\Gamma$, such that,

$$\forall M \in \Gamma, M \neq M^*: M \leq M^*,$$

and a model $M_0$ in $\Gamma$, for which the following property holds:

$$\forall M \in \Gamma, M \neq M_0: M_0 \leq M.$$  

$(\Gamma \times \Gamma, \leq)$ forms a lattice of models, in which $M_0$ denotes the bottom node and $M^*$ the upper node.

$M_0$, admittedly non-empty, is made up of a very small set of elementary abstract constructs, typically nodes, edges and labels. An ERA schema $S$ comprising an entity type $E$ with two attributes $A_1$ and $A_2$ would be represented in $M_0$ by the nodes $n_1, n_2, n_3$ which are given the labels "$E$", "$A_1$" and "$A_2$", and by the edges $(n_1,n_2)$ and $(n_1,n_3)$.

On the contrary, $M^*$ will include a greater variety of constructs, each of them being a natural abstraction of one or several constructs of lower-level models. This model should include, among others, the concepts of object type, attribute and inter-object association, so that the contents of schema $S$ will be represented in $M^*$ by an object type with name "$E$" comprising two attributes with names "$A_1$" and "$A_2$".

Due to their high level of abstraction, models $M_0$ and $M^*$ are good candidates to develop a transformational theory relying on a single model. Considering the context-dependent definition of $\Gamma$, $M_0$ and $M^*$, we cannot assert that these concepts are unique. Therefore, there is no guarantee that a universal theory can be built.

Approaches based on $M_0$ generally define data structures as semantics-free binary graphs on which a small set of rewriting operators are defined. The representation of an operational model $M$ such as ERA, relational or XML, in $M_0$ requires some additional features such as typed

---

6. Defining more formally what the assertion this construct of $M$ also belongs to $M'$ exactly means would require a development which would be useless in this paper. Therefore, we will rely on an intuitive meaning of this relation only. For example, the concepts of field and of column will be considered the same though some slight differences exist between them. The same can be said for entity type (ERA), object class (UML), segment type (IMS), record type (standard files, CODA-SYL) and table (SQL2).
nodes (object, attribute, association and roles for instance) and edges, as well as ad hoc assembly rules that define patterns. A transformation specific to M is also defined by a pattern, a sort of macro-transformation, defined by a chain of M0 transformations. (McBrien, 1998) is a typical example of such theories. We can call this approach constructive or bottom-up, since we build operational models and transformations by assembling elementary building blocks.

The approaches based on M* naturally require a larger set of rewriting rules. An operational model M is defined by specializing M*, that is, by selecting a subset of concepts and by defining restrictive assembly rules. For instance, a relational schema can be defined as a set of object types (tables), a set of attributes (column), each associated with an object type (at least one attribute per object type) and a set of uniqueness (keys) and inclusion (foreign keys) constraints. This model does not include the concept of association. The transformations of M are those of M* which remain meaningful. This approach can be qualified by specialization or top-down, since an operational model and its transformational operators are defined by specializing (i.e., selecting, renaming, restricting) M* constructs and operators. DB-MAIN (Hainaut, 1996b) is an example of this approach. In the next section, we describe the main aspects of its model, named GER.7

2.2 A data structure specification model

Database engineering is concerned with building, converting and transforming database schemas at different levels of abstraction, and according to various paradigms. Some processes, such as normalization, integration and optimization operate in a single model, and will require intra-model transformations. Other processes, such as logical design, use two models, namely the source and target models. Finally, some processes, among others, reverse engineering and federated database development, can operate on an arbitrary number of models (or on a hybrid model made up of the union of these models) as we will see later on. The GER model is a wide-spectrum formalism that has been designed to,

- express conceptual, logical and physical schemas, as well as their manipulation,
- support all the data-centered engineering processes,
- support all DMS8 models and the production and manipulation of their schemas.

The GER is an extended entity-relationship model that includes, among others, the concepts of schema, entity type, entity collection, domain, attribute, relationship type, keys, as well as various constraints. In this model, a schema is a description of data structures. It is made up of specification constructs which can be, for convenience, classified into the usual three abstraction levels, namely conceptual, logical and physical. We will enumerate some of the main constructs that can appear at each level:

7. For Generic Entity-Relationship model.
8. For Data Management System, a term that encompasses file managers and DBMSs.
• A conceptual schema comprises entity types (with/without attributes; with/without identifiers), super/subtype hierarchies (single/multiple, total and disjoint properties), relationship types (binary/N-ary; cyclic/acyclic; with/without attributes; with/without identifiers), roles of relationship type (with min-max cardinalities; with/without explicit name; single/multi-entity-type), attributes (of entity or relationship types; multi/single-valued; atomic/compound; with cardinality), identifiers (of entity type, relationship type, multivalued attribute; comprising attributes and/or roles), constraints (inclusion, exclusion, coexistence, at-least-one, etc.)

• A logical schema comprises record types, fields, arrays, foreign keys, redundancy, etc.

• A physical schema comprises files, record types, fields, access keys (a generic term for index, calc key, etc), physical data types, bag and list multivalued attributes, and other implementation details.

It is important to note that these levels are not part of the model. The schema of Fig. 1 illustrates some major concepts borrowed to these three levels. Such a hybrid schema could appear in reverse engineering. One remarkable characteristic of wide spectrum models is that all the transformations, inclu-
ding inter-model ones, appear as intra-model operators. This has highly interesting conse-
quences. First, a transformation $\Sigma$ designed for manipulating schemas in an operational model $M_1$ can be used in a model $M_2$ as well, provided $M_2$ includes the constructs on which $\Sigma$ opere-
ates. For instance, most transformations dedicated to COBOL data structure reverse enginee-
ring appear to be valid for relational schemas as well. This strongly reduces the number of
operators. Secondly, any new model can profit from the techniques and reasoning that have been developed for current models. For instance, designing methods for translating conceptual schemas into object-relational structures or into XML schemas (Estiévenart, 2003), or re-
verse engineering OO-databases (Hainaut, 1997) have proved particularly easy since these new methods can be, to a large extent, derived from standard ones.

The GER model has been given a formal semantics in terms of an extended NF2 model (Hainaut, 1989; Hainaut, 1996). This semantics will allow us to analyze the properties of transformations, and particularly to precisely describe how, and under which conditions, they propagate and preserve the information contents of schemas.

Let us note that we have discarded the UML class model as a candidate for $M^*$ due to its in-
trinsic weaknesses, including its lack of agreed upon semantics, its non-regularity and the ab-
sence of essential concepts. On the contrary, a carefully defined subset of the UML model could be be a realistic basis for constructive approaches.

### 2.3 Specifying operational models with the GER

In this section, we illustrate the specialization mechanism by describing a popular operational formalisms, namely the standard 1NF relational model. All the other models, be they concep-
tual, logical or physical can be specified similarly.

<table>
<thead>
<tr>
<th>GER constructs</th>
<th>relational constructs</th>
<th>assembly rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>database schema</td>
<td></td>
</tr>
<tr>
<td>entity type</td>
<td>table</td>
<td>an entity type includes at least one attribute</td>
</tr>
<tr>
<td>simple domain</td>
<td>domain</td>
<td></td>
</tr>
<tr>
<td>single-valued and atomic</td>
<td>nullable column</td>
<td></td>
</tr>
<tr>
<td>attribute with cardinality [0-1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>single-valued and atomic</td>
<td>not null column</td>
<td></td>
</tr>
<tr>
<td>attribute with cardinality [1-1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>primary identifier</td>
<td>primary key</td>
<td>a primary identifier comprises attributes with cardinality [1-1]</td>
</tr>
<tr>
<td>secondary identifier</td>
<td>unique constraint</td>
<td></td>
</tr>
<tr>
<td>reference group</td>
<td>foreign key</td>
<td>the composition of the reference group must be the same as that of the target identifier</td>
</tr>
<tr>
<td>GER names</td>
<td>SQL names</td>
<td>the GER names must follow the SQL syntax</td>
</tr>
</tbody>
</table>

Figure 2 - Defining the standard relational model as a subset of the GER model.
A relational schema mainly includes tables, domains, columns, primary keys, unique constraints, not null constraints and foreign keys. The relational model can therefore be defined as in Fig. 2. A GER schema made up of constructs from the first columns only, that satisfy the assembly rules, can be called relational. As a consequence, a relational schema cannot comprise is-a relations, relationship types, multivalued attributes or compound attributes. The physical aspects of the relational data structures can be addressed as well. Fig. 3 gives additional specifications through which physical schemas for a specific RDBMS can be specified. These rules generally include limitations such as no more than 64 columns per index, or the total length of the components of any index cannot exceed 255 characters.

<table>
<thead>
<tr>
<th>GER constructs</th>
<th>relational constructs</th>
<th>assembly rules for a specific DBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>access key</td>
<td>index</td>
<td>comprises from 1 to 64 attributes of the parent entity type</td>
</tr>
<tr>
<td>collection</td>
<td>table space</td>
<td>a collection includes 1 to 255 entity types; an entity type belongs to at most 1 collection</td>
</tr>
</tbody>
</table>

Figure 3 - Defining the main technical constructs of relational data structures as they are implemented in a specific RDBMS.

2.4 Transformation: definition

The definitions that will be stated here below are model-independent. In particular, they are valid for the GER model, so that the examples will be given in the latter. Let us denote by M the model in which the source and target schemas are expressed, by S the schema on which the transformation is to be applied and by S' the schema resulting from this application. Let us also consider sch(M), a function that returns the set of all the valid schemas that can be expressed in model M, and inst(S), a function that returns the set of all the instances that comply with schema S.

A transformation \( \Sigma \) consists of two mappings \( T \) and \( t \) (Fig. 4):

- \( T \) is the structural mapping from \( \text{sch}(M) \) onto itself, that replaces source construct \( C \) in schema \( S \) with 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 the minimal precondition \( P \) that any construct \( C \) must satisfy in order to be transformed by \( T \), and the maximal postcondition \( Q \) that \( T(C) \) satisfies. \( T \) specifies the rewriting rule of \( \Sigma \).
- \( t \) is the instance mapping from \( \text{inst}(S) \) onto \( \text{inst}(S') \), that states how to produce the \( T(C) \) instance that corresponds to any instance of \( C \). If \( c \) is an instance of \( C \), then \( c' = t(c) \) is the corresponding instance of \( T(C) \). \( t \) can be specified through any algebraic, logical or procedural expression.

According to the context, \( \Sigma \) will be noted either \( <T,t> \) or \( <P,Q,t> \).
Each transformation $\Sigma$ is associated with an inverse transformation $\Sigma'$ which can undo the result of the former under certain conditions that will be detailed in the next section.

### 2.5 Reversibility of a transformation

The extent to which a transformation preserves the information contents of a schema is an essential issue. Some transformations appear to augment the semantics of the source schema (e.g., adding an attribute), some remove semantics (e.g., removing an entity type), while others leave the semantics unchanged (e.g., replacing a relationship type with an equivalent entity type). The latter are called reversible or semantics-preserving. 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.

- A transformation $\Sigma_1 = <T_1,t_1> = <P_1,Q_1,t_1>$ is reversible, iff there exists a transformation $\Sigma_2 = <T_2,t_2> = <P_2,Q_2,t_2>$ such that, for any construct $C$, and any instance $c$ of $C$: $P_1(C) \Rightarrow (T_2(T_1(C))=C)$ and $t_2(t_1(c)=c)$. $\Sigma_2$ is the inverse of $\Sigma_1$, but the converse is not true.$^9$. For instance, an arbitrary instance $c'$ of $T(C)$ may not satisfy the property $c'=t_1(t_2(c'))$.
- If $\Sigma_2$ is reversible as well, then $\Sigma_1$ and $\Sigma_2$ are called symmetrically reversible. In this case, $\Sigma_2 = <Q_1,P_1,t_2>$. $\Sigma_1$ and $\Sigma_2$ are called $SR$-transformations for short.

Similarly, in the pure software engineering domain, Balzer (1981) introduces the concept of correctness-preserving transformation aimed at compilable and efficient program production.

We have discussed the concept of reversibility in a context in which some kind of instance

$^9$. The so-called decomposition theorem of the 1NF relational theory (Fagin, 1977) is an example of reversible transformation. Grossly sketched, it states that the schema $\{R(A,B,C); A \rightarrow \rightarrow B\}C$ can be losslessly replaced by $\{R_1(A,B); R_2(A,C)\}$, since, for any instance $r$ of $R$, the relation $t = r[A,B]*r[A,C]$ holds. However, there is no reason for any arbitrary instances $r_1$ of $R_1$ and $r_2$ of $R_2$ to enjoy the inverse property $r_1 = (r_1*r_2)[A,B]$. Therefore, this transformation is not symmetrically reversible. This example and some of its variants are developed in (Hainaut, 1996).
equivalence is preserved. However, the notion of inverse transformation is more general. Any transformation, be it semantics-preserving or not, can be given an inverse. For instance, \( \text{del-ET}(\text{et}_\text{name}) \), which removes entity type with name \( \text{et}_\text{name} \) from its schema, clearly is not a semantics-preserving operation, since its mapping \( t \) has no inverse. However, it has an inverse transformation, namely \( \text{create-ET} \text{(CUSTOMER)} \). Since only the \( T \) part is defined, this partial inverse is called a \textit{structural inverse} transformation.

### 2.6 Proving the reversibility of a transformation

Thanks to the formal semantics of the GER, a proof system has been developed to evaluate the reversibility of a transformation. More precisely, this system relies on a limited set of NF2 transformational operators whose reversibility has been proved, and that can generate a large number of GER transformations. Basically, the system includes five families of transformations, that can be combined to form more complex operators:

- \textit{denotation}, through which a new object set is defined by a derivation rule based on existing structures,
- \textit{project-join} which is a variant of the decomposition theorem,
- \textit{composition} which replaces two relations by one of them and their composition,
- \textit{nest-unnest}, the typical \( 1\text{NF} \leftrightarrow \text{N}1\text{NF} \) operators,
- \textit{container}, that states the equivalence between non-set containers (e.g., bags, lists, arrays) and sets.

Thanks to a complete set of mapping rules between the GER model and the NF2 model in which these basic transformations have been built, the latter can be applied to operational schemas. Fig. 5 shows how we have defined a decomposition operator for normalizing relationship types from the basic project-join transformation. It is based on a 3-step process:

1. the source schema (Fig. 5, top-left) is expressed in the NF2 formalism (bottom-left):
   \[
   \{ \text{entities:} A, B, C; R(A, B, C); A \rightarrow B \}
   \]
2. the basic project-join transformation is applied and yields a normalized relational schema (bottom-right):
   \[
   \{ \text{entities:} A, B, C; R_1(A, B); R_2(A, C); R_1[A]=R_2[A] \}
   \]
3. this NF2 schema is expressed in the GER, leading to the target schema (Fig. 5, top-right).

Since the the GER \( \leftrightarrow \text{NF2} \) mappings are symmetrically reversible and the project-join is an \( SR \)-transformation, the ERA transformation is symmetrically reversible as well. It can be defined as follows:

\[
T_1 = T_{13} o T_{12} o T_{11} \\
T'_1 = T'_{11} o T'_{12} o T'_{13}
\]

We note the important constraint \( R_1[A]=R_2[A] \) that gives the project-join transformation the \( SR \) property, while Fagin's theorem merely defines a reversible operator. We observe how
The reader interested in a more detailed description of this proof system is referred to (Hainaut, 1996).

2.7 The six mutation transformations

A mutation is an SR-transformation that changes the nature of an object. Considering the three main natures of object, namely entity type, relationship type and attribute, six mutation transformations can be defined. In Fig. 6, the couples of operators Σ1 to Σ3, show them applied on typical schema fragments. The transformations Σ4 are not primitive since they can be defined by combining other mutations. However, they have been added due to their usefulness. More sophisticated mutation operators can be defined as illustrated in (Hainaut, 1991) in the range of entity-generating transformations.

2.8 Other basic transformations

The mutation transformations can solve many database engineering problems, but other operators are needed to model special situations. The CASE tool associated with the DB-MAIN methodologies includes a kit of about thirty basic operators that have proved sufficient for most engineering activities. When necessary, user-defined operators can be developed throu-
gh the meta functions of the tool (Hainaut 1996b). We will describe some of the basic operators.

Expressing supertype/subtype hierarchies in DMS that do not support them explicitly is a recurrent problem. The technique of Fig. 7 is one of the most commonly used (Hainaut, 1996c). It consists in representing each source entity type by an independent entity type, then to link each subtype to its supertype through a one-to-one relationship type. The latter can, if needed, be further transformed into foreign keys by application of $\Sigma_2$-direct ($T_2$).

Figure 6 - The six mutation transformations $\Sigma_1$ to $\Sigma_3$. Though not primitive, compound transformations $\Sigma_4$ are shown as well.

Transformations $\Sigma_3$ and $\Sigma_4$ show how to process standard multivalued attributes. When the collection of values is no longer a set but a bag, a list or an array, operators to transform them into pure set-oriented constructs are most useful. Transformations $\Sigma_6$ in Fig. 8 are dedicated

<table>
<thead>
<tr>
<th>source schema</th>
<th>target schema</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Sigma_1$</td>
<td>$\Sigma_1'$</td>
<td>Transforming relationship type $r$ into entity type $R$ ($T_1$) and conversely ($T_1'$). Note that $R$ entities are identified by any couple $(a,b) \in A \times B$ through relationship types $rA$ and $rB$ (id:$rA.A,rB.B$).</td>
</tr>
<tr>
<td>$\Sigma_2$</td>
<td>$\Sigma_2'$</td>
<td>Transforming relationship type $r$ into reference attribute $B.A1$ ($T_2$) and conversely ($T_2'$).</td>
</tr>
<tr>
<td>$\Sigma_3$</td>
<td>$\Sigma_3'$</td>
<td>Transforming attribute $A2$ into entity type $EA2$ ($T_3$) and conversely ($T_3'$).</td>
</tr>
<tr>
<td>$\Sigma_4$</td>
<td>$\Sigma_4'$</td>
<td>Not a primitive operator. $T_4$ can be defined by application of $T_3$ to $EA2.A2$, then of $T_1'$ to $EA2$ in the schema above. Note that the $EA2$ entities depending on the same $A$ entity have distinct $A2$ values (id:$ra2.A,A2$).</td>
</tr>
</tbody>
</table>

from Transformation of Knowledge, Information, and Data: Theory and Applications 25/10/2004
to arrays. Similar operators have been defined for the other types of containers.

<table>
<thead>
<tr>
<th>source schema</th>
<th>target schema</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Diagram 1]</td>
<td>[Diagram 2]</td>
<td>An is-a hierarchy is replaced by one-to-one relationship types. The exclusion constraint (excl:s.C,r.B) states that an A entity cannot be simultaneously linked to a B entity and a C entity. It derives from the disjoint property (D) of the subtypes.</td>
</tr>
</tbody>
</table>

Figure 7 - Transforming an is-a hierarchy into one-to-one relationship types and conversely

<table>
<thead>
<tr>
<th>source schema</th>
<th>target schema</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Diagram 3]</td>
<td>[Diagram 4]</td>
<td>Array A2 (left) is transformed into a multivalued compound attribute A2 (right), whose values are distinct wrt component Index (id(A2):Index). The latter indicates the position of the value (Value). The domain of Index is the range [1..5].</td>
</tr>
</tbody>
</table>

Figure 8 - Converting an array into a set-multivalued attribute and conversely

Attributes defined on the same domain and the name of which suggests a spatial or temporal dimension (e.g., departments, countries, years or pure numbers) are called **serial attributes**. In many situations, they can be interpreted as the representation of an indexed multivalued attributes (Fig. 9). The identification of these attributes must be confirmed by the analyst.

### 3. HIGHER-LEVEL TRANSFORMATIONS

The transformations described in Section 2 are intrinsically atomic: one elementary operator is applied to one object instance, and (Σ4 excluded) none can be defined by a combination of others (orthogonality). This section develops three ways through which more powerful transformations can be developed.
3.1 Compound transformations

A compound transformation is made up of a chain of more elementary operators in which each transformation applies on the result of the previous one. The transformation $\Sigma 8$ in Fig. 10, illustrated by a concrete example, transforms a complex relationship type $R$ into a sort of bridge entity type comprising as many foreign keys as there are roles in $R$. It is defined by the composition of $\Sigma 1$-direct and $\Sigma 2$-direct. This operator is of frequent use in relational database design.

The transformation $\Sigma 9$ is more complex (Fig. 11). It is composed of a chain of four elemen-
tary operators. The first one transforms the serial attributes Expense-2000, ..., Expense-2004 into multivalued attribute Expense comprising sub-attributes Year (the dimension) and Amount (transformation $\Sigma 7$-direct). The second one extracts this attribute into entity type EXPENSE, with attributes Year and Amount (transformation $\Sigma 4$-direct). Then, the same operator is applied to attribute Year, yielding entity type YEAR, with attribute Year. Finally, the entity type EXPENSE is transformed into relationship type expense ($\Sigma 1$-inverse).

<table>
<thead>
<tr>
<th>source schema</th>
<th>target schema</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Sigma 9$</td>
<td>$\Sigma 9'$</td>
<td></td>
</tr>
<tr>
<td>Project</td>
<td>Project</td>
<td></td>
</tr>
<tr>
<td>Dep#</td>
<td>Dep#</td>
<td></td>
</tr>
<tr>
<td>InitialBudget</td>
<td>InitialBudget</td>
<td></td>
</tr>
<tr>
<td>Expense-2000</td>
<td>expense</td>
<td></td>
</tr>
<tr>
<td>Expense-2001</td>
<td>Amount</td>
<td>The serial attributes are first transformed into a multivalued attribute, which in turn is extracted as external entity type EXPENSE. The dimension attribute (Year) is also extracted as entity type YEAR. Finally, EXPENSE is mutated into relationship type expense.</td>
</tr>
<tr>
<td>Expense-2002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expense-2003</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expense-2004</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>YEAR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id: Year</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dom(Year) = [2000..2004]</td>
<td></td>
</tr>
</tbody>
</table>

Figure 11 - Extracting a temporal dimension from serial attributes.

3.2 Predicate-driven transformations

A predicate-driven transformation $\Sigma p$ applies an operator $\Sigma$ to all the schema objects that meet a definite predicate $p$. It will be specified by $\Sigma(p)$. $p$ is a structural predicate that states the properties through which a class of patterns can be identified. Interestingly, a predicate-based transformation can be interpreted as a user-defined elementary operator. Indeed, considering the standard definition $\Sigma = \langle P, Q, t \rangle$, we can rewrite $\Sigma p$ as $\Sigma^* (true)$ where $\Sigma^* = \langle P \land p, Q, t \rangle$. In general, the inverse of $\Sigma p$ cannot be derived from the expression of $\Sigma$ and $p$. Indeed, there is no means to derive the predicate $p'$ that identifies the constructs resulting from the application of $\Sigma p$, and only them.

We give in Fig. 12 some useful transformations that are expressed in the specific language of the DB-MAIN tool, which follows the $\Sigma(p)$ notation. Most predicates are parametric; for instance, the predicate $\text{ROLE}_{\text{per} \ RT}(\langle n1 \rangle \langle n2 \rangle)$, where $\langle n1 \rangle$ and $\langle n2 \rangle$ are integers such that $\langle n1 \rangle \leq \langle n2 \rangle$, states that the number of roles of the relationship type falls in the range $[\langle n1 \rangle..\langle n2 \rangle]$. The symbol “N” stands for infinity.
Figure 12 - Three examples of predicate-driven transformation

3.3 Model-driven transformations

A model-driven transformation is a goal-oriented compound transformation made up of predicate-driven operators. It is designed to transform any schema expressed in model M into an equivalent schema in model M'.

As illustrated in the discussion of the relational model expressed as a specialization of the GER (Fig. 2), identifying the components of a model also leads to identifying the constructs that do not belong to it. Except when M ≤ M', an arbitrary schema S expressed in M may include constructs which violate M'. Each construct that can appear in a schema can be specified by a structural predicate. Let PM denote the set of predicates that defines model M and PM' that of model M'. In the same way, each potentially invalid construct can be specified by a structural predicate. Let P_{MM'} denote the set of the predicates that identify the constructs of M that are not valid in M'. In the DB-MAIN language used in Fig. 12, ROLE_per_RT(3 N) is a predicate that identifies N-ary relationship types that are invalid in DBTG CODASYL databases, while MAX_CARD_of_ATT(2 N) defines the family of multivalued attributes that is invalid in the SQL2 database model. Finally, we observe that each such set as PM can be perceived as a single predicate formed by anding its components.

Let us now consider predicate p ∈ P_{MM'}, and let us choose a transformation Σ = \langle P, Q \rangle such that,

\( (p \Rightarrow P) \land (P_M \Rightarrow Q) \)

Clearly, the predicate-driven transformation Σp solves the problem of invalid constructs defined by p. Proceeding in the same way for each component of P_{MM'} provides us with a series of operators that can transform any schema in model M into schemas in model M'. We call such a series a transformation plan, which is the practical form of any model-driven transfor-
In real situations, a plan can be more complex than a mere sequence of operations, and may comprise loops to process recursive constructs for instance.

In addition, transformations such as those specified above may themselves be compound, so that the set of required transformations can be quite large. In such cases, it can be better to choose a transformation that produces constructs that are not fully compliant with $M'$, but that can be followed by other operators which complete the job. For instance, transforming a multivalued attribute can be obtained by an ad hoc compound transformation. However, it can be thought more convenient to first transform the attribute into an entity type + a one-to-many relationship type ($\Sigma_4$-direct), which can then be transformed into a foreign key ($\Sigma_2$-direct). This approach produces transformation plans which are more detailed and therefore less readable, but that rely on a smaller and more stable set of elementary operators.

The transformation toolset of DB-MAIN includes about thirty operators that have proved sufficient to process schemas in a dozen operational models. If all the transformations used to build the plan have the SR-property, then the model-driven transformation that the plan implements is symmetrically reversible. When applied to any source schema, it produces a target schema semantically equivalent to the former. This property is particularly important for conceptual $\rightarrow$ logical transformations. Fig. 13 sketches, in the form of a script, a simple transformation plan intended to produce SQL2 logical schemas from ERA conceptual schemas. Actual plans are more complex, but follow the approach developed in this section.

<table>
<thead>
<tr>
<th>step</th>
<th>predicate-based transformation</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>transform IS-A relations into one-to-one rel-types</td>
<td>operator $\Sigma_5$-direct;</td>
</tr>
<tr>
<td>2</td>
<td>transform complex rel-types into entity types</td>
<td>operator $\Sigma_1$-direct; complex means N-ary or binary many-to-many or with attributes;</td>
</tr>
<tr>
<td>3</td>
<td>disaggregate level-1 compound attributes</td>
<td>each compound attribute directly depending on an entity type is replaced by its components;</td>
</tr>
<tr>
<td>4</td>
<td>transform level-1 multivalued attributes into entity types</td>
<td>operator $\Sigma_4$-direct; each multivalued attribute directly depending on an entity type is replaced by an entity type;</td>
</tr>
<tr>
<td>5</td>
<td>repeat steps 3 to 4 until the schema does not include complex attributes any more</td>
<td>to cope with multi-level attribute structures;</td>
</tr>
<tr>
<td>6</td>
<td>transform relationship types into reference groups</td>
<td>at this point, only one-to-many and one-to-one rel-types subsist; they are transformed into foreign keys;</td>
</tr>
<tr>
<td>7</td>
<td>if the schema still includes rel-types, add a technical identifier to the relevant entity types and apply step 6</td>
<td>step 6 fails in case of missing identifier; a technical attribute is associated with the entity type that will be referenced by the future foreign key;</td>
</tr>
</tbody>
</table>

Figure 13 - A simple transformation plan to derive a relational schema from any ERA conceptual schema. To make them more readable, the transformations have been expressed in natural language instead of in the DB-MAIN language. The term rel-type stands for relationship type.
It must be noted that this mechanism is independent of the process we are modelling, and that similar transformation plans can be built for processes such as **conceptual normalization** or **reverse engineering**. Though model-driven transformations provide an elegant and powerful means of specification of many aspects of most database engineering processes, some other aspects still require human expertise that cannot be translated into formal rules.

### 4. TRANSFORMATION-BASED DATABASE DESIGN

Most textbooks on database design of the eighties and early nineties propose a five step approach that is sketched in Fig. 14.

**Figure 14 - The standard strategy for database design**

Through the *Conceptual Design* phase, users requirements are translated into a conceptual schema, which is the formal and abstract expression of these requirements. The *Logical Design* phase transforms the conceptual schema into data structures (the logical schema) that comply with the data model of a family of DMS such as relational, OO or standard file data structures. Through the *Physical Design* phase, the logical schema is refined and augmented with technical specifications that make it implementable into the target DMS and that gives it acceptable performance. From the logical schema, users views are derived that meet the requirements of classes of users (*View Design*). Finally, the physical schema and the users views are coded into the DDL of the DMS (*Coding*).
4.1 Database design as a transformation process

Ignoring the view design process for simplification, database design can be modelled by (the structural part of) transformation \( \text{DB-design} \):

\[
\text{code} = \text{DB-design(UR)}
\]

where \( \text{code} \) denotes the operational code and \( \text{UR} \) the users requirements.

Denoting the conceptual, logical and physical schemas respectively by \( \text{CS} \), \( \text{LS} \) and \( \text{PS} \) and the conceptual design, logical design, physical design and coding phases by \( \text{C-design} \), \( \text{L-design} \), \( \text{P-design} \) and \( \text{Coding} \), we can refine the previous expression as follows:

\[
\begin{align*}
\text{CS} &= \text{C-design(UR)} \\
\text{LS} &= \text{L-design(CS)} \\
\text{PS} &= \text{P-design(LS)} \\
\text{code} &= \text{Coding(PS)}
\end{align*}
\]

Clearly, these processes are model-driven transformations and can then be described by transformation plans. The level of formality of these processes depends on the methodology, on the existence of CASE support and of non functional requirements such as performance and robustness, that generally require human expertise. For instance, conceptual design (\( \text{C-design} \)) is a highly informal process based on human interpretation of complex information sources, while logical design can be an automated process completely described by a transformation plan. Anyway, these processes can be decomposed into sub-processes that, in turn, can be modelled by transformations and described by transformation plans, and so forth, until the latter reduce to elementary operators. Three of these processes are worth being examined a bit further.

4.2 Conceptual design

This process includes, among others, two major sub-processes, namely \textit{Basic Analysis}, through which informal or semi-formal information sources are analyzed and their semantic contents are translated into conceptual structures, and \textit{(Conceptual) Normalization}, through which these raw structures are given such additional qualities as readability, normality, minimality, extensibility, compliance with representation standards, etc. (Batini, 1992; Blaha, 1998). This second process is more formal than the former, and is a good candidate for transformational modelling. The plan of Fig. 15, though simplistic, can improve the quality of many raw conceptual schemas.
Figure 15 - A simple transformation plan to normalize ERA conceptual schemas. The term *rel-type* stands for *relationship type*.

### 4.3 Logical Design

As shown in the preceding sections, this process can be specified by a model-based transformation. In fact, we have to distinguish two different approaches, namely ideal and empirical. The *ideal design* produces a logical schema that meets two requirements only: it complies with the target logical model $M$ and it is semantically equivalent to the conceptual schema. According to the transformational paradigm, the logical design process is a $M$-driven transformation comprising SR-operators only. The plan of Fig. 13 illustrates this principles for relational databases. Similar plans have been designed for CODASYL DBTG, Object-relational and XML (Estievenart, 2003) databases, among others. *Empirical design* is closer to the semi-formal way developers actually work, relying on experience and intuition, rather than on standardized procedures. Other requirements such as space and time optimization often are implicitly taken into account, making formal modelling more difficult, if not impossible. Though no comprehensive model-driven transformations can describe such approaches, essential fragments of empirical design based on systematic and reproducible rules can be described by compound or predicate-driven transformations.

### 4.4 Coding

Quite often overlooked, this process can be less straightforward and more complex than generally described in the literature or carried out by CASE tools. Indeed, any DMS can cope with a limited range of structures and integrity constraints for which its DDL provides an explicit...
5. TRANSFORMATION-BASED DATABASE REVERSE ENGINEERING

Database reverse engineering is the process through which one attempts to recover or to rebuild the technical and functional documentation of a legacy database. Intensive research in the past decade have shown that reverse engineering generally is much more complex than initially thought. We can put forward two major sources of difficulties, derived from the analysis of Section 4. First, empirical design has been, and still is, more popular than systematic design. Secondly, only the codeddl part of the code provides a reliable description of the database physical constructs.

Empirical design itself accounts for two understanding problems. First, it often relies on non-standard, unpublished, translation rules that may be difficult to interpret. Secondly, actual logical schemas often are strongly optimized, so that extracting a conceptual schema from the logical schema involves understanding not only how the latter has been translated in the target model, but also how, and according to which criteria, it has been optimized.

The codeddl component expresses a part of the physical schema only. Therefore, the codeext part must be retrieved and interpreted, which leads to two independent problems. The first one requires parsing a huge volume of program code to identify code sections that cope with implicit, i.e., undeclared, constructs such as decomposed (flattened) fields or referential constraints. The second problem concerns the correct interpretation of these code fragments, that translates into constructs to be added to the physical schema.

The whole process is described in Fig. 16. It shows that database reverse engineering is decomposed into two main sub-processes, namely Extraction and Conceptualization. The objective of the Extraction process is to recover the complete logical schema of the legacy database. It includes three activities: Parsing the DDL code to extract the raw physical schema, schema Refinement through which implicit and hidden constructs are elicited from external code (as well as from other sources, such as the data themselves, but we will ignore them in this discussion) and Cleaning, in which the technical constructs of the physical schema are
removed.

Figure 16 - The main processes of database reverse engineering

The second main sub-process, Conceptualization, is intended to derive a plausible conceptual schema from the logical schema. It consists in identifying the trace of the translation of conceptual constructs, then in replacing them with their source. For instance, a foreign key is interpreted as (i.e., replaced by) a many-to-one relationship type.

The transformational interpretation of the reverse engineering process is straightforward:

\[
CS = DBRE(code)
\]

where \( code \) denotes the operational code and \( CS \) the conceptual schema.

DBRE can be developed as follows:

\[
LS = Extraction(code) \\
CS = Conceptualization(LS)
\]

Extraction itself includes three processes:

\[
PS = Parsing(code_{dd}) \\
PS = Refinement(PS, code_{ex}) \\
LS = Cleaning(PS)
\]

By comparing the transformational expression of the Database design and Database reverse engineering processes, we can state the following equivalence rules, in which, as usual, \( \Sigma' \) denotes the inverse of transformation \( \Sigma \):

\[
\Sigma'(LS) = \Sigma(CS)
\]

\[
\Sigma'(PS) = \Sigma(PS')
\]

\[
\Sigma'(Cleaning) = \Sigma(Clearing)^{-1}
\]

\[
\Sigma'(Extraction) = \Sigma(Extraction)^{-1}
\]
Conclusions and Perspectives

In this chapter, we have shown that schema transformation can be used as a major paradigm in database engineering. In particular, being formally defined, it can be used to precisely model complex processes and to reason on their properties such as semantics preservation. It has also been used to derive new processes from former ones, as illustrated by the formalization of database reverse engineering as the inverse of database design.

Due to their formality, transformations can be implemented in CASE tools, either as implicit operators, or as tools that are explicitly made available to the developer. Two implementations are worth being mentioned, namely (Rosenthal, 1994) and (Hainaut, 1996b). The latter reference describes the DB-MAIN CASE environment which includes a transformation toolbox as well as special engines for user-defined predicate-driven and model-driven transformations. Further information can be found at http://www.info.fundp.ac.be/libd

Transformations also have a great potential in other domains such as database interoperability, in which mediation between existing databases (McBrien, 2003) and data wrapping (Thirran, 2001) can be formalized and automated thanks to the use of transformational operators. In this domain, data instance transformations are modelled by the \( t \) part of the transformations. Specifying how the source schema is transformed into the target schema automatically provides a chain of instance transformation that are used to generate the data conversion code that is at the core of data migrators (ETL processors), wrappers and mediators.
7. REFERENCES


from *Transformation of Knowledge, Information, and Data: Theory and Applications* 25/10/2004


