Transformation-Based Database Engineering

Jean-Luc Hainaut
University of Namur, Belgium

1. 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 and Steinbrüggen (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 a special kind 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.

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 & Stickel, 1995), DBMS schema translation (Rosenthal & Reiner, 1994), schema integration (McBrien & Poulovassilis, 2003), schema equivalence (Jajodia, Ng, & Springsteel, 1983; Kobayashi, 1986), data conversion (Navathe, 1980), reverse engineering (Casanova & Amaral De Sa, 1984; Hainaut, Chandelon, Tonneau, & Joris, 1993), schema optimization (Halpin & Proper, 1995) database interoperability (McBrien & Poulovassilis, 2003; Thiran & Hainaut, 2001) and others. The reader will find in (Hainaut, 1995) an illustration of numerous application domains of schema transformations.

Though it has been explored for more than 25 years, this concept has only been gaining wider acceptance for two years, as witnessed by the recent references Omelayenko and Klein (2003) and van Bommel (2005).

The goal of this paper is to develop and illustrate a general framework for database transformations in which most processes mentioned above can be formalized and analyzed in a uniform way. Section 2 describes the basics of schema transformations. Section 3 explains how practical transformations can be used for database engineering . The database design and reverse engineering processes are revisited in Section 4, where we give them a transformational interpretation. Section 5 concludes the chapter.

2. BACKGROUND

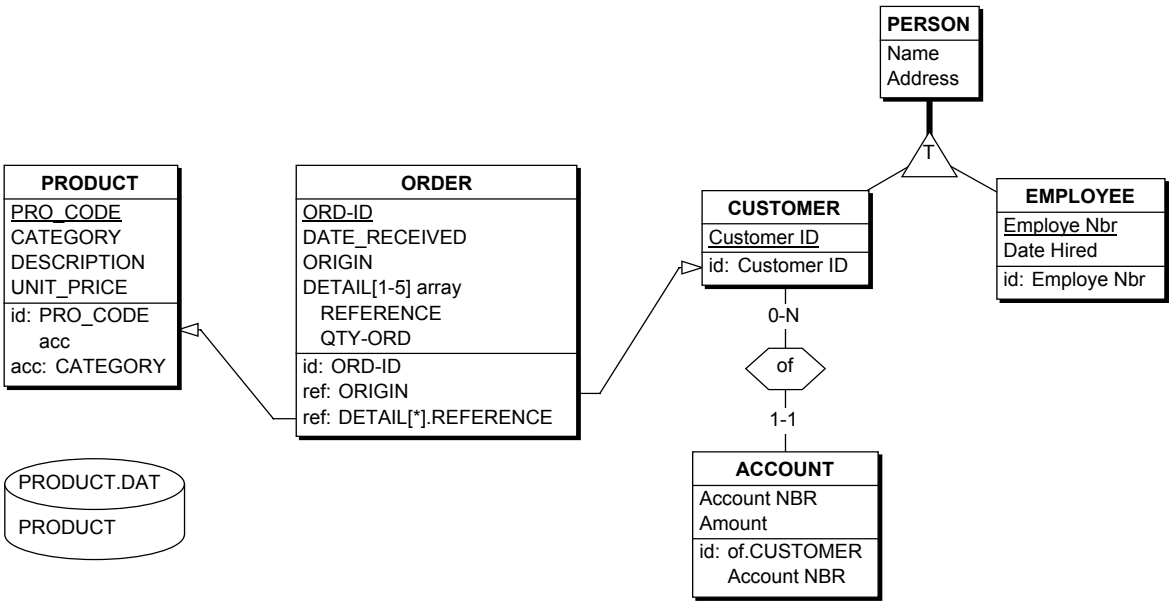
This section describes a general transformational theory that will be used as a basis for modelling database engineering processes. First, we define a wide-spectrum model from which operational models (i.e., those which are of interest for practitioners) can be derived. Then, we describe the concept of transformation and its semantics-preserving property.

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 within a single model, and require intra-model transformations. Other processes, such as logical design, use two distinct models, namely source and target. 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 Generic Entity-Relationship model (GER) is a wide-spectrum formalism intended to encompass most popular operational models, whatever their abstraction level and their underlying paradigms (Hainaut, 1996).

The GER includes, among others, the concepts of schema, entity type, entity collection, domain, attribute, relationship type, key, as well as various constraints. In this model, a schema is a description of data structures (Figure 1).

***Figure 1** - A typical hybrid schema made up of conceptual constructs (e.g., entity types* PERSON, CUSTOMER, EMPLOYEE *and* ACCOUNT*, relationship type of, identifiers* Customer ID *of* CUSTOMER*), logical constructs (e.g., record type* ORDER*, with various kinds of fields including an array, foreign keys* ORIGIN *and* DETAIL.REFERENCE*) and physical objects (e.g., table* PRODUCT *with primary key* PRO_CODE *and indexes* PRO_CODE *and* CATEGORY*, table space* PRODUCT.DAT*). The identifier of* ACCOUNT*, stating that the accounts of a customer have distinct Account numbers, making* ACCOUNT *a dependent or weak entity type. The cardinality constraint of a role follows the* participation *interpretation and not the UML* look-across *semantics.*



It is made up of specification constructs which can be, for convenience, classified into the usual three abstraction levels, namely conceptual, logical and physical:

- A *conceptual schema* comprises entity types, super/subtype (isa) hierarchies, relationship types, roles, attributes (multi/single-valued; atomic/compound), identifiers (or unique keys) and various constraints.
- A *logical schema* comprises such constructs as record types, fields, arrays, foreign keys, redundant fields, 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.

Since it includes the main concepts of most operational models, the GER can be used to precisely specify each of them thanks to a *specialization* mechanism. According to the latter, each construct of model M is a specialization of a construct of the GER model. For instance, tables, columns, primary keys and foreign keys are specializations of, respectively, entity types, attributes, primary identifiers and referential attributes. In addition, schemas in M must satisfy specific assembly rules, such as the following: *each entity type has at least one attribute*. As an important consequence, all intra- and inter-model transformations are specializations of GER-to-GER transformations. For instance, the standard ERA-to-SQL transformation is modelled by a chain of transformations from the ERA specialization of the GER to its SQL specialization.

The GER model has been given a formal semantics in terms of an extended NF2 model (Hainaut, 1996). This semantics allows 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.

Transformation: Definition

The definitions that will be stated are model-independent. In particular, they are valid for the GER model, so that the examples will be given in the latter. We 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.

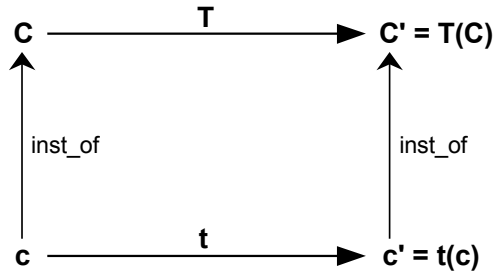
A **transformation Σ** consists of two mappings **T** and **t** (Figure 2):

- **T** is the *structural mapping* 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 Σ.
- **t** is the *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' = t(c) is the corresponding instance of T(C). **t** can be specified

through any algebraic, logical or procedural expression.

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

Figure 2 - The two mappings of schema transformation Σ ≡ <T,t>. The inst_of arrow from x to X indicates that x is an instance of X.



Each transformation Σ is associated with an inverse transformation Σ' which can undo the result of the former under certain conditions.

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.

- A transformation Σ1 = <T1,t1> = <P1,Q1,t1> is *reversible*, *iff* there exists a transformation Σ2 = <T2,t2> = <P2,Q2,t2> such that, for any construct C, and any instance c of C: P1(C) ⇒ ([T2(T1(C))=C] and [t2(t1(c)=c]). Σ2 is the inverse of Σ1, but the converse is not true. For instance, an arbitrary instance c' of T(C) may not satisfy the property c'=t1(t2(c')).
- If Σ2 is reversible as well, then Σ1 and Σ2 are called *symmetrically reversible*. In this case, Σ2 = <Q1,P1,t2>. Σ1 and Σ2 are called *SR-transformations* for short.

Thanks to the formal semantics of the GER, a proof system has been developed to evaluate the reversibility of a transformation (Hainaut, 1996).

3. TRANSFORMATIONS FOR DATABASE ENGINEERING

In this section, some important basic transformations are described, as well as higher-level operators.

Basic 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 (Fig. 3). Mutations can solve many database engineering problems, but other operators are needed to model special situations. Fig. 3/bottom shows how a supertype/subtype hierarchy can be transformed by representing each source entity type by an independent entity type, then linking 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 Σ2-direct (T2). Field studies suggest that about thirty basic operators suffice for most engineering activities.

Higher-level Transformations

The transformations described so far are intrinsically atomic: one elementary operator is applied to one object instance, and none can be defined by a combination of others. The next sections describe two ways through which more powerful transformations can be developed.

Predicate-driven Transformations

A predicate-driven transformation Σp applies an operator Σ to all the schema objects that meet a definite predicate p. It is specified by Σ(p) where p is a *structural* predicate that states the properties through which a class of structures can be identified.

We give in Figure 4 some useful transformations that are expressed in the specific language of the DB-MAIN tool (Hainaut, Englebert, Henrard, Hick, & Roland, 1996), which follows the Σ(p) notation. Most predicates are parametric; for instance, the predicate ROLE_per_RT(<n1> <n2>), where <n1> and <n2> are integers, states that the number of roles of the relationship type falls in the range [<n1>..<n2>]. The symbol "N" stands for infinity.

We give in Figure. 4 some useful transformations that are expressed in the specific language of the DB-MAIN tool (Hainaut, Englebert, Henrard, Hick, & Roland, 1996), which follows the Σ(p) notation.

Most predicates are parametric; for instance, the predicate ROLE_per_RT(<n1> <n2>), where <n1> and <n2> are integers, states that the number of roles of the relationship type falls in the range [<n1>..<n2>]. The symbol "N" stands for infinity.

Model-driven Transformations

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

Figure 3 - The six mutation transformations Σ1 to Σ3. Σ4 transforms an is-a hierarchy into one-to-one relationship types and conversely. The term *rel-type* stands for *relationship type*.

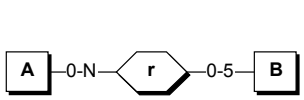
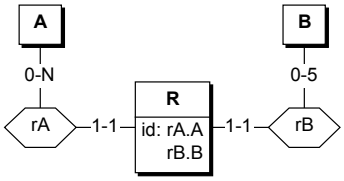
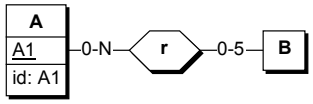
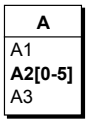
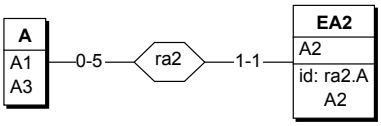
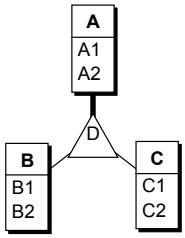
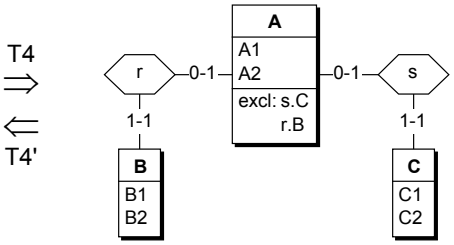
	source schema		target schema	comment
Σ1				Transforming rel-type r into entity type R (T1) and conversely (T1'). Note that R entities are identified by any couple (a,b) ∈ AxB through rel-types rA and rB (id:ra.A,rB.B).
Σ2				Transforming rel-type r into reference attribute B.A1 (T2) and conversely (T2').
Σ3				Transforming attribute A2 into entity type EA2 (T3) and conversely (T3'). Note that the EA2 entities depending on the same A entity have distinct A2 values (id:ra2.A,A2).
Σ4				An is-a hierarchy is replaced by one-to-one rel- types. The exclusion constraint (excl:s.C,r.B) states that an A entity cannot be simultane-ously linked to a B entity and a C entity. It derives from the disjoint property (D) of the subtypes.

Figure 4 - Three examples of predicate-driven transformation

predicate-driven transformation	interpretation
RT_into_ET(ROLE_per_RT(3 N))	transform each rel-type R into an entity type (RT_into_ET), if the number of roles of R (ROLE_per_RT) is in the range [3..N]; in short, *convert all N-ary rel-types into entity types*.
RT_into_REF(ROLE_per_RT(2 2) and ONE_ROLE_per_RT(1 2))	transform each rel-type R into reference attributes (RT_into_REF), if the number of roles of R is 2 and if R has from 1 to 2 one role(s), i.e., R has at least one role with max cardinality 1; in short, *convert all one-to-many rel-types into foreign keys*.
INSTANTIATE(MAX_CARD_of_ATT(2 4))	transform each attribute A into a sequence of single-value instances, if the max cardinality of A is between 2 and 4; in short, *convert multivalued attributes with no more than 4 values into serial attributes*.

Identifying the components of a model also leads to identifying the constructs that do not belong to it. An arbitrary schema S expressed in M may include constructs which violate M'. Each class of constructs that can appear in a schema can be specified by a structural predicate. Let P_M denote the set of predicates that defines model M and $P_{M'}$ that of model M'. In the same way, each potentially invalid construct can be specified by a structural predicate. Let $P_{M/M'}$ 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. 4, 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 multivalued attributes that are invalid in the SQL2 database model. Finally, we observe that P_M can be perceived as a single predicate formed by *anding* its components.

Let us now consider predicate $p \in P_{M/M'}$, and let us choose a transformation $\Sigma = \langle P,Q \rangle$ such that,

$$(p \Rightarrow P) \wedge (P_{M'} \Rightarrow Q)$$

Clearly, the predicate-driven transformation Σp solves the problem of invalid constructs defined by p. Proceeding in the same way for each component of $P_{M/M'}$ provides us with a chain of operators that can transform any schema in model M into a schema in model M'. We express such a

chain through a *transformation plan*, which is the practical form of any model-driven transformation.

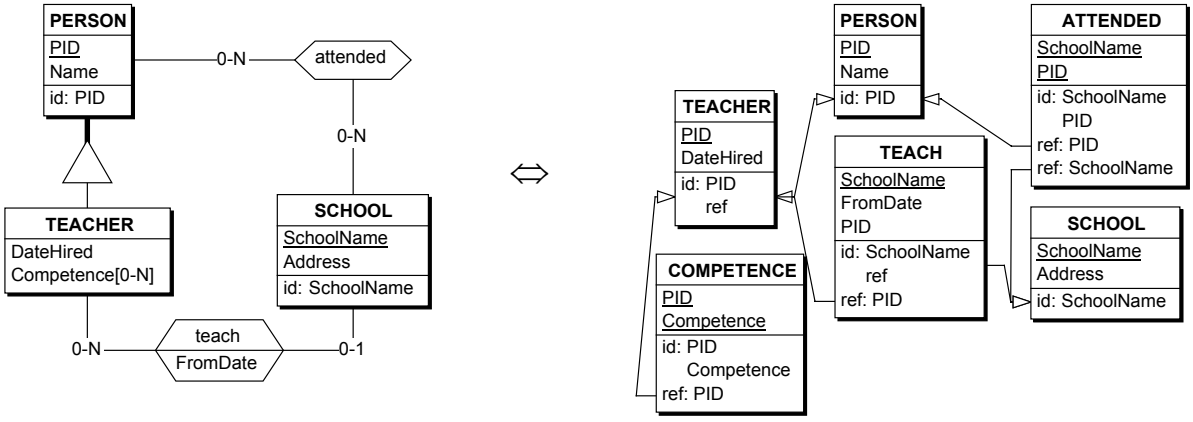
If a plan comprises SR-properties only, then the model-driven transformation that it implements is symmetrically reversible. When applied to any source schema, it produces a target schema semantically equivalent to the former. Figure 5 sketches a simple transformation plan intended to produce SQL2 logical schemas from ERA conceptual schemas. Actual plans are more complex, but follow this approach.

Figure 6 shows the application of this plan to a small ERA schema. 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.

Figure 5 - 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.

step	predicate-based transformation	comment
1	transform IS-A relations into one-to-one rel-types	operator Σ4-direct;
2	transform complex rel-types into entity types	operator Σ1-direct; *complex* means N-ary or binary many-to-many or with attributes;
3	disaggregate level-1 compound attributes	each compound attribute directly depending on an entity type is replaced by its components;
4	transform level-1 multivalued attributes into entity types	operator Σ3-direct; each multivalued attribute directly depending on an entity type is replaced by an entity type;
5	repeat steps 3 to 4 until the schema does not include complex attributes any more	to cope with multi-level attribute structures;
6	transform relationship types into reference groups	at this point, only one-to-many and one-to-one rel-types subsist; they are transformed into foreign keys (Σ2-direct);
7	if the schema still includes rel-types, add a technical identifier to the relevant entity types and apply step 6	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;

Figure 6 - Transforming a conceptual schema into a SQL2-compliant logical schema through the plan of Figure 5



4. NEW PERSPECTIVES IN DATABASE ENGINEERING PROCESS MODELLING

The transformational approach provides us with an elegant paradigm through which standard and non-standard engineering processes can be revisited and built.

Transformation-based Database Design

Figure 7. describes the standard approach to database design as the transformation of users requirements into DDL code. This transformation itself comprises five processes that, in turn are transformations. Ignoring the view design process for simplicity, database design can be

modelled by (the structural part of) transformation **DB-design**:

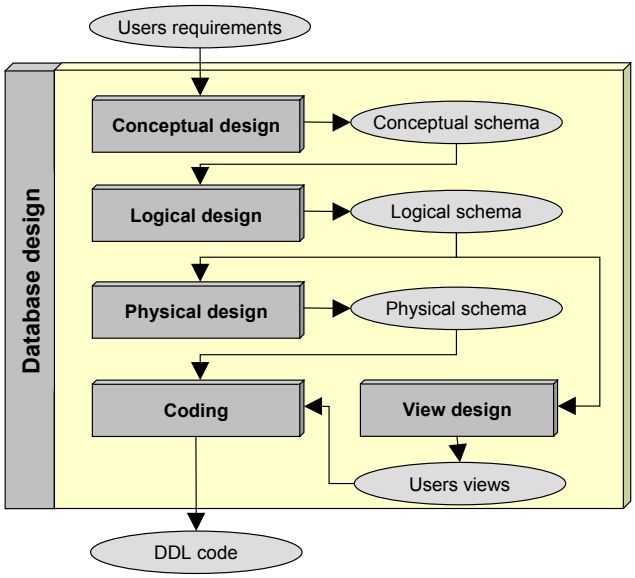
DDL-code = DB-design(Users-requirements)

We can refine this expression as follows:

Conceptual-schema = Conceptual-design(Users-requirements)
Logical-schema = Logical-design(Conceptual-schema)
Physical-schema = Physical-design(Logical-schema)
DDL-code = Coding(Physical-schema)

Clearly, all 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 availability of a CASE tool and on non functional requirements such as performance and robustness, that generally require human expertise.

Figure 7 - The standard strategy for database design



For instance, conceptual design is a highly informal process based on human interpretation of complex information sources, while logical design can be completely described by a transformation plan (such as that of Figure 5 for relational databases). 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.

Transformation-based Database Reverse Engineering

Modelling database design in this way provides us with an elegant basis to describe database reverse engineering. Indeed the latter can be perceived to be the inverse of database design. An in-depth analysis of reverse engineering can be found in (Hainaut, 2002).

5. CONCLUSIONS

In this chapter, we have shown that schema transformations 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 and Reiner (1994) and Hainaut et al. (1996). 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 & Poulovassilis, 2003) and data wrapping (Thiran & Hainaut, 2001) can be formalized and automated through 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.

6. REFERENCES

Balzer, R. (1981). Transformational implementation : An example. IEEE TSE, Vol. SE-7(1).

Casanova, M., & A., Amaral De Sa. (1984). Mapping uninterpreted Schemes into Entity-Relationship diagrams : two applications to conceptual schema design. IBM J. Res. & Develop., 28(1).

Fikas, S., F. (1985). Automating the transformational development of software, IEEE TSE, Vol. SE-11.

Hainaut, J-L., Chandelon M., Tonneau C., & Joris M. (1993). Contribution to a Theory of Database Reverse Engineering. Proc. of the IEEE Working Conf. on Reverse Engineering, Baltimore, May 1993, IEEE Computer Society Press, Los Alamitos, CA, pp. 161-170.

Hainaut, J.-L. (1995, September). Transformation-based database engineering. VLDB'95 Tutorial notes. Pub.

University of Zürich, Switzerland, 200 pages. Retrieved August 2005 from *http://www.info.fundp.ac.be/libd*.

Hainaut, J-L. (1996). Specification preservation in schema transformations - application to semantics and statistics, Data & Knowledge Engineering, 11(1).

Hainaut, J-L, Englebert, V., Henrard, J., Hick J-M., & Roland, D. (1996b). Database Reverse Engineering : from Requirements to CASE tools. Journal of Automated Software Engineering, 3(1).

Hainaut, J.-L. (2002). Introduction to database reverse engineering In *LIBD lecture notes*, Pub. University of Namur, Belgium, 160 pages. Retrieved August 2005 from *http://www.info.fundp.ac.be/~dbm/publication/2002/DBRE-2002.pdf*

Halpin, T., A., & Proper, H., A. (1995). Database schema transformation and optimization. Proc. of the 14th Int. Conf. on ER/OO Modelling (ERA), Springer LNCS, vol. 1021, Berlin/Heidelberg, Germany, pp. 191-203.

Jajodia, S., Ng, P., A., & Springsteel, F., N. (1983). The problem of Equivalence for Entity-Relationship Diagrams, IEEE Trans. on Soft. Eng., SE-9(5).

Kobayashi, I. (1986). Losslessness and Semantic Correctness of Database Schema Transformation : another look of Schema Equivalence, Information Systems, 11(1), 41-59.

McBrien P., & Poulovassilis, A. (2003). Data integration by bi-directional schema transformation rules, Proc 19th International Conference on Data Engineering (ICDE'03), IEEE Computer Society Press, Los Alamitos, CA, p. 227-238.

Navathe, S., B. (1980). Schema Analysis for Database Restructuring, ACM TODS, 5(2), June 1980.

Omelayenko, B., & Klein, M. (eds.). (2003). Knowledge Transformation for the Semantic Web. IOS Press, Amsterdam, The Netherlands.

Partsch, H., & Steinbrüggen, R. (1983). Program Transformation Systems. Computing Surveys, 15(3).

Rauh, O., & Stickel, E. (1995). Standard Transformations for the Normalization of ER Schemata. Proc. of the CAiSE•95 Conf., Jyväskylä, Finland, LNCS, Springer-Verlag, 932, Berlin/Heidelberg, Germany, pp. 313-326.

Rosenthal, A., & Reiner, D. (1994). Tools and Transformations - Rigourous and Otherwise - for Practical Database Design, ACM TODS, 19(2).

Thiran, Ph., & Hainaut, J-L. (2001). Wrapper Development for Legacy Data Reuse. Proc. of WCRE'01, IEEE Computer Society Press, Los Alamitos, CA, pp. 198-207.

van Bommel, P., (ed.) (2005). *Transformation of Knowledge, Information and Data: Theory and Applications*, IDEA Group Pub.

7. KEY TERMS

Transformational software engineering: A view of software engineering through which the production and evolution of software can be modelled, and practically carried out, by a chain of transformations which preserve some essentiel properties of the source specifications. Program compiling, but also transforming tail recursion into an iterative pattern are popular examples. This approach is currently applied to software renovation, reverse engineering and migration.

Transformation: A rewriting rule through which the instances of some pattern of an abstract or concrete specification are replaced with instances of another pattern. The concept also applies on database schemas where the rewriting rule replaces a set of constructs of a database schema with another set of constructs. Such a transformation comprises two parts: a schema rewriting rule (structural mapping) and a data conversion rule (instance mapping). The latter transforms the data according to the source schema into data complying with the target schema.

Mutation transformation: A schema transformation that changes the nature (entity type, relationship type, attribute) of a schema construct. For instance, an entity type is transformed into a relationship type and conversely.

Semantics-preserving transformation: A schema transformation that does not change the information contents of the source schema. Both schemas describe the same universe of discourse.

Inverse transformation: Considering schema transformation Σ, the transformation Σ' is the inverse of Σ if its application undoes the result of the application of Σ. The instance mapping of Σ' can be use to undo the effect of applying the instance mapping of Σ on a set of data.

Predicate-driven transformation: A couple $<\Sigma,p>$ where Σ is a transformation and p a structural predicate that identifies schema patterns. The goal of a predicate-based transformation is to apply Σ to the patterns that meet p in the current schema.

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