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

TRAMIS : a transformation-based database CASE tool

J-L Hainaut, M. Cadelli, B. Decuyper, O. Marchand

Institut d'Informatique - University of Namur
rue Grandgagnage, 21 - B-5000 Namur (Belgium)
tel. : +32 81/724996
e-mail : jlh@info.fundp.ac.be

Abstract

The paper describes some important aspects of TRAMIS, an database CASE tool that has been developed at the University of Namur. This tool is to provide designers with more freedom than currently available in most CASE tools thanks to its unique generic data model, to its transformational principles, and to its toolbox architecture. The paper develops in some details the transformational aspects of the tool, showing how executable schemas can be produced by successive transformations of an initial conceptual schema, and how TRAMIS operators provide designers with a three-level toolbox of transformation operators.

Keywords : database design, CASE tools, transformational approach

Résumé

L'article présente les aspects les plus importants de TRAMIS, un atelier logiciel destiné à la conception de bases de données, et développé par les Facultés Universitaires de Namur. Grâce à un modèle de spécification générique, à son architecture de boite à outils, et surtout à ses fonctions transformationnelles, cet atelier offre au concepteur une liberté de comportement qu'on ne trouve pas dans les environnements disponibles sur le marché. L'article met l'accent sur les aspects transformationnels de l'atelier, montrant comment des schémas de bases de données opérationnels (exécutables) peuvent être obtenus par des transformations successives d'un schéma conceptuel. On détaille également les trois niveaux d'outils de transformation.

Mots clés

conception de bases de données, atelier de génie logiciel, approche transformationnelle

1. INTRODUCTION

Most current database CASE tools provide four major families of functions for their users, namely conceptual specifications acquisition and management, conceptual specifications validation, reporting and executable code generation. They concentrate mainly on conceptual specification step, leaving the problem of producing efficient physical schemas practically unsolved. They are based on simple and rigid strategies that give the illusion that database design is a straighforward and deterministic process once the conceptual schema has been developed. Indeed, many CASE tools propose a *draw-and-generate* strategy that leads to poor and sometimes unreadable DBMS schemas. More sophisticated strategies that would allow the integration of more realistic requirements (time efficiency, space efficiency, distribution, privacy, modularity, hardware constraints, etc) are impossible. Even AI-based systems [3] [5] fail to tackle this problem successfully.

These weaknesses lead designers to two unacceptable practices : integrating these requirements into the conceptual schema, or modifying the generated DDL text with a text processor.

TRAMIS is an experimental CASE tool[1] that proposes a different functional architecture that can support various design strategies according to the skill of the designer and the requirements to be satisfied by the final product. It is based on a transformational approach according to which most database design activities are based on formally defined **schema transformations**. This approach, that was so far mainly theoretical [4], is getting more and more practical acceptance as testified in [2] for instance. This paper describes these principles from a practical point of view, and illustrates them through an example of design scenario. The reader interested in a more general presentation of the principles can consult [13].

Organization of the paper is as follows. Section 2 states the basic hypothesis about practical design behaviours on which TRAMIS is defined. Section 3 defines briefly the generic specification model through which database structures are expressed. Section 4 develops the fundamental concept of schema transformation. Section 5 is dedicated to the architecture of the TRAMIS CASE tool, while section 6 illustrates the principles with an example of database design scenario.

Let's mention finally that the reader is supposed to be somewhat familiar with the problem of database design, and particularly with entity-relationship modelling.

2. THE METHODOLOGICAL PRINCIPLES OF TRAMIS

Currently, standard database design methods [8][9] [2] propose a strict layered approach, according to which the conceptual schema has to be built from users requirements (conceptual analysis), then this schema is translated into a logical schema (logical design) which in turn is transformed into a physical schema (physical design).

The basic hypotheses that underly the architecture of TRAMIS concern the way designers *really* behave when they build software (e.g. database) systems, do they use standard methods or not. We argue that software design is basically a problem-solving activity that can be characterized as follows :

- there is a **limited set of problems** to be solved;
- each problem is related to specific functional (user- or business-oriented) or non-functional (mainly technical) **requirements**;
- solving a problem is done by a **design process** (or activity) the objective of which is to make the current state of the specifications (or design product) **satisfy the corresponding requirements** (e.g. to normalize a schema, or to make the database schema optimized according to disk space);
- to much extend, a problem can be **solved independently** of the others through a limited (though not always small) set of rules, reasonings and heuristics;
- most design processes can be perceived as **specifications transformations** that produce output products from input products (e.g. transform a collection of users views into a unique conceptual schema);
- a **specific design method**, such as MERISE or SADT, can be described as a suggested arrangement of design processes carried out on specific products, according to specific sets of requirements;
- in practice, the ways designers build software systems do not always strictly follows standard methods, even when such a method is suggested or enforced; hence the notions of formal method and heuristic method. A designer that blindly follows a MERISE-like method is said to use a formal method[2], while a

---

1 A commercial version of the first prototype has been distributed by CONCIS under the name TRAMIS/Master.

2 Although MERISE as well as most popular methods can only be recognized as *semi-formal* methods.

programmer that defines a database by directly encoding the SQL schema uses a heuristic method. However, both are supposed to solve, explicitly or implicitly, the same problems related to the future database. This observation is particularly important when considering reverse engineering [12].

TRAMIS is intended to support, at least partly, these problem-solving design behaviours by means of four basic principles that allow to define and carry out both standard and heuristic methods for designing files and databases. These principles are a *unique generic specification model* that allows the definition of a large variety of specific design products, *transformational functions* as major database design tools, a *toolbox architecture*, allowing maximal independence between functions, and *multiple model definition* through parametrization of the unique generic model. Due to the limited scope of this paper, we will concentrate mainly on the transformation functions. Further details on the other aspects can be found in [13].

3. THE GENERIC MODEL

A database schema, whatever its state and its abstraction level, is described in TRAMIS as an entity-relationship schema. In short, conceptual schemas as well as physical schemas are expressed into a unique, generic, *extended entity-relationship model*. The advantage of using a unique formalism throughout the database life cycle is threefold.

First, it lowers the cognitive effort required from the designer, since one model only has to be understood and mastered instead of two or three. For instance, many methods make use of an entity-relationship model at the conceptual level, a Bachman- or CODASYL-like model at the logical level, and, say, a relational model at the physical level.

Second, it is the ideal support for a general transformational toolset. Indeed, transformations can be used at any abstraction level. For instance, the same schema transformation can be used to normalize a conceptual schema and to optimize a CODASYL schema. In addition, intra-model transformations are easier to define, to understand and even to implement than inter-model ones.

Finally, since no fixed set of design products are defined a priori, such as conceptual, logical, normalized, optimized, DB2-compliant, etc, schemas, the definition of both standard and non-standard design methods is made quite easy.

The concepts of the model are illustrated in figures 1 to 3, and briefly enumerated herebelow. A more comprehensive description can be found in [13] while its formal definition has been developed in [10].

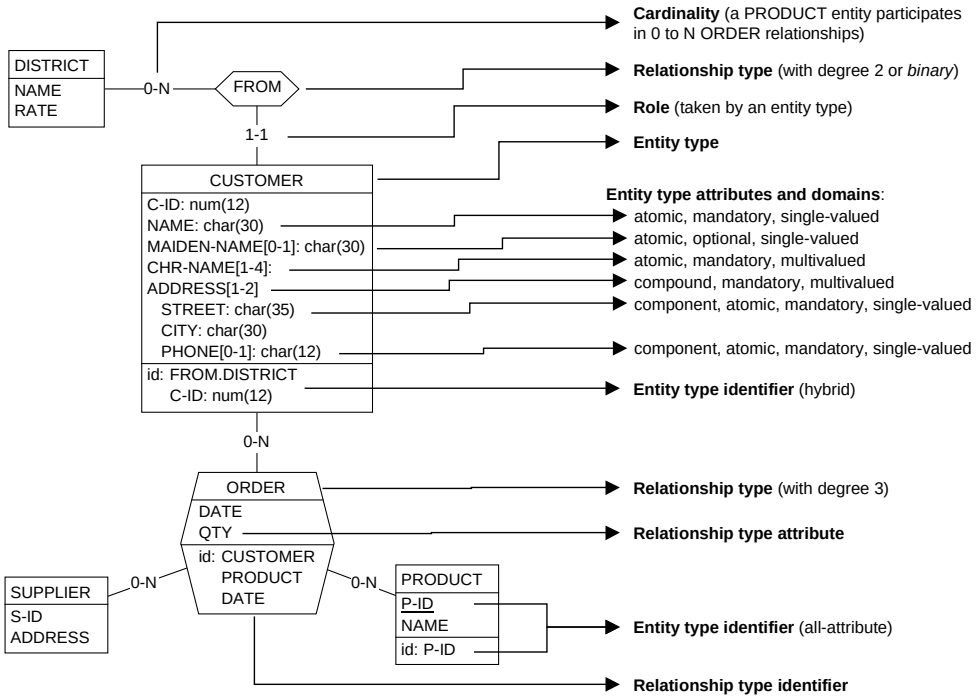

Figure 1 *- Illustration of the main constructs of a TRAMIS conceptual schema*

A *TRAMIS conceptual schema* comprises **entity types** (representing classes of real world abstract or concrete objects), **relationship types** (representing classes of associations between real world objects) and **attributes** (modelling properties of objects or associations). A relationship type comprises two or more **roles**, each of them being taken by an entity type, and characterized by **cardinality constraints** that state in how many relationships (min-max) an entity must (min) and can (max) play this role. An entity type has zero, one or several attributes. Entities and relationships can be identified in their class by zero, one or several **identifiers**. An identifier is made up of attributes and/or roles. An attribute is single-valued or multivalued, optional or mandatory, atomic or compound. Figure 1 illustrates these concepts.

When incorporating specifications about more technical aspects of the data structures, a TRAMIS schema corresponds to some kind of logical schema. It may comprise additional constructs such as **access keys** (abstraction of value-based access mechanisms such as indices, hash files, etc), **access paths** (expressing that a relationship type can be used by application programs to navigate into the database), **spaces** (abstraction of files, table spaces, data sets, etc), and **inclusion constraints** (a special case of which is the referential integrity constraint[3]). Figure 2 illustrates these concepts.

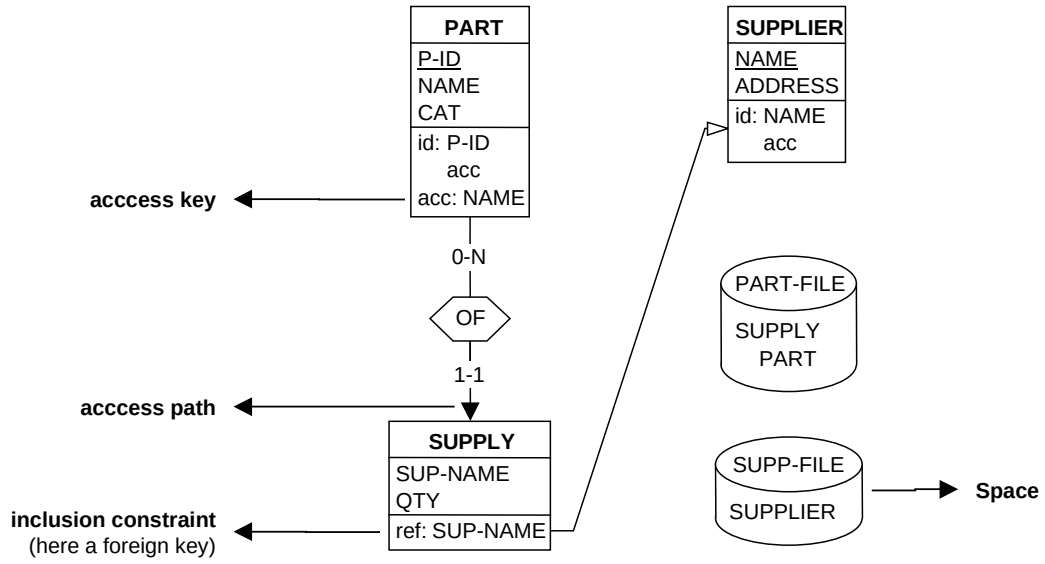

Figure 2 *- Illustration of major additional building blocks of a TRAMIS logical schema*

Any TRAMIS schema can be complemented with additional specifications concerning informal descriptions and statistics. **Informal descriptions** give either the semantic interpretation of each schema object or information about design choices (called *Technical notes*). **Static statistics** describe for instance the size of objects populations, the average length and the average number of occurrences of attribute values, the average participation of entities in roles or the average number of key values. **Dynamic statistics** quantify the usage of data : for each elementary update and access operation, the average number of activations per time unit, and the average number of entities processed per activation is specified. The static statistics model of TRAMIS is described in [14]. Figure 3 illustrates these concepts. In addition, objects can have three names, a *natural name*, a *short-name* (useful for building the names of automatically defined objects) and a *technical name*, used for DDL generation.

[3] In fact, these constraints basically belong to conceptual specifications. However, since they will appear mostly in relational or standard file schemas, they have been classified into this category.

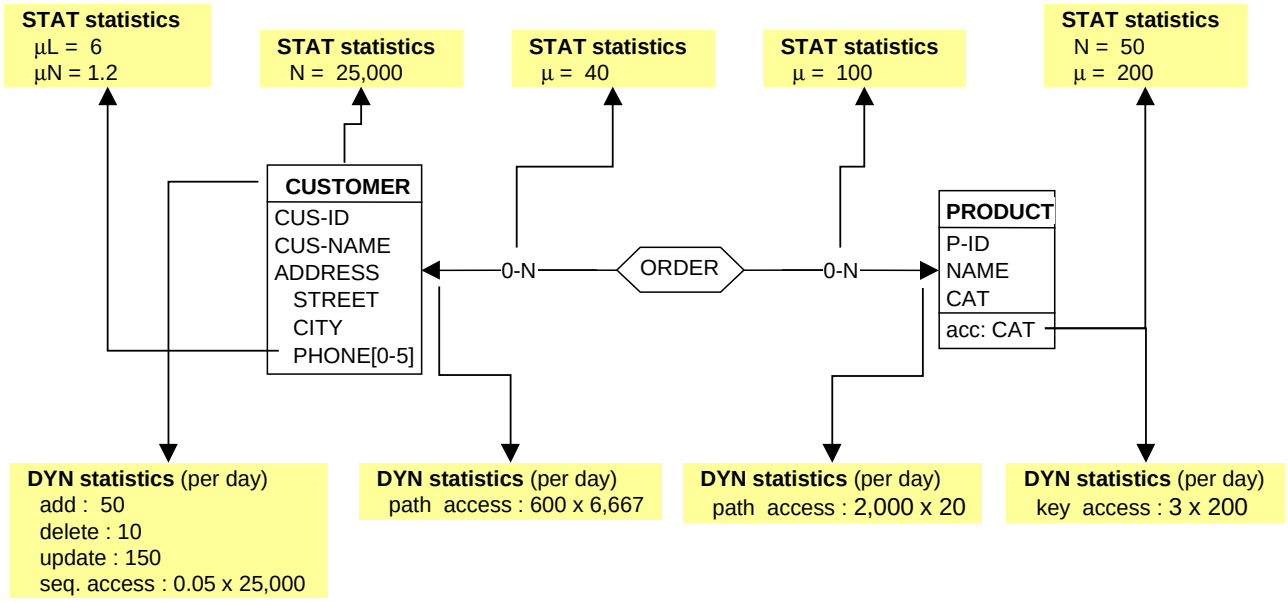

Figure 3 *- Illustration of statistical description of a TRAMIS schema*

This generic specification model 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. For example, we can call MERISE-like[4] a TRAMIS schema that satisfies, among others, the following rules :

- Entity types have at least one attribute;
- Attributes are single-valued, mandatory and atomic;
- Entity types have one and only one identifier;
- An entity identifier is made of one attribute;
- The cardinality constraints allowed are 0-1, 1-1, 0-N, 1-N;
- Relationship types have one and only one attribute;
- A relationship identifier is made of roles only.;
- There is no access keys, access paths nor spaces (i.e. the model is purely conceptual).

Similarly, an ORACLE-compliant schema is a TRAMIS schema in which,

- There are no relationship types;
- There is from one to 254 attributes per entity type;
- The attributes are single-valued and atomic;
- A group must be an access key (i.e. an index);
- An entity type cannot be in more than one space.
- A name is made up of 1 to 30 characters, the 1st being a letter, and the other ones being letters, figures, '_', '$' or '#';
- A name cannot belong to the ORACLE reserved-word list ('create', 'index', etc);
- The total length of the attributes that make a group cannot exceed 240 char.

In these rules, an entity type is to be interpreted as a *table*, an attribute as a *column*, a space as an ORACLE *space*.

The TRAMIS tool proposes about one hundred parametric rules that can be used to define submodels. Here are some examples of such rules (`min`, `max`, `fname`, `s1`, `s2` and `s3` are parameters) followed by their interpretation:

`1;min;max;`	an entity type has from `min` to `max` attributes
`2;min;max;`	an entity type has from `min` to `max` identifiers
`6;`	if an entity type has some access keys, one of them must be an identifier as well
`11;min;max;`	a relationship type has from `min` to `max` roles;
`15;`	the roles of a relationship type are taken by distinct entity types
`22;`	no relationship types are allowed
`35;fname;`	an object name cannot belong to the list that appear in the ASCII file `fname`

4 According to early definitions. More advanced MERISE proposals are less restrictive.

`36;s1;s2;s3;`	an object name must comply to the following syntax : the first character is in list `s1`, the last character is in list `s3` while the other ones are in list `s2`
`57;`	an access key cannot have multivalued components
`92;min;max;`	an access key or an identifier has from `min` to `max` components
`95;`	an access key cannot be the prefix of another key or identifier

A TRAMIS submodel is defined by a list of such rules. The CASE tool includes some built-in submodel definitions (standard E-R, relational, CODASYL, COBOL files, RDB, SYBASE, ORACLE, etc). In addition, users can define any number of specific submodels by building an ASCII file containing a list of rules in this format; these definitions are automatically integrated in the tool at run-time.

4. SCHEMA TRANSFORMATION

The concept of transformation is particularly attractive in the domain of databases, though it has not often been made explicit (for instance as a user function) in current CASE tools. A schema transformation is most generally considered as an operator by which a data structure S1 is replaced by another structure S2 which has some sort of equivalence with S1. Schema transformation is a ubiquitous concept in database design. Proving the equivalence of schemas [4], refining a conceptual schema [2], integrating two partial schemas [1] [7], producing a DBMS-compliant schema from a conceptual schema [6] [9], restructuring a physical schema [2], DB reverse engineering [2] [12] [15], are basic design activities that can be carried out by carefully chosen schema transformations .

Though developing this concept and its formalization is beyond the scope of this paper (see [11] for a more formal treatment), we shall sketch the main definitions and properties that will be important from the methodological viewpoint.

A **transformation T** is an operator that replaces a source construct C in schema S by another construct C'; C' is the target of C through T, and is noted **C' = T(C)**

More precisely, T is defined by,

- a precondition P that any construct C must satisfy in order to be transformed by T,
- a maximal postcondition Q that T(C) satisfies.

T can therefore be written T = <P,Q> as well. P and Q are pattern-matching predicates that identify the components and the properties of C and T(C).

In some cases, (i.e. when the graphical language is powerful enough), it is possible to give a more readable representation of a transformation by expressing C and T(C) graphically. We shall make use of graphical representation in the following.

A transformation T1 = <P,Q> is *reversible*[5] *iff* there exists a transformation T2 such that, for any construct C,

$$P(C) \Rightarrow T2(T1(C)) = C$$

T2 is the reverse of T1, and conversely. We have the following property : T2 = <Q,P>

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 possibly with a different presentation (syntax). Reversible transformations are also called *semantics-preserving*. The transformations provided by TRAMIS preserve not only the semantics of the source schema (i.e. the conceptual structures), but the other aspects of the specifications as well : access structures, names, statistical and informal information. For instance, the access structures (access keys and access paths) in source schema S1 are transformed into different, but functionally equivalent access structures in target schema S2; statistics in S1 are recomputed according to the new structures in S2; informal specifications of objects that disappear from S1 are transferred, possibly with some adaptations, to the new objects in S2. TRAMIS transformations are not only semantics-preserving but also *specification-preserving.*

TRAMIS offers a large set of transformations that can be used at any level of the design process. They are in no way goal-oriented (except for model-driven ones, as will be seen later), and can therefore be used in any design process and any strategy.

---

[5] In fact, the issue is a bit more complex, since a transformation must be defined not only by a mapping T between schemas but also by a mapping t between data populations (instances) of the schemas. Two kinds of semantics preservation can be defined, namely *reversibility* and *symmetrical reversibility* [7]. T1 is reversible iff, *for any instance s of schema S such that P(S), s = t2(t1(s))*; T1 is symmetrically reversible iff both T1 and T2 are reversible, i.e., in addition to the property mentioned above, *for any instance s' of S' such that Q(S'), s' = t1(t2(s'))*. For simplicity, we shall ignore this distinction in this paper.

Here follows a list of some **elementary transformations** proposed by TRAMIS. For conciseness, only conceptual structure transformations will be considered first and will be given a short explanation; the transformation of the other aspects will be evoked later. A formal description and the proof of the reversibility of the transformations that introduce/remove an entity type (Tr-E1, Tr-E2, Tr-R1, Tr-A1, Tr-A2) can be found in [11].

Transformation of entity types

- Tr-E1 : Decomposes an entity type E into two entity types E1 and E2, and distributes the attributes, the roles and the groups of E among E1 and E2.
- Tr-E2 : Integrates an entity type E2 into an entity type E1 through a one-to-one relationship type.

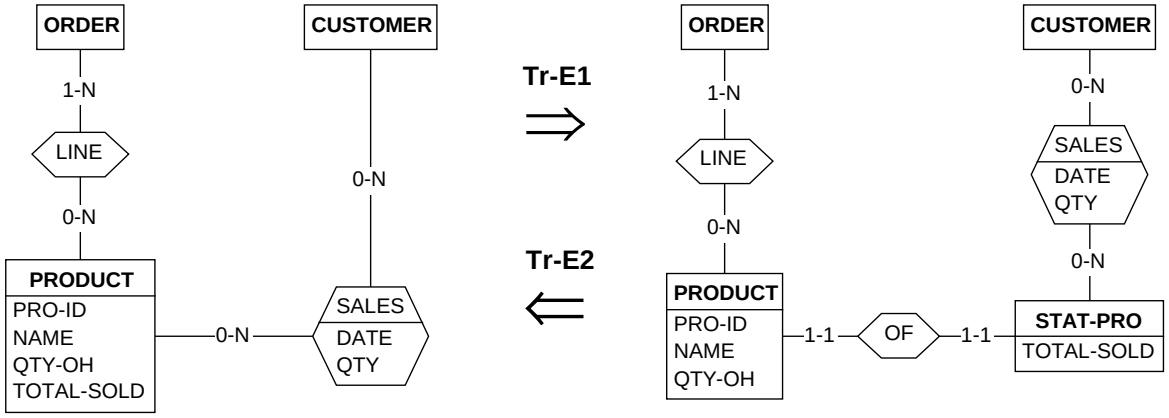

Figure 4 ***-*** *Decomposition of PRODUCT into two entity types by extracting one attribute and a role.*

Transformation of relationship types

- Tr-R1 : Replaces a relationship type R with an entity type and as many one-to-many relationship types as R had roles.
- Tr-R2 : Replaces a one-to-many relationship type R between E1 and E2 by attributes of E2 that are a copy of the identifying attributes of E1 + referential integrity constraint.

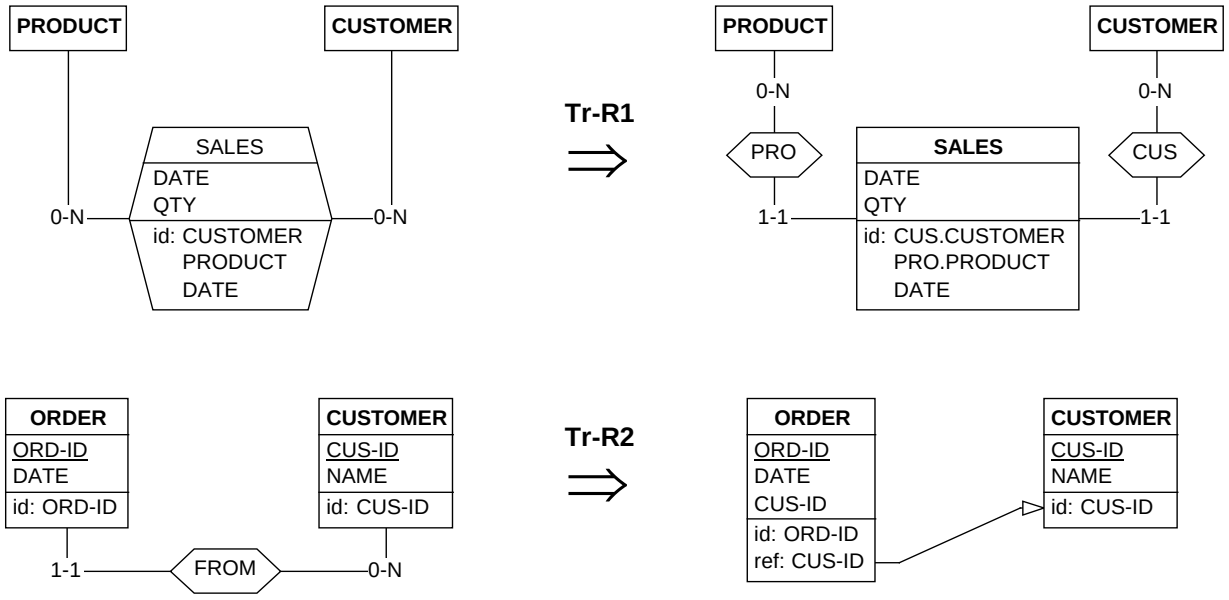

Figure 5 *- Transformation of relationship type SALES into an entity type and transformation of relationship type FROM into reference attributes (foreign key).*

Transformation of attributes

- Tr-A1 : Transforms an attribute A of entity type E into an entity type EA and a many-to-many relationship type RA between E and EA. Each EA entity represents a distinct value of A.
- Tr-A2 : Transforms an attribute A of entity type E into an entity type EA and a one-to-many relationship type RA between E and EA. Each EA entity represents an occurrence of an A value attached to an E entity.
- Tr-A3 : Transforms a multivalued attribute into a list of single-valued attributes.
- Tr-A4 : Transforms a multivalued attribute into one single-valued attribute.
- Tr-A5 : Aggregates a group of attributes into a compound attribute.
- Tr-A6 : Decomposes a compound attribute. Reverse of Tr-A5.
- Tr-A7 : Transforms a compound attribute into an atomic attribute.

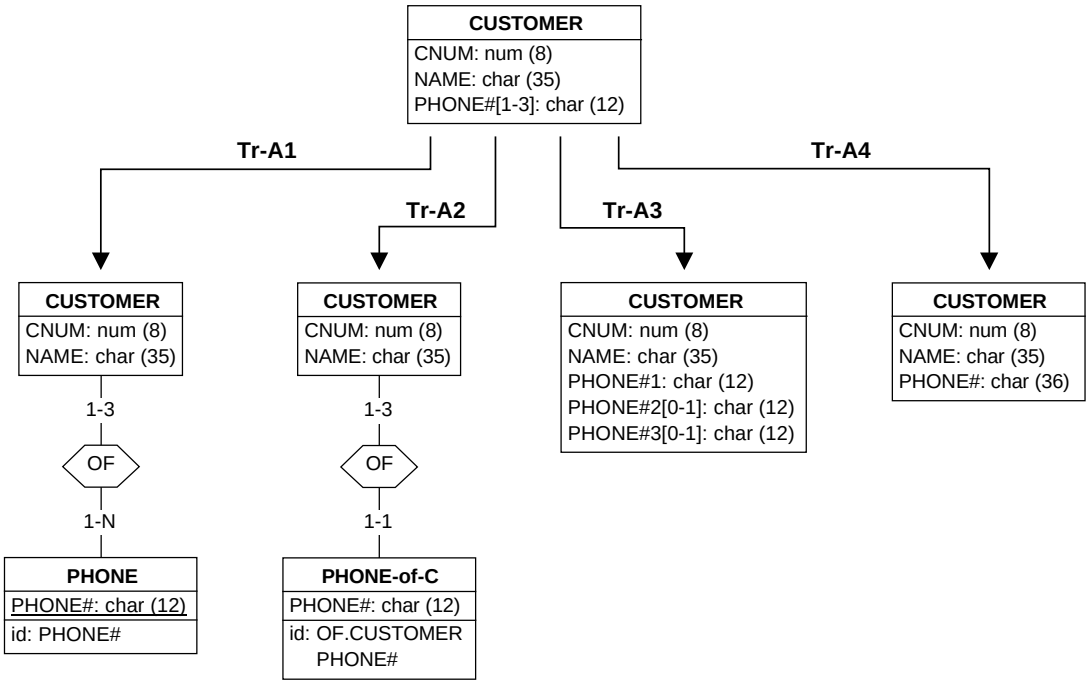

Figure 6 *- Four possible transformations to get rid of a multivalued attribute.*

Transformation of identifiers and access keys

- Tr-G1 : Separates two overlapping identifiers or access keys by duplicating common attributes (e.g. for COBOL data structures).
- Tr-G2 : Replaces a role component by the identifier of the entity type playing that role (e.g. for CODASYL schemas as in the schema below).
- Tr-G3 : Adds a singular relationship type role to a group (e.g. for CODASYL schemas).

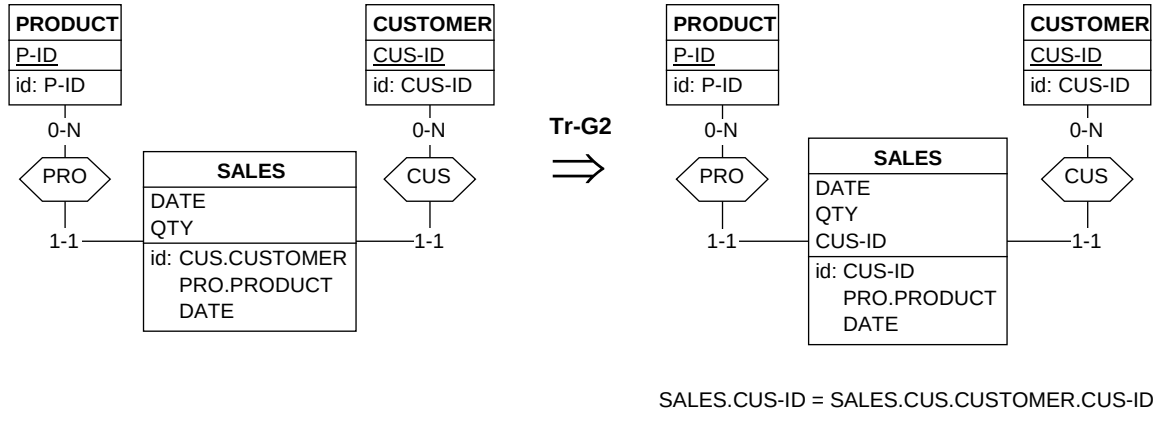

Figure 7 *- Replacing role CUSTOMER in the identier of SALES by CUS-ID, the identifier of CUSTOMER (the CODASYL data model doesn't accept more than one role in a* duplicates not allowed *constraint).*

Transformation of names

- Tr-N1 : Substring substitution.
- Tr-N2 : Name prefixing.

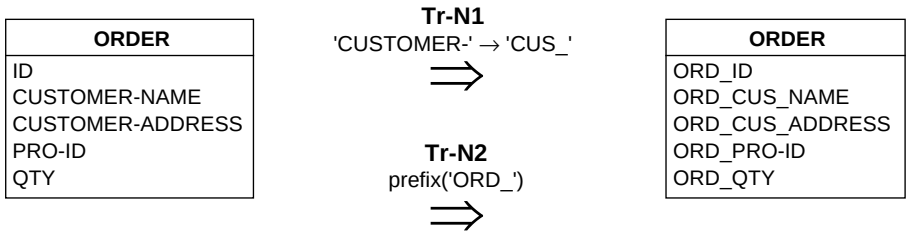

Figure 8 - *Translation and prefixing operations.*

Transformation of the non-conceptual aspects

As already specified, the transformations concern all aspects of the objects. For instance, transformation Tr-A1 processes the access structures (figure 9), as well as the statistical facets (figure 10).

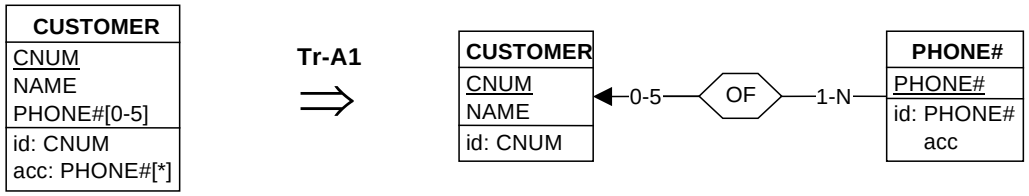

Figure 9 - *Translating an access key into an access path (e.g. in a CODASYL schema that does not accept multivalued access keys).*

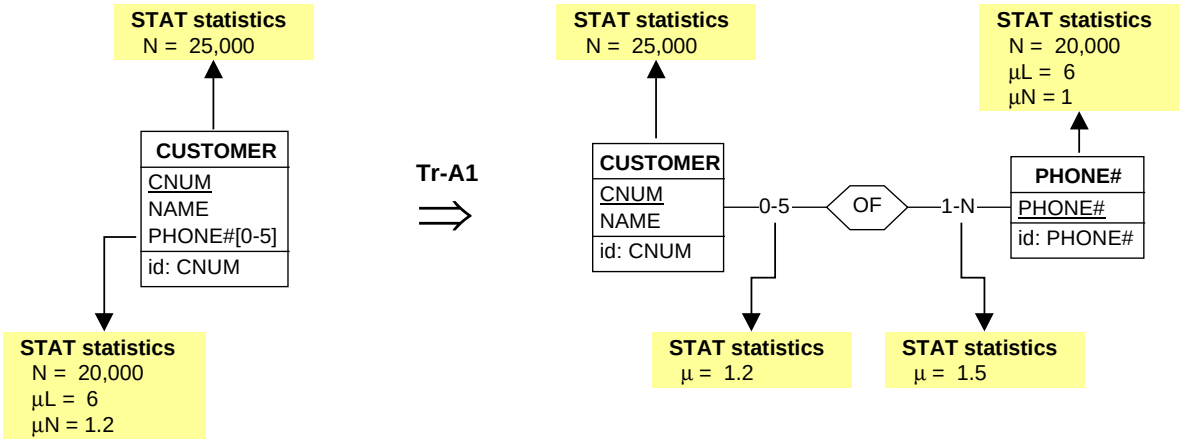

Figure 10 - *Conversion of static statistics in a structural transformation.*

Based on these operators, TRAMIS proposes a three-level transformation toolset[6] that can be used freely, according to the skill of the user, the desirable efficiency of the executable schema and the time allowed for producing that schema :

- **elementary transformations** : one transformation is applied to one object; with these tools, the user keeps full control on the schema transformation since similar situations can be solved by different transformations; e.g. a multivalued attribute can be transformed in four ways;

---

6 A schema transformation should not be confused with a design process, although the latter is said to *transform* a product into another one. Indeed, a design process that preserves the specifications need not be reversible (an unfortunate fact as far as reverse engineering is concerned). For example, adding an entity type to a schema transforms it into a richer schema that preserves the previous specifications, while the reverse process (deleting the entity type) doesn't preserve the specifications that are in the second schema.

Formally : given construct C in schema S and transformation T, both selected by the user, C is replaced by T(C) if $P_T(C)$.

- **global transformations** : one transformation is applied to all the relevant objects of a schema.
  Formally : given transformation T and a class of constructs CC, both selected by the user, for each C in CC such that $P_T(C)$, replace C by T(C). Examples : replace all one-to-many relationship types by foreign keys + referential constraints; replace all multivalued attributes by entity types + many-to-one relationship types.
- **model-driven transformations** : all the constructs of a schema that do not comply with the rules a given submodel are transformed; these transformations require virtually no control from the user; the resulting schema is correct, it complies with, say, the relational or CODASYL model, but has few refinements as far as efficiency is concerned[7]. The resulting schema is said to be relational-, CODASYL-, etc, compliant, and its expression into the corresponding DDL is quite straighforward.
  As an example, the following strategy is used to produce an SQL-compliant schema :

```
1. for each non-functional[8] relationship type R, do :
       apply Tr-R1 to R;
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 :
           apply Tr-A6 to A;
       for each attribute A that is multivalued, and that depends directly on an
   entity type, do :
           apply Tr-A2 to A;
3. until no relationship type can be transformed, do :
       for each (functional) relationship type R, do :
           if Tr-R2 is applicable, apply Tr-R2 to R
           otherwise skip R;
4. until no relationship types remain, do :
       for each relationship 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;
```

TRAMIS includes strategies for transforming an E-R schema into SQL, CODASYL-73 and COBOL files schemas. It is worth noticing that transformations of the three levels can be used whatever the state of the schema. For instance, a model-driven transformation can be used to quickly produce an SQL schema from a pure conceptual schema (by a novice designer or for fast prototyping). It can also be used to give the finishing touch to a schema that has already been optimized and partly made SQL-compliant through elementary and global transformations (by an expert designer).

5. ARCHITECTURE OF TRAMIS

The architecture of TRAMIS is based on two main features, namely a repository and a toolbox structure.

The TRAMIS repository, or specification database, contains the current state of the schema under development. It is implemented with MDBS-3 (from MDBS), a high-performance and compact CODASYL-like DBMS. An access module has been built in order to give the functions an entity-relationship interface, providing the tool with both ease of development and DBMS independence. The repository is itself a medium-complexity database since its schema comprises about 30 entity types. The functions of the tool work directly on the stored data, a performance that would not have been realistic on PC workstations with, say, a relational DBMS.

The toolbox structure gives users the maximum freedom as far as their design behaviour is concerned. Indeed, incremental, trial&error, prototyping, activities can be carried out easily. TRAMIS provides a set of tools that can be used freely. The tools are independent from each other and communicate through the repository only. Both skilled and novice (or hurried) designers can find their way easily : some users need fine-tuning tools on which they have full control, while other users need simple and automated tools for quick production of executable descriptions.

---

7 This is the only schema production function that is provided by many current database CASE tools.

8 A *functional* relationship type is binary, one-to-many or one-to-one, and has no attributes.

There are two kinds of tools. The kernel comprises basic tools that offer elementary functions to manage and transform schemas. The other tools are intrinsically process-oriented and are built on basic tools. Model-driven transformations are examples of such tools.

The tools are grouped into toolsets that are accessed through a common dialog manager providing a WIMP-based user-interface.

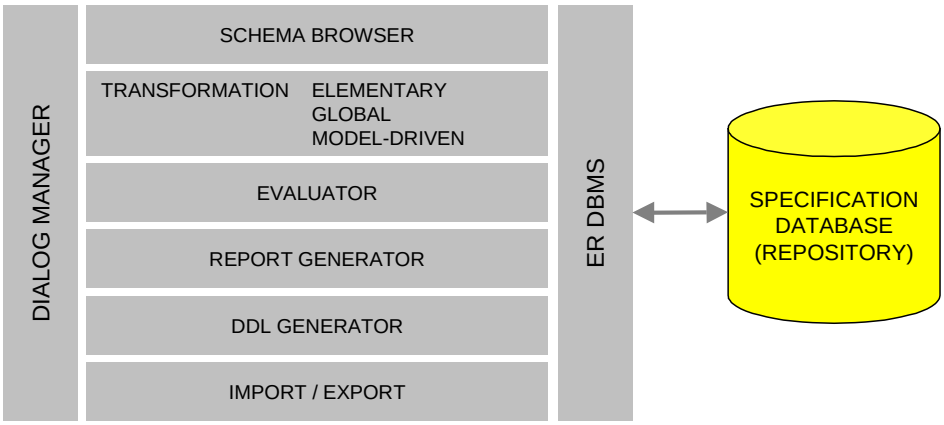

Figure 11 - *Gross architecture of the TRAMIS CASE tool.*

SCHEMA BROWSER: allows the browsing of the current state of the specification database, object selection by name or in a list of objects and the navigation from object to associated objects. Allows the consultation and updating of the different aspects of the current object, together with the creation and deletion of objects.

SCHEMA TRANSFORMATION : allows the application of one elementary transformation on the current object, the application of one transformation on all the objects of the schema, and the application of all the necessary transformations on the objects of the schema in order to transform it into a new schema that complies with a selected submodel. All the facets of the transformed objects are recomputed in order to preserve their information contents (a typical elementary transformation requires from 10 to more than 100 elementary updates, a performance that cannot be done by hand).

EVALUATOR: at any moment, the user can validate its specifications against a selected submodel. TRAMIS compiles the rules defining the submodel and analyzes the current specifications according to these rules. TRAMIS generates diagnostics that will be examined by the user. A submodel is defined in an ASCII text consisting of a list of statements expressed in the rule language mentioned in 3. The user can build any number of submodels. TRAMIS includes several predefined submodel, either general-purpose (standard E/R, relational, CODASYL, COBOL files), or specific DBMS-oriented (RDB, SYBASE, ORACLE, etc).

REPORT GENERATION : production of a detailed report , for all objects or for one object, a statistical report (with volumes and operations), a dictionary report (concise structured list of object names with *natural/technical* names translation or conversely).

DDL GENERATION : translation of a TRAMIS schema into executable descriptions according to a selected DBMS. The generation is twofold : production of a **DDL text** and generation of an **additional document** that reports all the integrity constraints that have not been translated in the DDL text together with validation procedures for these constraints. This report, aimed at the programmer and the DBA, includes the technical notes as well. TRAMIS includes generators for the following data(base) managers : CODASYL, RDB, SYBASE, ORACLE, standard COBOL files, etc.

IMPORT / EXPORT : TRAMIS has been given an external specification language (Information System Specification Language, or ISL) in which any specification can be expressed. This language allows communications between tools, selective backup or even data input. Through the *export function*, a selected part (what objects and what facets) of the specification database contents can be generated in ISL. Through the import function, the contents of an ISL text can be integrated to the specification database. This function allows an incremental integration of different schemata.

TRAMIS has been implemented in C in the MS-Windows environment.

6. A DESIGN SCENARIO

Figure 12 presents the standard three-level database design method together with suggested design processes for each of the three phases. This design scenario is derived from the material that can be found in [9] and [2]. The right-hand side column mentions the TRAMIS tools that can be used to carry out these processes.

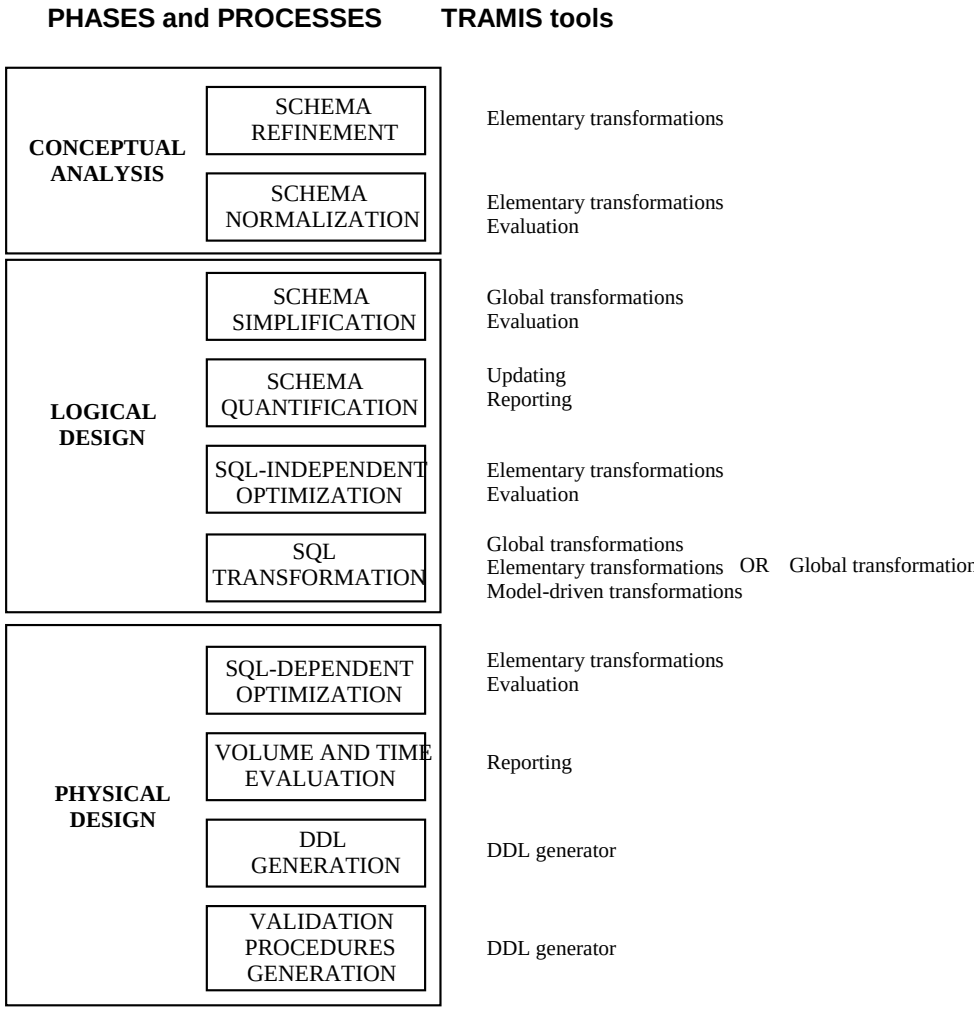

Figure 12 *- An example of design processes and the corresponding TRAMIS tools to use.*

7. CONCLUSION

Due to its rich set of explicit transformational operators, to its unique generic model, and to its toolset architecture, TRAMIS allows very different database design approaches to be practiced. Both so-called formal, rigid, methods and heuristics behaviours can be followed. However, the potential of these basic paradigms are not fully exploited yet in this first version. Therefore, a new version is under development.

A large kernel of the next version of TRAMIS is already operational. Though it includes and extends the principles that have been described in this paper, it differs significantly from the first version as follows.

- The repository is no longer a database, but a persistent object base (written in C++) the main classes of which describe, among others, *schemas*, *entity types*, *relationship types* and *roles*, *attributes* and *domains*, *constraints*, *access mechanisms*, *processes*. The methods include navigation and management functions, but also transformation and evaluation operators.
- The TRAMIS model has been extended to accomodate, for instance, *is-a* relations and additional integrity constraints.
- The set of transformations has been extended to allow more powerful schema restructuring, for instance for optimization and reverse engineering purposes.

Other inovative functions are currently under development. We will mention four of them :

- The statistical model is being revised in order to make it more intuitive and more powerful. In particular, the specification of time-dependent statistics will allow to simulate the evolution of volume and time performance [14].
- Schema integration functions have already been studied and will be included. They provide structural conflict resolution [1] [7].
- The very concept of design method, as well as that of design behaviour, are being studied in order to model them. This model, and its associated language, will allow TRAMIS to control and enforce any design method (the first version was passive with this respect).
- TRAMIS will be extended to allow the evolution of databases. Following any modification of a schema (whatever its abstraction level), it must be possible to automatically *replay* all the design processes that are still relevant, and to assist the designer in processing the new aspects. In addition, data conversion procedures must be generated automatically. This function is closely linked to design process modelling.

8. GENERAL REFERENCES

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

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

[3] Bouzeghoub, M., *SECSI : Un système expert en conception de systèmes d'information*, Thèse de doctorat, Université Paris VI, Mars 1986

[4] 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

[5] Lloyd-Williams, M., Beynon-Davis, P., *Knowledge-based CASE Tools for Database Design*, in *CASE - Current Practice, Future Prospects*, Spurr, K., Layzell, P. (Ed.), Wiley, 1992

[6] 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

[7] Spaccapietra, S., Parent, Ch., *View integration : a step forward in solving structural conflicts*, IEEE Transactions on Knowledge and Data Engineering, October, 1992

[8] Tardieu, H., Rochfeld, A., Coletti, *La méthode Merise*, Les Editions d'Organisation, 1984

9. SPECIFIC REFERENCES

[9] Hainaut, J-L., *Conception assistée des applications informatiques - 2. Conception de la base de données*, Masson, 1986

[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 Conf. on Entity-Relationship Approach, San Mateo, 1991, North-Holland, 1992

[12] Hainaut, J-L., *Database Reverse Engineering, Models, Techniques and Strategies*, in preProc. of the 10th Conf. on Entity-Relationship Approach, San Mateo, 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., *A Temporal Statistical Model for Entity-Relationship Schemas*, in Proc. of the 11th Conf. on the Entity-Relationship Approach, Karlsruhe, Oct. 1992, Springer-Verlag, LNCS, 1992

[15] Joris, M., Van Hoe, R., Hainaut, J-L., and al., *PHENIX : Methods and Tools for Reverse Engineering*, in Proc. of the 5th Int. conf. on Software Engineering and its Applications, Toulouse, December 1992