DATABASE DESIGN TOOLS: AN EXPERT SYSTEM APPROACH

Mokrane BOUZEGHOUB, Georges GARDARIN
Elisabeth METAIS

Projet SABRE
Laboratoire MASI, Institut de Programmation
Université Paris VI, 4, place Jussieu 75230
and
INRIA, BP.105 78153 Le Chesnay Cedex France

ABSTRACT: In this paper, we report on the implementation of SECSI, an expert system for database design written in Prolog. Starting from an application description given with either a subset of the natural language, or a formal language, or a graphical interface, the system generates a specific semantic network portraying the application. Then, using a set of design rules, it completes and simplifies the semantic network up to reach flat normalized relations. All the design is interactively done with the end-user. The system is evolutive in the sense that it also offers an interactive interface which allows the database design expert to modify or add design rules.

1. INTRODUCTION

Today, relational technology is widely spread. Many users are designing their databases with the relational model. However, using the relationnal model as a conceptual design tool is somewhat controversial [KENT79]. Indeed, the relational concepts of domain, attribute, relation, referential constraint, functional and multivalued dependencies are neither simple to use nor sufficient to capture the semantics of the user's applications. To enhance the semantics, integrity constraints may be used, but their expression is not always easy nor natural for the end-user.

To capture the semantics of the real world with more preciseness and naturalness, many researchers have proposed several so-called semantic data models, such as SHM [SMIT77], SHM+ [BROD81], RM/T [CODD79], SDM [HAMM80], TAXIS [MYLO81], LAURA [BROW83], MORSE [BOUZ83a]. Objects are generally assembled together using some kind of constructs borrowed from semantic networks used in Artificial Intelligence. Except some differences in the formalization and the way of expressing certain constraints, these models offer similar concepts of object, classification, aggregation,association, and generalization.

Using semantic data models is far from being sufficient to make the design process easy for large applications. Indeed the design process is an iterative, long and tedious task. It is characterized by a certain indetermination in the way of choosing data stuctures and constraints. Several different schemas may describe the same reality. The design process is also characterized by an intuitive and empirical methodology. Consequently, the quality of the schema obtained is heavily dependent on the database administrator's experience and insight in the database design.

Even if a methodology produces a "good" conceptual schema, it is not trivial at all to translate it into a physical database schema. The conceptual to physical schema mapping is dependent from both the user application (e.g. the transactions) and the database system (relational,network). Consequently, an accurate and efficient internal schema is difficult to produce from a good conceptual schema without automatic tools and human interactions.

Several design tools have already been proposed [CERI83, DAVI83, WASS82, COBB84, TAHN84, DAEN84] for database design. Some of them are attractive and original, but most of them suffer from the following shortcomings:
(1) They are not completly integrated; in other words, they do not constitute a complete system but a set of sparse programs which rarely interface each other.
(2) They are not evolutive; some change in the design rules often implies reprogramming the whole tool.

At the beginning of 1982, starting from this analysis of the database design state of the art, we proposed a new approach based on expert systems techniques [BOUZ83b]. This approach is supported by an original tool called SECSI (an acronym for Système Expert en Conception de Systèmes d'Informations) which has been implemented in PROLOG, on top of SABRE, a relational database management system. The system is strongly based on a semantic data model [BOUZ83a], the relational technology and certain artificial intelligence techniques as expert systems.

An expert system is an intelligent program which is devoted to a specific domain of application and where there exists enough knowledge to infer one or several solutions, but where there does not exist any precise or performant algorithm which performs the same results. This approach is chatacterized by an original architecture which distinguishes between :
- a knowledge base which contains concepts, facts, rules and skills,
- an inference engine which is a set of management techniques of the knowledge base,
- and a friendly external interface by which the end-user interacts with the system.
The power of an expert system is characterized by the content of its knowledge base and its capabilities to work as efficiently as possible like a human expert [LAUR81, HAYE83].

SECSI is intended to have the same characteristics of the expert systems, but it is not designed to replace the human expert. Its knowledge base is organized as modules of rules; a set of modules compose an abstract level of knowledge; going down the levels offers gradual refinement of knowledge. Thus the knowledge base is specified in such a way that it is easy to integrate new design rules and to update the existing ones as soon as the knowledge progresses. SECSI is not designed as a black box providing useful services but as an open system which is able to explain and to transfer its expertise to the end-user and to learn new knowledge.

The purpose of this paper is to present the architecture and the implementation of SECSI. This paper is organized as follows. In section 2, we introduce the objectives and the architecture of SECSI. In section 3, we present the various external interfaces of SECSI. The section 4 is devoted to the internal representation of knowledge which is based on a specific semantic network and production rules. They are both represented by Prolog clauses. In section 5, we detail the logical design process which is currently implemented.

2. OBJECTIVES AND ARCHITECTURE OF SECSI

2.1. OBJECTIVES

SECSI has been designed as an integrated intelligent tool for helping the user in the tedious process of database design. The design of SECSI was directed by the following specific objectives :
(1) To constitute a knowledge base composed of all useful concepts and algorithms developped in the relational theory and in the semantic data models area. This will be especially helpful for common designer who are not necessarily expert in database design theory. This knowledge base may also include some experimental and specific rules related to the user's experience in database design and to a specific domain of application (banking, reservation, medicine...).
(2) To define an interactive methodological environment which permits to perform as far as possible the design steps with incomplete specifications, and which permits to backtrack to any step in order to change some specifications or to integrate new information.
(3) To identify for each design step the general or specific principles of reasonning, and to provide as detailed explanations as possible about these principles, the models, and the rules on which they are based
(4) To build an open system of tools which enables in one hand to integrate

new theoretical concepts and design rules, and in the other hand to transfert its expertise both via its usual use and via explanations and justifications of its results.
(5) To facilitate interaction with the human designer by offering him a semantically rich and easy to use interface.

This tool is qualified as an expert system in the sens that:
- it offers an evolutive knowledge base,
- it accepts incomplete specifications,
- it justifies and explains its results,
- it permits to backtrack to any design step in order to change specifications or to ask for explanations.

Of course, all these objectives are far from being thoroughly reached with the current implementation. However, the architecture of SECSI allows us to pursue them.

2.2. SYSTEM ARCHITECTURE

The general architecture of SECSI is portayed in figure 1. Like most expert systems [LAUR82,HAYE84], SECSI is organised around an expert knowledge base composed of a set of design rules (BR) and a set of facts (BF). The set of rules captures the design methodology while the set of facts describes the user's application. In the current

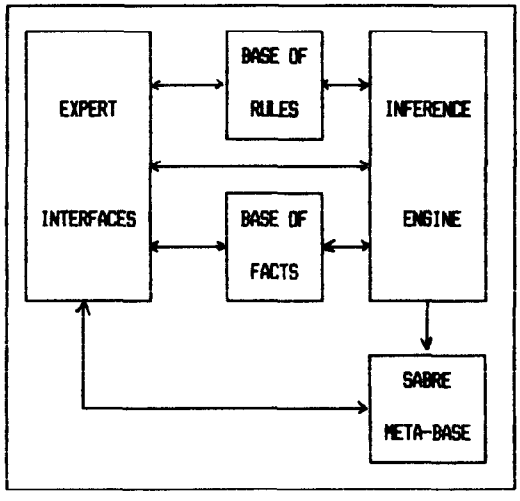

Fig.1: General architecture of SECSI.

version of SECSI, Prolog acts as the inference engine of the system. Using the set of design rules, this inference engine carries out the deduction process. It first generates a normalized relational schema composed of a set of permanent relations with their keys, a set of virtual relations derived from the formers by given queries, and a set of integrity constraints. The integrity constraints include domains, referential and inclusion constraints. It should be extended to more general constraints in the near future.

A SECSI session is organized in steps. Whenever a step is activated, the system may ask for complementary information and the end-user may ask for explanations. Whenever a schema has been generated by SECSI, the user can desagree with the result. In this case, the session may be restarted at any of the design steps according to the user request. As soon as the schema satisfies the user's needs, the design process is terminated and the schema is stored in the Sabre meta-base.

The external interfaces address two different experts : the expert in database design (shortly refered here as the expert) and the specialist in the specification of applications (shortly refered here as the end-user). The expert is responsible of the creation and the modification of the base of design rules. The end-user is in charge of the creation and the modification of the base of facts describing the application. In the following, we explain in more details the various interfaces and the corresponding process of SECSI (Figure 2).

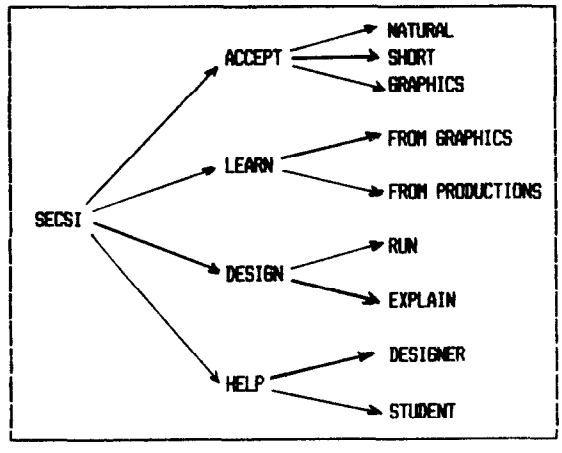

Fig.2: Interfaces and processes of SECSI

SECSI offers the LEARN function to the expert and the ACCEPT, RUN and HELP functions to the end-user (see figure 2). LEARN enables the expert to

introduce and update the design rules. Such rules are introduced by using either graphical interface or production rules. In the first version they are directly written in Prolog. ACCEPT enables the end-user to introduce, to list and to update the description of an application. Three languages are offered to the end-user by the ACCEPT process : a restricted natural language (ACCEPT-NATURAL), a simple declarative language (ACCEPT-SHORT) and a graphical interface (ACCEPT-GRAPHICS). The DESIGN process yields a normalized relational schema from an application description (RUN) and brings out explanations about the produced schema and the applied rules (EXPLAIN). HELP informs and assists the end-user about the model used, the applied design rules and the functioning of the system itself (HELP-DESIGNER). Also, we plan that this module may help students learning data models and database design (HELP-STUDENT) from predefined rules and examples.

2.3. THE LIMITS OF THE SYSTEM

The database design methodology may be seen as three complementary phases:
(1) view specification and integration
(2) logical schema design and
(3) physical schema design.
The first version of SECSI which is described in this paper is only concerned by the second phase (i.e. logical design) including some aid in schema specification and consistency verification. The objective of this first version is to learn expert systems and to show through one design phase how do they apply to database design. We are specifying a second and a third version for view integration and physical design.

Currently, the system only helps in the design of the data structure and integrity rules, that is the passive components of an information system. It is of no help for the transaction design phase. However, SECSI does not ignore the influence of potential transactions both on the conceptual and the physical structure of the database. Indeed, the physical design process should integrate information about transaction frequencies, volumes and required level of response time for the main transactions.

3. END-USER AND EXPERT INTERFACES

3.1. HOW TO DESCRIBE AN APPLICATION

To specify the data structures of an application, the end-user may choose between three types of interfaces: a simple but formal declarative language, a restricted subset of the natural language and a graphical interface. He may also use two or all of them.

The declarative language is derived both from programming language type declarations and from functional language constructs (see for example DAPLEX [SHIP81]). It is defined by a very simple grammar which is illustrated by the example given figure 3. This grammar permits to declare IS-A relationships: STUDENT : PERSON,
n-ary associations between entities:
ENROLLED(STUDENT,COURSE),
including hierarchies as in the network and in the hierarchical model
EMPLOYEE(DEPARTMENT),
attributes that characterise entities with their basic types:
NAME(PERSON): TEXT,
and some constraints as functional and multivalued dependencies:
NAME(DEPARTMENT) -> ADDRESS(DEPARTMENT).

```
EMPLOYEE : PERSON.                    NAME(PERSON) : TEXT.
STUDENT : PERSON.                     ADDRESS(DEPARTMENT) : TEXT.
STAFF : EMPLOYEE.                     NAME(DEPARTMENT) = (MATH,DB).
TEACHER : EMPLOYEE.                   SSN(EMPLOYEE) : INTEGER.
INSTRUCTOR : TEACHER.                 SALARY(EMPLOYEE) : REAL.
PROFESSOR : TEACHER.                  TEL(TEACHER) : INTEGER.
HEAD-OF-SECTION : STAFF.              ADDRESS(TEACHER) : TEXT.
DIR-OF-LABORAT : STAFF, TEACHER.      FREE-GIFT(STAFF) : REAL.
                                      NUMBER(STUDENT) : INTEGER.
                                      DATE(ENROLLED) : INTEGER.
RESPONSIBLE(PROFESSOR,COURSE).        NUMBER(CLASS) : INTEGER.
GIVEN_BY(CLASS,INSTRUCTOR).           NAME(COURSE) : TEXT.
ENROLLED(STUDENT,COURSE).             DAY(COURSE) : TEXT.
EMPLOYEE(DEPARTMENT).                 HOUR(COURSE) : INTEGER.
CLASS(COURSE).                        ROOM(COURSE) : INTEGER.

NAME(DEPARTMENT) -> ADDRESS(DEPARTMENT).
SSN(EMPLOYEE) -> NAME(EMPLOYEE), SALARY(EMPLOYEE).
NAME(COURSE) -> ROOM(COURSE), DAY(COURSE), HOUR(COURSE).
```

Fig.3: Example of the declarative language description.

The natural language based interface offers a very restricted subset of French which makes the specification readable and easily communicable. In fact, it appeared quickly not feasible to start with very complex specifications in French. One reason is that natural language understanding is very complex and constitutes a vast

domain of research in itself; another is that an information system designer is an expert who has his own jargon and who needs synthetic and unambiguous powerful tools instead of subject-verb-complement sentences. Hence, we limited our natural language interface to a strict translation in a more natural form of our declarative language. An example corresponding to a sample of the previous declarative language interface is given in figure 4.

```
EMPLOYEES AND STUDENTS ARE PERSONS.
STAFF_MEMBERS AND TEACHERS ARE EMPLOYEES.
INSTRUCTORS AND PROFESSORS ARE TEACHERS.

DEPARTMENT ADDRESS AND NAMES ARE TEXTS.
EMPLOYEE'SSN IS AN INTEGER.
EMPLOYEE'SALARY IS A REAL.

A PROFESSOR IS RESPONSIBLE OF A COURSE.
A CLASS IS GIVEN BY AN INSTRUCTOR.
A STUDENT IS ENROLLED IN ONE OR MORE COURSES.

A DEPARTMENT NAME DETERMINES A DEPARTMENT ADDRESS.
AN EMPLOYEE'SSN DETERMINES AN EMPLOYEE'S NAME.
AN EMPLOYEE'SSN DETERMINES A SALARY.
```

Fig.4: An example of description in natural language.

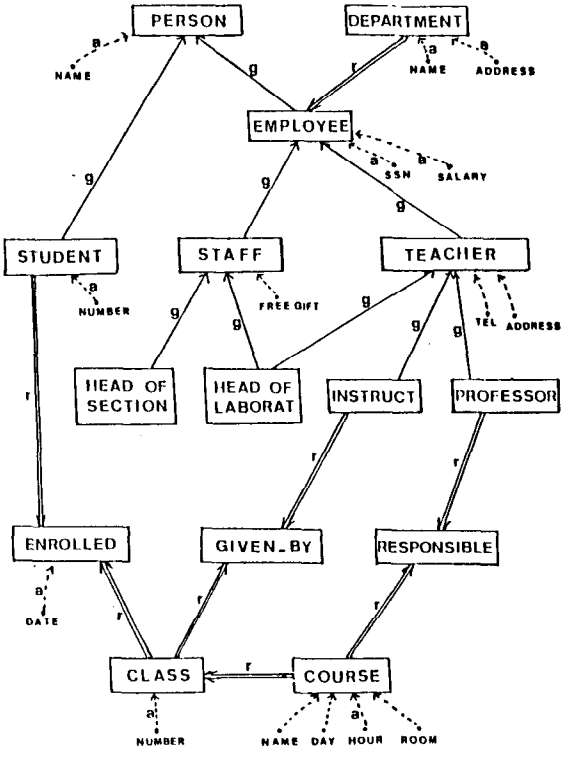

Fig.5: An example of the graphical interface.

The graphical interface may be either a direct implementation of the semantic network we utilize to represent the internal knowledge, or one of the traditional data models such as the Entity Relationship model and the Codasyl network data model. An example of a semantic network corresponding to the previous example in figure3 is given in figure 5. The semantics of the different arcs will be explaned in secstion 4.

3.2. HOW TO SPECIFY DESIGN RULES

As for the end-user, the experts need powerful languages to specify their expertise in database design. We envision to offer the experts two types of interfaces : a declarative language based on if-then statements and a graphical tool for expressing mappings between two types of semantic network.

The declarative language should accept statements of the form:
IF CONDITION THEN ACTION.
The condition expresses a relationship between two objects. Possible relationships are:
- AGGREGATION OF / ATTRIBUTE OF,
- CLASS OF / INSTANCE OF,
- GENERALIZATION OF / SPECIALISATION OF,
- ASSOCIATION OF / PARTNER OF,
- EQUIVALENT TO.

For example, suppose we have to specify as a design rule the inheritance property in the generalization hierarchies; it can be written as the production rule portrayed in figure 6.

```
IF X IS A GENERALIZATION OF X1
   AND A IS AN ATTRIBUTE OF X
   AND X IS A PARTNER OF R
THEN A IS AN ATTRIBUTE OF X1
   AND X1 IS A PARTNER OF R.
```

Fig.6: An example of a rule expressed in a declarative form.

The graphical interface is a very convenient tool to express mapping rules between two types of semantic network. As the conceptual modeling is often a question of schema representation and schema mapping, this latter facility is very important. We shall see later that most of the design rules are mapping rules, thus having facilities to visualize these rules will probably increase the friendliness of the system. Generating rules from examples may also be an attractive issue. However, many

rules cannot be expressed by graph transformations; in this case production rules should be used.

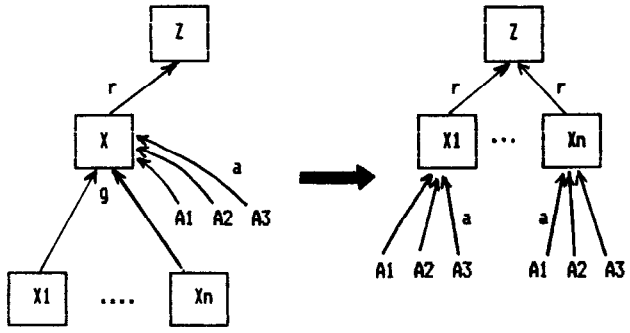

Fig.7: An example of rule expressed as a graph transformation.

The two preceding interfaces will be compiled, and processible rules will be generated. In the first version of SECSI which is currently running, these two interfaces are not yet implemented; rules are directly represented as Prolog clauses.

4. INTERNAL REPRESENTATION OF KNOWLEDGE

As stated before, we have two types of knowledge: facts and rules. To represent this knowledge, we use a combination of two models: semantic networks to represent facts and production rules to represent application constraints and design rules. The following sub-sections deal with these two kind of models.

4.1. INTERNAL REPRESENTATION OF FACTS

To implement the base of facts, we use a specific kind of semantic network because of the privileged position of this tool between database models and natural languages. Our semantic network presented hereafter contains most of the concepts of semantic data models like aggregation, generalization and classification. The main differences with these models are, first, the formalization with a few basic constructs (a, r, c, g); second, the categorization of the different nodes and arcs and the distinction between two types of aggregation (aggregation of attributes called aggregation, and aggregation of entities called association). Moreover, several constraints may be added on each kind of arcs and nodes.

We are now going to present a more formal definition of this semantic data model. Our semantic network is defined as a triple (NC,AC,IC) where NC stands for the category of nodes, AC the category of arcs and IC the category of constraints, such that for each element f of AC, there exists an application :

f: NC X NC -----> {True,False}

such that f(ni,nj) is true if there exists an arc of class f between ni and nj, and false otherwise. The elements of NC can be classified in two ways :

(1) Atomic objects (attributes and values) and molecular objects (entities and instances).

(2) Classes (attributes or entities) and elements of classes (values or instances).

The elements of these different categories of nodes are connected by the following categories of arcs:

- Aggregation arc denoted a(X,Y) specifies that X is a part of Y or that Y has the property X. This arc links an atomic object to a molecular object. For example, using the application portrayed figure 5, we can write a(NAME,PERSON), a(ADDRESS,PERSON).

- Association arc denoted r(Y,Z) specifies that Y is involved in the association Z. An association connects molecular objects (entities or instances). For example, the binary relationship ENROLLED(STUDENT,COURSE) may be written as:

  r(STUDENT,ENROLLED),
  r(COURSE,ENROLLED).

- Classification arc denoted c(X,Y) specifies that X is an element of the class Y. Classification are not recursive and can only link a value to its attribute class or an instance to its entity-class. For example, we have

  c(PARIS,ADDRESS),
  c(<COMPUT-SCE PARIS>,DEPARTMENT)

where <COMPUT.SC. PARIS> is a tuple representing an instance of DEPARTMENT.

- Generalisation arc denoted g(Y,Z) specifies that Y is a sub-class of Z. It corresponds to the well-known is-a relationship. It may be used recursively in a hierarchy of objects and has the transitivity property. For example, we have in the given application, g(STUDENT,PERSON) and g(PROF,TEACHER).

- Equivalence arc denoted e(Z1,Z2) specifies that two nodes are equivalent. This arc is especially useful when it is

important to see the same object in different ways. For example, e(STUDENT,PUPIL), e(STUDENT,SPORTSMAN) specify that STUDENT, SPORTSMAN and PUPIL are equivalent classes if we assume that all students practice one sport. More generally, e(X,Y) is equivalent to the two following assertions g(X,Y) and g(Y,X).

The previous arcs can be interpreted in the reverse direction respectively as particularization (p), partnership (o), instantiation (i), specialization (s) and equivalence (e) arcs.

Some constraints have to be added to these arcs and nodes to enhance the semantics of the preceding network. Most of them may be expressed by additional nodes and/or arcs, or by appropriate expressions of predicates. We list hereafter some types of these constraints :

- Domain constraint: Each attribute has a domain which is extensionally defined by enumerating its values, or intensionally defined as a basic data type (integer, real, or text). Moreover, data type values can be constrained by any predicate.

- Intersection constraint: There is an intersection between two classes X1 and X2 when it exists a third class X3 such that the predicates g(X3,X1) and g(X3,X2) hold. For example, with the university application, the two classes STUDENT and INSTRUCTOR intersect because
g(STUD-INSTR,STUDENT)
and g(STUD-INSTR,INSTRUCTOR).

- Union constraint: expressed with respect to a generalization hierarchy. It specifies whether the union of all specialization classes is equal or not to the root class of the hierarchy. Let X1,..Xn be the subclasses of X and let I, Ii be repectivelly the elements of X and Xi, then if $\bigcup_{i}^{n}$ Ii = I, X is called a completely specialized class, otherwise X is called a partially specialized class.

- Cardinality constraint: This constraint is associated with r arcs and a/p arcs (keep in mind that p is the reverse of a). Cardinalities are represented by a pair of values (m,n) which specifies on the one hand whether the relationship is total (m>0) or partial (m=0), and on the other hand whether the relationship is functional (n=1) or not (n>1). For example, if r(STUDENT,ENROLLED) has the cardinality (1,4), this means that a student has at least one enrollment and at most 4 enrollments. If p(TEACHER,TEL) has the cardinality (0,2) then it means that a teacher may have zero, one or two telephone numbers. In general, the relevant values are 0 or 1 for m and 1 or N for n (with N>1).

- Functional dependency constraint: We consider here functional dependencies between attributes of the universal relation schema composed of all the attributes of a semantic network. As in our semantic network we do not assume the uniqueness of attribute names, we qualify each attribute by the name of its entity. For example, a possible functional dependency is :
NAME(DEPARTMENT)-->ADDRESS(DEPARTMENT).

4.2. INTERNAL REPRESENTATION OF RULES

Three important classes of design rules have been distinguished.

The first class includes consistency enforcement rules and structural transformation rules which act upon the semantic network. Consistency enforcement rules enable the system to verify and to maintain the consistency of the conceptual model described with the semantic network. Structural transformation rules enable the system to transform the semantic network in a normalized and/or optimized relational schema.

The second class of rules collects general knowledge. First, this category includes the definition of the types of arcs and nodes of the semantic network and the general properties of these types. Second, it also contains the definition of relational concepts (i.e. relation, attribute, domain functional dependency) and properties (i.e. Armstrong's inference rules and normal forms). Finally, as we manipulate sets and lists of objects, general knowledges about set theory and lists are also included in this second class.

The third class of rules is composed of a hierarchy of meta-rules which control the sequence of design steps and select the rules to apply at each step and the facts over which these rules operate.

All these classes of rules are encoded in Prolog. Figure 8 portrays the

inheritance rule of figure 7 expressed in Prolog. g and a are predicates which specify respectively the generalization and aggregation arcs. r refers to an association.

```
inheritance <- g(*X1,*X), a(*A,*X),
               insert_clause(a(*A,*X1)),
               delete_clause(a(*A,*X)).

inheritance <- g(*X1,*X), r(*X,*R),
               insert_clause(r(*X1,*R)),
               delete_clause(r(*X,*R)).
```

Fig. 8 : The inheritance rule in Prolog

Another example is a meta-rule which describes a depth-first strategy to search and suppress generalisation hierarchies (see figure 9). s is the specialization arc of the semantic network, x, y, z, w are Prolog variables standing for the node of the generalization hierarchy; transform is a structural transformation rule.

```
depth(*x) <- s(*x,*y),
             s(*y,*z),
             //,
             depth(*y).
depth(*x) <- s(*w,*x),
             insert_clause(father(*w,*x)),
             transform(*x).
depth(*x) <- father(*w,*x),
             depth(*w).
depth(*x)<-delete_clause(father(*y,*z)).
```

Fig.9: An example of a meta-rule expressed in Prolog.

One well-known principle of expert system design is that the modularity and the independence of rules greatly enhance the evolutivity of the system. This is a good philosophy. But unfortunately, when we have a large base of knowledge, this important principle decreases the performances of the system, especially when the Prolog interpreter does not provide a sophisticated search strategy. That is why in some cases we have turned aside from this principle. Indeed, as in some design steps several rules have some overlapping premises, we have choosed to built trees composed of these premises and where each path from the root down to the leaves corresponds to a given rule.

5. THE LOGICAL DESIGN PROCESS

The logical design process generates, from an external description of an application, a sound conceptual schema stored as a semantic network with associated constraints. Then, a fourth normal form relational schema with associated integrity constraints is produced. The global process is divided in steps which are more precisely described below.

This process is performed in a combination of a forward and a backward chaining. The general principle is to successively transform a given specification, trying all the rules until no rule is applicable. This is the definition of the forward chaining. But at each design step, we may use a backward chaining to enforce a consistency constraint for example, or to verify that a given information is not redundant (i.e. not derivable from another information). This is especially the case of functional dependencies.

5.1. THE STEPS OF THE METHODOLOGY

The first step is called the verification step. It performs the validation of the application description in order to generate a sound and consistent conceptual schema. In addition to the syntactic controls, this step checks and solves the problem of homonymous and synonymous informations. It also detects generalization cycles. The system tries to evacuate the possible inconsistencies with the end-user's help.

The second step is called the relational step. It performs the interactive acquisition of constraints and the choice of first normal form relations. Constraints such as intersection and union of classes, cardinalities of relationships (aggregation and association) and functional dependencies between attributes are acquired. Normal form relations are constructed by suppressing generalization hierarchies and separating multivalued attributes.

The last step is called the normalisation step. Normalization is carried out using both the functional dependencies between attributes given in the initial specification, and the cardinalities of associations which allow the system to infere some functional and multi-valued dependencies. The normalization process is composed of two phases : partial

normalization using local functional dependencies (between attributes of the same entity), and total normalization using global functional dependencies (between attributes of different entities).

The result of the logical design process is a set of 4NF relations with their keys (both unique and multiple keys), a set of virtual relations with their deriving relational queries, and a set of constraints including domain constraints and inclusion constraints (in particular, referential integrity constraints). The methodology is characterized by a sequence of steps which alternatively require algorithmic tasks (e.g. verification and normalization) and human decisions (e.g. acquisition of constraints and choice of entities and relationships). The following paragraphs describe in more details how steps two and three are implemented to produce a normalized relational schema.

5.2 PRODUCTION OF A NORMALIZED RELATIONAL SCHEMA

Starting with a sound semantic network, the production of a normalized relational schema is performed during the relational and the normalization steps, as stated above. Each step is composed of three actions. The relational step encompasses the following actions :
R1) The supression of the generalization hierarchies.
R2) The acquisition of aggregation constraints (cardinalities) and the separation of multivalued attributes to obtain 1NF relations.
R3) The acquisition of functional dependencies between attributes of each 1NF relation.

The normalization step includes the following actions :
N1) A partial normalization process using a simplified synthesizing algorithm [BEER79].
N2) The acquisition of association constraints (cardinalities) and the suppression of the association arcs.
N3) A complete normalization process using the decomposition algorithm [FAGI77, ZANI81].

In the first version of SECSI, these six actions are processed in the given order. However, the order of the first three actions may be changed. The chosen order has the advantage of getting cardinalities and functional dependencies more precisely. Indeed, a functional dependency which is valid for the TEACHER attributes is not necessarily valid for the PERSON attributes. For example, we may have NAME(TEACHER)-->ADDRESS(TEACHER) and not NAME(PERSON)-->ADDRESS(PERSON). It is the same problem for cardinalities which may hold at the specialization levels and not at the generalization levels. But changing the action order could improve performances because attributes are not duplicated by inheritance properties and the dialogue of the constraints acquisition would be shorter. In the second version of SECSI, we implement some meta-rules to decide wether it is interesting to begin by step R1, R2 or R3. These meta-rules are essentially based on the number of attributes and specialization entities. The next sub-sections detail each of the preceding actions.

5.2.1 The suppression of generalization hierarchies

The problem is to choose between different nodes of a generalization hierarchy which node(s) must be kept as possible relation(s) and which one must be replaced either by new attributes, or virtual relations, or integrity

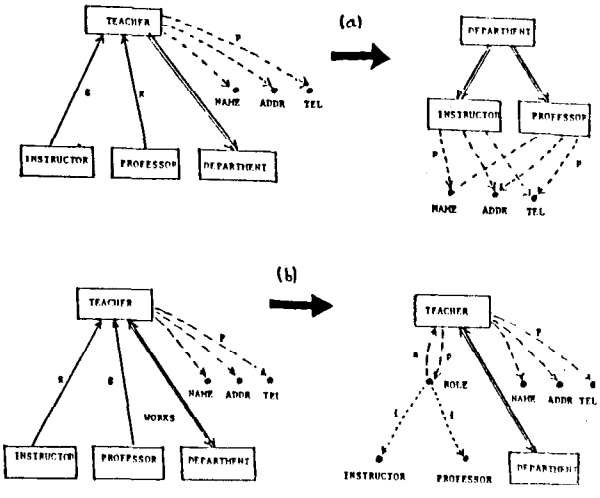

Fig.10: Examples of structural transformations of generalization hierarchies.

constraints. The general principle is to keep the "more semantically referenced" nodes (i.e. the nodes which are surraounded by the greatest number of arcs). The main criterias used are the

number of specialization nodes, the number of specific attributes of each node, the intersection and the union constraints, and the depth of the hierarchy. For example:

```
IF X HAS MORE THAN 3 SPECIALIZATION ENTITIES
  AND THESE SPECIALISATIONS HAVE NO SPECIFIC ATTRIBUTES
  AND THESE SPECIALIZATIONS DO NOT PARTICIPATE TO ANY ASSOCIATION
  AND THERE IS NO INTERSECTION BETWEEN THESE SPECIALISATIONS
  AND THE UNION OF SPECIALIZATION CLASSES IS EQUAL TO THE
      GENERALIZATION CLASS
THEN ADD A NEW ATTRIBUTE NAMED "ROLE" TO THE AGGREGATION OF X
          WHICH DOMAIN IS THE SEQUENCE OF NAMES OF THE
          SPECIALIZATION ENTITIES,
     DELETE THE SPECIALIZATION ENTITIES OF X.
```

This rule is illustrated in figure 10b.

5.2.2. The acquisition of aggregation cardinalities

First, certain cardinalities are given by the end-user in his specification or acquired by a question-answering dialogue such as the following one :

```
SECSI > COULD ANY TEACHER HAVE SEVERAL ADDRESSES?
USER  < YES.
SECSI > COULD ANY TEACHER HAVE SEVERAL PHONE NUMBERS?
USER  < YES.
SECSI > IS THE PHONE-NUMBER DEPENDANT ON THE ADDRESS?
USER  < NO.
SECSI > AND INVERSELY?
USER  < YES.
SECSI > FOR EACH ADDRESS IS THERE ONE OR SEVERAL TEACHERS?
USER  < ONE.
.....
```

Some other cardinality constraints may be inferred from the functional dependencies. For example, if the functional dependency:
NAME(DEPARTMENT)-->ADDRESS(DEPARTMENT)
is given in the description and if the departement has only one name, then SECSI infers that the department has only one address.

At the end of this dialogue, the system has transformed the base of facts (i.e. the semantic network) and provides the first normal form relations by applying transformation rules as those illustrated in figure 11.

The previous dialogue is very important because it prevents some multivalued dependencies to occur. In a certain sense, it prepares the schema for being in 4NF. But it may appear as a surprising approach to "normalize in 4NF" during the first normal form process. Indeed, if we interprete the multivalued dependencies as an independency of two sequences of attributes (or as two merged but different objects), this phenomenon is detected and solved during the previous dialogue and non-trivial multivalued dependencies does not hold further. However, we shall see later that this approach is not sufficient to detect all the possible multivalued dependencies.

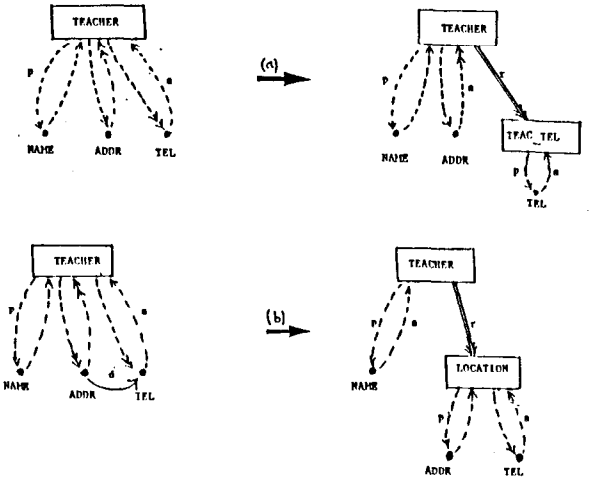

Fig.11: Examples of structural transformation of aggregation

5.2.3. The acquisition of functional dependencies

Functional dependencies can be acquired from four different sources:
(1) the user's description of the application explicitely specifies certain functional dependencies,
(2) the cardinalities of the aggregation arcs enable the system to infere functional dependencies. For example, if an EMPLOYEE has only one SSN and only one ADDRESS, and for each SSN there is one EMPLOYEE, then SECSI infers the functional dependency SSN-->ADDRESS. This is a direct application of the transitive inference rule of functional dependencies if we assume that EMPLOYEE plays a role of an attribute : if SSN --> EMPLOYEE and EMPLOYEE--> ADDRESS then SSN --> ADDRESS,
(3) a dialogue with the end-user is also possible. As for cardinalities, SECSI asks questions of the form :

```
SECSI > DOES THE NAME OF EMPLOYEE DETERMINE HIS SALARY?
USER  < NO.
SECSI > DO NAME AND ADDRESS OF EMPLOYEE DETERMINE HIS SALARY?
.....
```

During this dialogue, the system is

directed by Armstrong's inference rules which enable SECSI to derive new functional dependencies from those given by the user. The system asks questions only for those functional dependencies it could not derive. However, even in this case, this dialogue phase may somewhat appear as very tedious and tiring for the user. Thus instead of searching for possible functional dependencies, we try first to search for impossible dependencies. This is done with the help of some examples of tuples given by the end-user. For example, SECSI asks the following questions:

```
SECSI > PLEASE, WOULD YOU GIVE ME SOME EXAMPLES OF TUPLES
        OF THE RELATION TEACHER(SSN,NAME,ADDRESS,TEL),
        (5 TUPLES AT MOST)?
USER < 1234  DUPONT   PARIS       2224775
     < 1234  DUPONT   MARSEILLE   662532
     < 2500  DURAND   GRENOBLE    886542
     < 3000  PERRIER  PARIS       2740785
     < 3000  PERRIER  LYON        426830
     < .
```

From these tuples of the TEACHER relation, SECSI infers that the following functional dependencies do not hold: NAME --> ADDRESS, ADDRESS --> TEL, ADDRESS --> NAME, NAME --> TEL, (SSN,NAME) --> ADDRESS.
Thus the number of possible candidate dependencies is reduced. However, from this extension of the relation TEACHER, SECSI can say nothing about SSN-->NAME, TEL-->ADDRESS,...Another way to avoid the combinatory explosion in functional dependencies acquisition is to reduce the number of attributes in the left hand side of functional dependencies. Indeed, it does not practically appear as an important constraint to limit this number to four or five attributes. Figure 12 synthesizes the functional dependency acquisition principle.

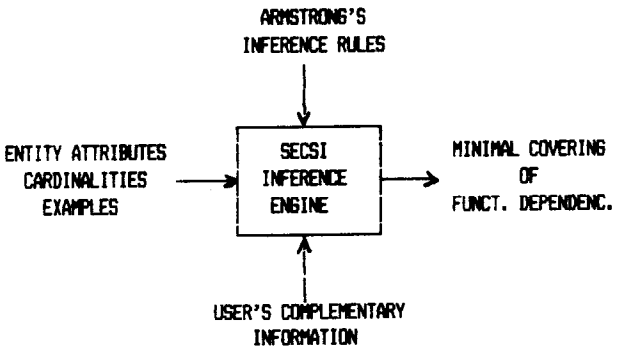

Fig.12: Acquisition of Functional Dependencies.

5.2.4. Partial normalization process

This process is considered as partial because it concerns only the attributes and functional dependencies of a unique entity and it does not handle functional dependencies between attributes of different entities which are not already acquired. This process is also called partial as it is only applied for entities which do not appear as targets of association arcs (r).

This normalization process is based on the second version of the synthesinzing algorithm of [BEER79]. During the previous dialogue phase, whenever a functional dependency holds, SECSI applies the membership algorithm which consists of testing whether a functional dependency is implied or not by those already existing in the base of facts. Then the minimal covering is progressively built and third normal form relations deduced with all their possible keys.

Although the Prolog language is not adapted to this type of algorithms, the efficiency remains acceptable as the number of attributes of an entity is not generally very high.

5.2.5. Acquisition of association cardinalities

Association cardinalities are either given in the initial description of the application or interactively acquired from the end-user with the following dialogue:

```
SECSI > MAY EACH PROFESSOR BE RESPONSIBLE OF ONLY ONE
        OR SEVERAL COURSES?
USER < SEVERAL.
SECSI > MAY EACH COURSE HAVE ONLY ONE OR SEVERAL RESPONSIBLES?
USER < ONE.
SECSI > DOES ANY COURSE EXIST WITHOUT A RESPONSIBLE?
USER < NO.
.....
```

This dialogue determines the (m,n) couples of values from which SECSI infers some functional and multivalued dependencies. For example, the dialogue above produces two couples of cardinalities (0,N) and (1,1) from which SECSI infers functional dependencies:

key(COURSE)-->key(PROFESSOR).

The key variable is later replaced by the corresponding keys found in the normalization process. If the user becomes a little familiar with the system, he may introduce directly his couples of cardinalities to avoid the

preceding dialogue.

The decision of suppressing association arcs depends on the number of arcs involved in each association, the cardinality of each association arc r, and the number of attributes of this association. When associations are organized into a hierarchy, a meta-rule specifies the strategy to search this hierarchy. Figure 13 shows some transformation rules depending on the cardinalities of r. Whenever an arc r is eliminated, a referential constraint is created between the association and the involved entity, or between the involved entities.

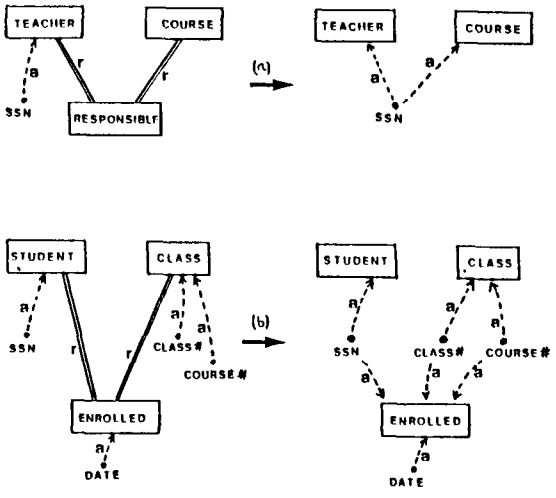

Fig.13: Examples of structural transformations of associations.

5.2.6. Complete normalization process

The suppression of association arcs moves attributes from one entity to another and introduces new functional and multivalued dependencies that make some relations not normalized. Thence, SECSI has to proceed another normalization process based on the decomposition algorithm [FAGI77, ZANI81]. This process concerns all the entities which are not yet normalized by the partial normalization process (i.e. entities which are the targets of r arcs).

The principle of these algorithms is to eliminate by projection all the functional and multivalued dependencies whose the left hand side is not the key of the relation. The process is finite but the relation schemas obtained depend on the order in which dependencies are considered. As in the partial normalization process, the efficiency remains acceptable as generally, entities have not more than two or three dozens of attributes.

5.3. The final results

When the design process described above is terminated, we obtain the following results:

(1) A set of basic relations in 4NF and the various keys of these relations. Figure 14 shows the normalized relational schema produced from the university example portrayed in the same figure. Notice that in the results some new attributes appears (e.g. TEACHER.ROLE and STAFF.ROLE) which were not in the initial description. They have been created to replace specialization entities which have been suppressed during the action 5.2.1 of the design. Some other attributes are duplicated in different relations; they replace the association arcs r that have been deleted in the design action 5.2.6 These attributes are prefixed by the first three characters of the name of the entity from where they have been derived (CLA.NUMBER,COU.NAME,STU.NUMBER) or by the association which has caused the attribute migration. Also in the same example, there are some surprising names of relations FREE-GIFT-STAFF, ADDRESS-TEL-TEACHER coming from the normalization process. These will later be renamed with the user's help (for example put LOCATION instead of ADDRESS-TEL-TEACHER. The key(s) of each relation are specified. As for relation names, some attributes composing the keys may be prefixed by entity names.

(2) A set of virtual relations and the definition of the corresponding relational queries which permit to derive them from the database real relations. These virtual relations correspond to some entities given in the initial description and which have disappeared during the design process. However, with respect to the user, these objects (e.g. PERSON, EMPLOYEE) which exist in the real world must exist in the conceptual schema exactly as other objects (STUDENT,COURSE). Notice that all the transformed entities are not necessarily replaced by virtual relations; some of them are replaced by role attributes (e.g. INSTRUCTOR, PROFESSOR). However, sometimes, both

virtual relations and roles are necessary to capture the semantics of the real world (e.g. HEAD_OF_LABO). In the example portrayed figure 14, relational queries are simply represented by relational operators.

```
** RELATIONS
   ENROLLED ( CLA-NUMBER COU-NAME STU-NUMBER DATE )
   TEACHER ( DEP-NAME SALARY NAME SSN TEA-ROLE )
   STAFF ( DEP-NAME SALARY NAME SSN STA-ROLE )
   STUDENT ( NAME NUMBER )
   COURSE ( TEA-SSN ROOM DAY NAME HOUR )
   CLASS ( NUMBER COU-NAME TEA-SSN )
   DEPARTMENT ( ADDRESS NAME )
   FREE-GIFT-STAFF ( STA-SSN FREE-GIFT )
   ADDRESS-TEL-TEACHER ( TEA-SSN TEL ADDRESS )

** CONSTRAINTS
   key(ENROLLED) : CLA-NUMBER COU-NAME STU-NUMBER
   key(TEACHER) : SSN
   key(STAFF) : SSN
   key(STUDENT) : NUMBER
   key(COURSE) : NAME
   key(CLASS) : COU-NAME NUMBER
   key(DEPARTMENT) : NAME
   key(FREE-GIFT-STAFF) : STA-SSN FREE-GIFT
   key(ADDRESS-TEL-TEACHER) : TEA-SSN ADDRESS

** VIRTUAL RELATIONS
   PERSON = UNION( STUDENT[NAME] TEACHER[NAME] STAFF[NAME] )
   EMPLOYEE = UNION( STAFF TEACHER )
   DIR_OF_LABO = REST( JOIN( STAFF TEACHER
                          / STAFF.ROLE = TEACHER.ROLE )
                     / TEACHER.ROLE = "DIR_OF_LABO" )

** DOMAIN CONSTRAINTS
   STAFF.ROLE = ( DIR_OF_LABO HEAD_OF_SECTION )
   TEACHER.ROLE = ( DIR_OF_LABO INSTRUCTOR  PROFESSOR)
   COURSE.NAME = ( AI DB MATH )

** REFERENTIAL AND INCLUSION CONSTRAINTS
   ENROLLED.CLA-NUMBER = CLASS.NUMBER
   ENROLLED.CLA-NAME = CLASS.NAME
   ENROLLED.STU-NUMBER = STUDENT.NUMBER
   CLASS.COU-NAME = COURSE.NAME
   TEACHER.DEP-NAME = DEPARTMENT.NAME
   STAFF.DEP-NAME = DEPARTMENT.NAME
   FREE-_GIFT-STAFF.STA-SSN = STAFF.SSN
   ADDRESS-TEL-TEACHER.TEA-SSN = TEACHER.SSN

** OTHER SEMANTIC CONSTRAINTS
   COURSE.TEA-SSN = TEACHER.SSN AND TEACHER.ROLE = "PROFESSOR"
   CLASS.TEA-SSN = TEACHER.SSN AND TEACHER.ROLE = "INSTRUCTOR"
```

Fig.14: An example of application run with SECSI.

(3) A set of constraints like domains referential and other general semantic constraints. Domain constraints are relevant for new attributes generated during the design process (especially roles). Referential dependencies are generated to replace the real world associations. They are essential information without which relational joins of tables cannot be done efficiently and the database integrity cannot be maintained. Semantic constraints are all other constraints composed of a conjunction or disjunction of predicates and which capture a given semantics graphically expressed in the semantic network or in the user's application in general. All these constraints are expressed in a specific language described in [SIMO84], that is the language of the SABRE system.

6. CONCLUDING REMARKS AND FURTHER RESEARCH DIRECTIONS

We have described the main features of an expert system for database design. This system is written in PROLOG and runs on MULTICS at INRIA. The main originalities of the system are :
(1) It does integrate a complete methodology for database design, starting from a naive description of the application and using intensively dialogues with the end-user.
(2) It is strongly based on a semantic data model which is implemented as a semantic network in the system.
(3) It encompasses most of the simple theory about database design (e.g. normalization, dependency inference rules ...) which is expressed as PROLOG clauses.
(4) It is evolutive in the sense that we can add new design rules in the system.
(5) It is a tool integrated in the relational DBMS SABRE in order to facilitate database design and creation.

However, the system is far from being complete. Many points have to be improved including the graphical interface, the expert interface, the design algorithms and the explanation of the decisions... Further steps which are not yet addressed in the current implementation are the view integration and the physical design. New versions integrating these aspects are currently in specification.

The substantial results already achieved with the first version of SECSI lead us to state that expert systems are very suitable to database design. They introduce a new design style in the manner of directing the dialogues, correcting the inconsistencies and justifying the results. They also

introduce new capabilities for database restructuring. Expert systems may also open new possibilities in database teaching.

REFERENCES

[BADA81] BARR A. , DAVIDSON J. Representation of Knowledge (in Handbook of AI, Barr & Feigenbaum ed., Comp. Sce Depart., Stanford University)

[BEER79] BEERI C., BERNSTEIN P.A. "Computational problems related to the design of normal form relation schemes" ACM Transact On Databases, vol4,nb1, march 1979.

[BOUZ83a] BOUZEGHOUB M. "MORSE: A Functional Query language and its semantic data model. INRIA RR270 and Proceed of 84 Trends and Application conf on Databases, IEEE-NBS Gaithersburg (USA), 1984.

[BOUZ83b] BOUZEGHOUB M. et GARDARIN G. "The design of an expert system for database design" in New Applications of Databases. Gardarin and Gelenbe edit. Academic Press 1983.

[BOUZ84] BOUZEGHOUB M., GARDARIN G., METAIS M. "SECSI: Une application des systemes experts a la conception des bases de donnees relationnelles" Actes colloque internat. d'Intelligence Artificielle, Marseille oct. 1984.

[BROW83] BROWN & STOTT-PARKER LAURA: A formal Database model and her Logical Design Methodology Proceed. VLDB Conf, Florence 1983.

[BROD81] BRODI M.L. "On Modelling Behavioural Semantics of Data Bases (Proceed of 7th VLDB Conf IEEE 1981)

[BROD84] BRODIE M., MYLOPOULOS J., SCHMIDT Y. On Conceptual Modelling: Perspectives from Artificial Intelligence, Data Bases and Programming languages. Springer-Verlag, NY 1984.

[CARL83] CARLIS J.V., MARCH S.T., DICKSON G.W. Physical Database Design: A DSS Approach. in Information and Management 6(1983).

[CERI83] CERI S. (edit) "Methodology and Tools for Database Design. North Holland 1983.

[CHEN76] CHEN P.P. "The Entity Relationship Model - Toward a Unified View of Data" (ACM TODS V1, N1, March 1976)

[COBB84] COBB R.E. , FRY J.P. and TEOREY T.J. "The Database Designer's Workbench. Information System Sces nb 32, 1984.

[CODD79] CODD E.F. Extending the Database Relational Model to capture more Meaning. ACM Trans. On Databas Systems, 4,4, Dec 79.

[DaEn84] Database Engineering Revue vol7 nb4, dec84. Special issue on Database Design Aids, Methods and Environments.

[DAVI83] DAVIS C.G. et al (edit) Entity Relationship Approach to Software Engineering. North Holland Publ. Co 1983.

[FAGI77] FAGIN R. Multivalued Dependencies and a New Normal Form for Relational Databases. ACM Trans. Database Systems, vol2,nb3 sept 77.

[GARD82] GARDARIN G. "Bases de Données: les systèmes et leurs langages" edit. Eyrolles Paris 1983.

[GARD83] GARDARIN G. et al Design of a multiprocessor Relational Database System. Proceed IFIP Cogress, Paris Sept 1983.

[HAMM81] HAMMER N. and McLEOD D. Data Base Description with SDM: A Semantic Data Model (ACM TODS V6, N3, Sept 81)

[HAYE83] HAYES-ROTH F., WATERMAN D.A., LENAT D.B. Building Expert Systems Addison-Wesley pub. Co. Inc. 1983

[HAYE84] HAYES-ROTH F. The Knowledge Based Expert System: a tutorial. Computer revue vol17, nb9, Sept 1984.

[KENT79] KENT W. Limitations of Record-Based Information Models. ACM Trans. Database Systems 4,1,1979.

[LAUR82] LAURIERE J.L. Les systèmes experts (AFCET TSI No 1 et 2 1982)

[MYLO80] MYLOPOULOS J. BERNSTEIN P.A. WONG H.K.T. A language facility for designing database intensive applications" ACM TODS vol5,nb 2, 1980.

[SHIP81] SHIPMAN D.W. The Functional Data Model and the Data Language DAPLEX ACM TODS V6, N1, MAR 81

[SIMO84] SIMON E. and VALDURIEZ P. Design and Implementation of an Extendible Integrity Subsystem ACM SIGMOD 1984, ACM Ed.

[SMIT77] SMITH J.M. and SMITH D.C.P. Data Bases Abstractions Aggregation and Generalization ACM TODS June 77

[TAHN84] Tan TAHN JOO, TAN KAH POH, GOH AH MOI, DATADICT: A data analysis and logical database design tool. Proceed. VLDB Conf. Singapore, Aug 1984.

[ULMA80] ULLMAN J.D. "Principles of Database Systems" computer Sce Press, 1980.

[WASS82] WASSERMAN A.I. and SCHNEIDER H.J. editors Automated tools for Information System Design, North Holland Publ. Co. 1982.

[ZANI81] ZANIOLO C and MELKANOFF M.A. On the Design of Relational Database Schemata. Trans. Database Systems ACM, vol 6, nb 1, march 1981.