Semantic Modeling of Object Oriented Databases

Mokrane Bouzeghoub, Elisabeth Métais

Laboratoire MASI, Université P. et M. Curie - CNRS, Centre de Versailles
45, avenue des Etats-Unis 78000 Versailles, France
Email: mokrane@Zeus.ibp.fr

Abstract:

This paper describes a design methodology for an object oriented database, based on a semantic network. This approach is based on the assumption that semantic data models are more powerful and more easy to use than current proposed object oriented data models. They are especially more poweful in representing integrity constraints and various relationships. Object oriented data models are generally based only on class hierarchies and inheritance, plus their ability to represent the behavior of objects. But this latter capability is generally provided through an algorithmic language which cannot be considered as a conceptual language. This paper describes a design procedure which generates an object oriented database schema (both the structural aspect and the dynamic aspect) from an abstract specification given in a high level language. This specification language is built upon a semantic network and allows to define integrity constraints and behavior rules. This approach is presented through a CASE tool environment.

1. Introduction

Like relational databases, the design of an object oriented database is a complex art which needs many expertise in the domain. The simultaneous modeling of the structural aspect and the behavioural aspect of objects increases the complexity of the design. The current object oriented data models are mainly defined by a few basic constructors (like the tuple constructor and the set constructor) and a taxonomy of objects (i.e. hierarchy of classes and inheritance). The power of object oriented data models is highlighted by their ability to describe the dynamic behavior of the objects (methods). However, as generally proposed in the object oriented database systems, this dynamic description is made in a procedural language; this fact makes the specification of the methods too difficult at the conceptual level. Another weakness of current object oriented data models is that, except through methods, they do not easily permit to specify integrity constraints on the objects.

Except for the dynamic aspect, the expressive power of semantic data models is stronger than that of object-oriented data models. Various relationships and integrity constraints can easily be specified. Class hierarchies and inheritance are generally defined in the same way. The dynamic aspect can be fulfilled by introducing the concept of behavior in the semantic data models. In some sense, this was already done in the AI domain by the concept of script which has been developped to enhance the expressive power of semantic networks and frames. We follow the same approach and describe the behavior of a semantic data model by means of production rules. This kind of a declarative language permits to avoid the complexity of procedural languages which are generally used in object oriented data models. The behavior of each object in the semantic data model will be described by one or several rules expressing either integrity constraints or any management rules concerning objects.

This paper highlights on one hand the object-oriented database design methodology we have developped, and on the other hand the design tool which supports this methodology. This methodology is based on two design levels: a semantic object oriented level and an operational object oriented level (Figure 1). The process of interactive acquisition, completeness and consistency checkings of the behavioural rules is particularly emphasized in the first level. At the second level, we use as an operational object oriented data model, called O2 model, which was developed by Altaïr project [Bancilhon 87]. Then a mapping process between the two models is proposed. Besides the data structure mappings, the transformation of a semantic object oriented schema into an operational object oriented schema consists, among others, in the generation of procedural methods (C written) from a declarative language specification (production rules).

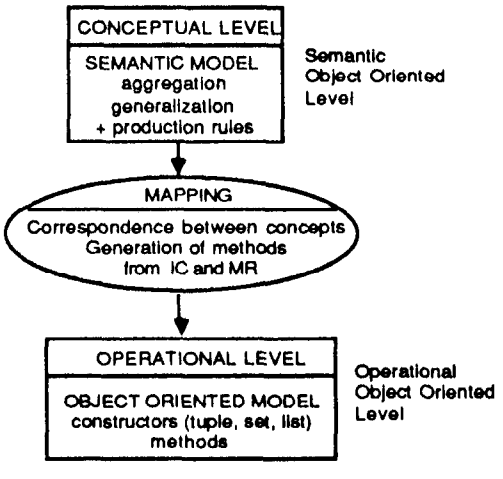

Fig.1: A two level design approach

The second section of this paper will give an overview of the two data models used. Section 3 describes the general mapping process from the semantic network to the object oriented model O2. This mapping process is only concerned with the structural aspect and with some static integrity constraints. Section 4 describes the mapping of general integrity constraints (first order formulas) and behavioural rules (production rules) to the operational level expressed in C language. Problems of method definition and attachment are also addressed in this section. Section 5 gives a general flavour of the modelling prototype which was designed. Section 6 concludes on the obtained results and the remaining problems to solve.

2. The Hierarchy of Data Models Used

This section gives a general overview of the two data models considered in the methodology. The first one is a generic semantic network called Morse. The second one is an object oriented data model called O2.

2.1. The Semantic Network Model

A semantic network is an oriented diagram where the nodes represent real world objects and the arcs represent semantic relationships between these objects. In addition, constraints can be defined over these nodes and arcs. In the following, such a semantic network data model is designated by the name Morse [Bouzeghoub 84]. This model is formalized in such a way it can represent the most important concepts used in semantic data models. The objective of this formal definition is to provide a general framework whithin which a CASE tool could be specified and programmed. The external vision of the model could be any desired diagram (e.g. Entity-Relationship) or abstract syntaxe.

An object is a generic term to designate the different real world individuals refered to in Morse schemas. We distinguish four categories of objects: in one hand *instances of atomic objects* (IA) and *instances of molecular objects* (IM), in the other hand *classes of atomic objects* (NA) and *classes of molecular objects* (NM). Then, in the following, we use the term object in a generic way, and whenever necessary, we use the more specific term.

The distinction between atomic objects and molecular objects permits to highlight their structural links for a better specification of the corresponding constraints. Atomic objects have values taken from basic domain such as: integer, real, boolean and string. The set of all atomic values in all domains are refered to by the name VA. Molecular objects have molecular values which are composed from the corresponding atomic objects which constitute the molecular object. Both atomic objects and molecular objects have unique identifiers which are independant of their values.

Semantic links are basic binary relationships between the different categories of objects mentioned above. These binary relationships formalize the well-known concepts of *aggregation* and *generalization* [Smith 77]. Specific refinements of these concepts are introduced to take into account the distinction between atomic objects and molecular objects. The aggregation concept is refined as *atomic aggregation* (arc a(X,Y)) and *molecular aggregation* (arc r(X,Y)). Generalization is refined as *instance generalisation* (arc c(X,Y)) and *class generalization (arc g(X,Y))*. Each binary relationship has its reverse link (respectively p(Y,X), o(Y,X), i(Y,X), s(Y,X)). To simplify the graphical representation, we use only one specification which subsumes the other (for example p, o, c and g) except if constraint specification is needed for each specific arc.

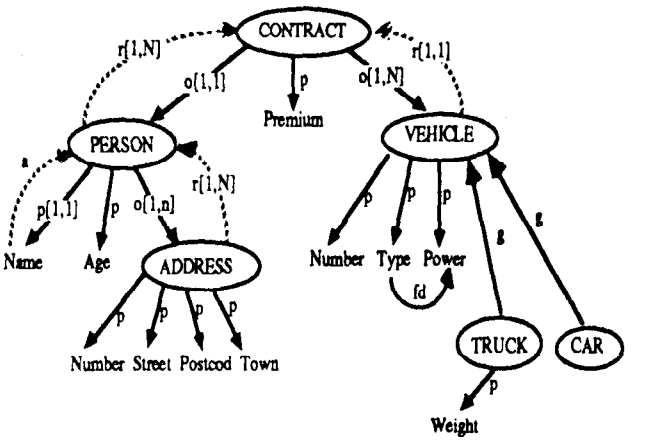

Fig.2: An example of a semantic network

The inheritance is one of the interesting properties of generalization hierarchies; each atomic or molecular component of an object X can be transfered by inheritance to objects X1...Xn, if these latters are sub-classes of X. Inversely, each instance of a sub-class is an instance of its super-classes. We say that components of objects propagate toward the leaves of the hierarchy whereas the instances propagate toward the root(s) of the hierarchy.

Different integrity constraints can be specified in a Morse semantic network to enhance its capability to capture more meaning from the real world. Among these constraints, we can mention domains, cardinalities, functional dependencies, keys, intersection and disjunction of classes, etc. In the semantic network, some of these constraints are defined over nodes, others are defined over arcs. The constraints are specified either as a complementary information of binary arcs or as new predicates. For example, cardinality constraints are expressed as complementary information over a/p arcs and r/o arcs, while other constraints like functional dependencies are represented by a new fd arc:

```
a(Number,VEHICLE,[1,1]),
p(VEHICLE,Number,[1,1]),

a(Type,VEHICLE,[1,N]),
p(VEHICLE,Type,[1,1]),

a(Power,VEHICLE,[0,N]),
p(VEHICLE,Power,[1,1]),

r(VEHICLE,CONTRACT,[1,1]),
o(CONTRACT,VEHICLE,[1,N]),

r(PERSON,CONTRACT,[1,N]),
o(CONTRACT,PERSON,[1,1]).

fd(VEHICLE,lhs(Type),rhs(Power))
```

Graphically, a given semantic network can be represented as portrayed in figure 2.

2.2. The Object Oriented Data Model

The O2 data model belongs to the category of the so-called object oriented data models [Lecluse 87]. Then, its basic concepts are objects and types, type constructors and type hierarchies. The data manipulation language could be the C language with embedded O2 expressions (called CO2) [Haux 88] or an SQL like declarative query language (called LOOQ).

In the O2 data model, an *object* is composed of an identifier (the name of the object) and a value. Values could be either: (i) atomic values (integers, reals, booleans, strings), for example: (i_1,22), (i_2,3.14); (ii) tuple values, for example (i_3, [name:"John",age:22]); and (iii) set values, for example (i_4,{red, black, green}). Objects can be defined by construction using list, tuple and set constructors. Objects can mutually reference each other. For example:

(i_7, [name:"John",wife:i_8]), (i_8, [name:"Mary",husband:i_7]).

A class is an abstraction which represents a set of objects with their behavior. A class is composed of two parts: (i) a type which contains the structure that characterizes all the instances of the class, (ii) methods which contain operations which will be applied to these instances. A class may have a basic type (integer, real, boolean, string) representing atomic objects, a tuple type representing objects with tuple values or a set type representing objects having set values. The following expressions are examples of classes : Person=name:string,age:integer], Employees={p:Person}. The constructors could be composed to create more elaborated types (e.g. sets of tuples or tuples of sets).

The O2 data model makes a clear distinction between identified objects and non identified objects. The formers can be stored and manipulated independently, while the latters exist only as property values of other objects. For example, in the following specifications , persons and vehicles could be manipulated independently: Person= [name:string, age:integer, vehicle:Vehicle], Vehicle= [number:integer, color:string] But in the following example, the object vehicle exists only as a composite attribute value of person:

Person=[name:string,age:integer,
vehicle:[number:integer,color:string]]

A partial order between types defines a hierarchy of types within which the inheritance concept permits to transfer components from one type toward its subtypes [Lecluse 88].

A method is a procedure which is associated to a type in order to describe the behavior of the instances of this type. Methods introduce the notion of **encapsulation** which permits the manipulation of objects without any knowledge about their structure, nor about the internal code of the procedures corresponding to these methods.

The CO2 language is an embedded database language (O2) into a procedural host language (C) [Haux 88]. Besides the usual programming of algorithms, it permits to specify and access database objects. Objects are manipulated through methods. A method is characterized by its signature (its name, its type and the type of its parameters) and its body (procedure). The following example shows the declaration of types and the programming of methods in CO2 (version 1.0) :

```
/* type declaration */
add class Person
type tuple(name:string,age:integer,
     address:string,children:set(Person))
add class Agent inherits Person
type tuple(code:string,salary:integer)

/* method declaration */
method category: string is public
body category:string in class Agent CO2
                /* method procedure */
{    if (self->salary > 50)
     return("VIP");
}
```

The keyword **inherits** defines a hierarchy of types. The keywords **public** makes the object-integrity method visible from anywhere. The keyword **in class CO2** defines the class for which this body is defined; this is useful to solve ambiguities of names, as method bodies can be specified independently of the class description. The brackets {} delimite the C source statements of the procedure.

The definition of a database schema in CO2 needs the knowledge of the objects structure, the status of objects, i.e. identified object or non identified object (value), and the sharing of the objects.

3. Mapping from the Semantic Level to the Operational Level

The CO2 model describes both the static aspect (data structures) and the dynamic aspect (methods). Relationships between objects or object classes are not represented by a specific concept; but they are represented by a uniform way based on objects composition and objects sharing. As in the relational model, references are the unique way to represent relationships between objects. Integrity constraints are not considered as specific concepts of the model; they are defined in a uniform way as any procedure describing the behavior of an object. The object identity allows to make a clear distinction between objects having their own existence, and values which are only relevant when characterizing other objects. The object identity is represented in CO2 by different syntactic forms.

The semantic data model Morse is concerned only with the static aspect. The different categories of aggregation arcs allow to specify different types of relationships between objects. Integrity constraints are represented as declarative assertions on the data structure.

In the following we are only interested in the mapping from Morse to CO2 and not in the reverse mapping. First we consider the structural mappings between the two models, then we study the representation of constraints by methods, and we finally describe the general mapping process. This plan is made only for the soundness of the paper; in fact structural mapping rules often depend on the integrity constraints [Bouzeghoub 90].

3.1. The mapping between objects

An atomic object defined in Morse is equivalent in O2 either to an identified atomic object or to an atomic value (non identified object) inside another object. A molecular object defined in Morse is equivalent to either an identified tuplestructured object or to a tuple value in O2. A class of objects defined in Morse is partly equivalent to a class of objects defined in O2. Indeed, and as stated before, Morse

classes describe only the static aspect of the objects, while O2 classes describe their behavior too, thanks to methods. The first part (a) of Figure 3 summerizes the correspondence between the Morse objects and the O2 objects.

3.2. The mapping between constructors

Both atomic and molecular aggregations defined in Morse are equivalent to the tuple constructor of the O2 model. More precisely, we have to include what is considered as domain constraints in Morse to obtain what is considered as attribute basic type in O2. For example, the following Morse specification:

```
p(PERSON, Name)        dom(Name,string)
p(PERSON, Age)         dom(Age,integer)
o(PERSON, Address)
p(Address, Number)     dom(Number,integer)
p(Address, Street)     dom(Street,string)
p(Address, Postcod)    dom(Postcod,integer)
```

will be mapped into O2 as for example:

```
Person=[Name:string,Age:integer,Addr:Address]
Address=[Number:int,Street:string,Postcod:int]
```

which can be described in CO2 by the following statements:

```
add class Person
type tuple (Name:string,Age:integer,
                Addr:Address)
add class Address
type tuple(Number:int,Street:string,
             Postcod:integer))
```

If we consider that all of Name and Age are values of the Person (thus they are not identified), but the Address is an object by itself (thus it is identified). Addr is called a reference; it is considered as an attribute of Person which references another object, i.e. Address.

The classification/instanciation defined in Morse is partly equivalent to an O2 class. In fact the Morse abstraction can define a class only by extension, without necessarily describing its structure. The generalization/specialization is equivalent to the inheritance hierarchy in O2. In Morse, a given class can be defined by generalization from other classes even the structures of these latters are unknown. Inversely, a Morse subclasse can be defined as a restriction of a superclass, but without any refinements on its structure. This makes the generalization/specialization more general than a partial order of types which is defined in O2.

The inheritance is defined in Morse as a logical property which propagates components and constraints of generic classes to their subclasses. In the O2 model, there is a uniform formalization of hierarchies of types and inheritance (partial order of types). The part (b) of Figure 3 summerizes the different mappings between the Morse constructors and the O2 constructors.

3.3. The mapping of the constraints

Semantic integrity constraints are useful for many reasons: (i) to check the consistency of the object structure and values, (ii) and possibly to assist in the decision process which determines whether a Morse object coincides or not with an O2 object. Except for the usual domains which are represented by basic types in O2 (integer, real, boolean, string), all the other Morse integrity constraints are represented by methods in the O2 model. In the following, we illustrate this latter case with cardinalities and functional dependencies. Methods which implement integrity constraints are particular in the sense they are not directly invoked by the users but by other methods which guarantee the encapsulation of the concerned object. The part (c) of Figure 3 summerizes the different mappings between the Morse constraints and the O2 concepts.

	MORSE CONCEPT	O2 CONCEPT
(a)	atomic object	atomic object/ atomic value
	molecular object	structured object/tuple value
	class	class
	subclass	subclass
	instance	object
	object identifier	object identifier
(b)	atomic aggregation	tuple constructor
	molecular aggregation	tuple constructor
	Class generalization	Inheritance
(c)	Cardinality	set constructor method
	Functional dependency	method
	Keys	method
	General Integrity Constraint	method

Fig.3: Correspondence between Morse and O2 concepts

Among various constraint we can specify over the semantic network, we consider here the mapping of cardinalities. Formally, cardinalities characterize binary relationships (a/p and r/o arcs) by specifiing the frequence of object participation in a given binary relationship. More precisely, a cardinality is a couple of values [m,n] which respectively specify the minimum and the maximum number of a given relationship instances to which the same object could participate. Cardinalities where n=1 are called monovalued cardinalities and those where n>1 are called multivalued cardinalities. In the following, we study the methods which will implement these constraints. As we have several situations, we will only focus on two examples.

Case 1: p(X,Y,[1,1]) : which specifies that for a given instance of X, there is only one instance of Y. For example:

```
p(PERSON, Name,[1,1])   Dom(Name, string)
p(PERSON, Age,[1,1])    Dom(Age, integer)
```

will be implemented into O2 as:

```
add class PERSON
type tuple (Name:string,Age:integer)
method  Nulle_value:boolean
body Nulle_value:boolean
in class PERSON CO2
{if((!(self->Name==(o2 string) NULL))
    && (!(self->Age==(o2 int) NULL)))
    {return (true);}
    else return (false);
}
```

Case 2: o(X,Y,[1,N]): which specifies that for a given instance of X, there is N instances of Y. For example:

```
o(PERSON, Address, [1,N])

p(Address, Number, [1,1])     Dom(Number, integer)

p(Address, Street, [1,1])     Dom(Street, string)

p(Address, Postcod, [1,1])    Dom(Postcod, string)

p(Address, Town, [1,1])       Dom(Town, string)
```

will be mapped into O2 as:

```
add class PERSON
type tuple(Addr:setof(Address))
method Bounded_set(min:integer,
                   max:integer):boolean
add class Address
type tuple(Number:integer, Street:string,
           Postcod:string, Town:string))
body Bounded_set(min:integ, max:integ):bool
in class PERSON CO2
{    O2 set(Address) x;
     x = (self->Addr);
     if ((min =< count(x)) && (count(x) =< max))
       {return (true);} else return (false);
}
```

3.4. The object identity and the object sharing

In the Morse semantic data model, everything is considered as an object. Each object has a unique identification, then objects can be shared between different other related objects. In the O2 object oriented data model, there are objects and values; objects are sharable while values are not. So, when mapping a Morse schema into an O2 schema, we have to decide whether a Morse object can be considered as an O2 object or as an O2 value. This decision mainly depends on the user's desire in the way to manipulate his database. He could arbitrarily decide whether a given Morse object is an O2 object or value. For example, for the mapping of the following Morse schema (figure 4), he can envision many solutions:

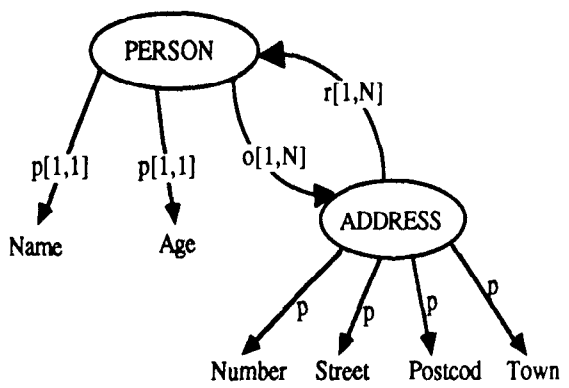

Fig.4: Morse objects

Solution 1: One O2 object PERSON describing the whole Morse structure:

```
PERSON=[Name:string, Age:integer,
        Address:{[Number:integer,
        Street:string,Postcod:integer,
        Town:string]}]
```

All other components are considered as values characterizing a person.

Solution 2: One O2 object ADDRESS corresponding to the whole Morse structure:

```
ADDRESS:[Number:integer,Street:string,
         Postcod:integer, Town:string,
         Person:{[Name:string,
                  Age:integer]}]
```

In this case, persons do not have any existence, they are just characterizing addresses.

Solution 3: Two O2 objects corresponding to the two Morse molecular objects:

```
PERSON=[Name:string,Age:integer,
        Addr:{ADDRESSE}]
ADDRESSE=[Name:integer, Street:string,
          Postcod:integer, Town:string,
          Pers:{PERSON}]
```

In this case there is a mutual reference between the two objects. A person references its set of addresses and an address references its set of persons.

There are many other solutions where we can consider that towns or telephones are independant objects. To decide between all these solutions, a computer design tool can help in the decision process by taking into account several heuristics derived, for example, from the following parameters:

- *Users'operations* and general constraints defined on the Morse objects: basic operations like insert, delete and update, can be considered as the main means to identify objects. We shall see in the next section how these operations are defined in the Morse semantic data model.
- *Cardinality constraints* defined over arcs **a/p** and **r/o** of the semantic network: if the minimal cardinality of one of these arcs is equal 0, then the origine object of the arc can exist independently of the related one.

4. Extending the Semantic Data Model to Represent General Constraints

This section describes the generalized integrity constraints. Many previous works have been done to manage integrity constraints [Brodie 81] [Brodie 84] [Brodie 86] [Bertino 84] [Tsalgatidou 90] Different formalisms have already been proposed to specify integrity constraints in the conceptual schema, for example [Morgenstren 89] allows their specification through constraints equations and [Olivé 89] uses a Deductive conceptual model. In our approach, general integrity constraints are first order logic formula whose variables refer to the content of the semantic database. Before presenting these constraints, let us give a formal representation of a semantic database as well as for its conceptual schema and for its extension. This representation is not intended to implement real databases but just to give a formal abstract representation in order to correctly specify integrity constraints.

4.1. The representation of a semantic database

A Morse database schema is composed of:

- the list of names of all classes of atomic objects (i.e. instances of NA),
- the list of names of all classes of molecular objects (i.e. instances of NM),
- for each atomic object, its domain values (basic type),
- for each molecular object, its data structure (i.e. the set of all its p/a and o/r arcs),
- for each binary relationship (i.e. p/a and o/r arcs), its cardinalities,
- for each multiple reference to the same component, the different roles played by the component in the abstraction.

For Example:

```
i(NA,Name,string)
i(NA,Age,integer)
i(NA,number,integer)
...
p(PERSON,Name,[1,1][1,N])
p(PERSON,Age,[1,1][1,N])
p(VEHICLE,Power,[1,1][0,N])
...
i(NM,PERSON)
i(NM,VEHICLE)
i(NM,CONTRACT)
...
o(CONTRACT,PERSON,[1,1][1,N])
o(CONTRACT,VEHICLE,[1,1][1,1])
...
g(CAR, VEHICLE)
```

As previously stated, everything in Morse is an object. Then each atomic or molecular object is formally identified. The relationship between an atomic object identifier and its corresponding value is represented by a specific predicate v. The relationship between a molecular object identifier and its corresponding structured value is represented by a sequence of v predicates. This systematic identification of all objects implies a systematic sharing of objects. Then values of objects are represented only once. This identification permits also an independent manipulation of all object classes. The generalization arcs (i.e. g/s) are not directly represented in a database extension. They are captured by the inclusion of sets of identifiers with respect to the generalization hierarchy. The following example is an extension of the previous database schema:

```
i(PERSON,P1)      i(Name,N1),      v(N1,dupond)
                  i(Age,A1)        v(A1,33)
i(PERSON,P2)      i(Name,N2),      v(N2,durand)
                  i(Age,A2)        v(A2,44)
i(VEHICLE,V1)     i(Number,I1)     v(I1,123)
                  i(Power,W1)      v(W1,5)
i(VEHICLE,V2)     i(Number,I2)     v(I2,345)
                  i(Power,W2)      v(W2,7)
i(VEHICLE,V3)     i(Number,I2)     v(I2,345)
i(CONTRACT,C1)    i(Premium,M1)    v(M1,5500)
i(CONTRACT,C2)    i(Premium,M2)    v(M2,6000)
```

Obviously this representation is not defined for implementing real databases, but just as a formal representation for a formal reasoning. It can be considered as an abstract representation of the content of a given database. This representation permits a better understanding of the constraint specifications, and provides a convenient framework for a CASE tool.

4.2. The internal representation of general integrity constraints

A general integrity constraint is a first order closed formula, restricted to conjunction connectors and at most only one implication symbole. Variables can be quantified existencially or universally. The universe of discourse in which these formulas are interpreted is constituted as follows:

- a set of constants: composed of (i) the union of atomic objects domains (VA), (ii) the union of atomic objects identifiers (IA) and molecular objects identifiers (IM) and of (iii) the union of class names of atomic objects (NA) and class names of molecular objects (NM),
- a set of variables taking their values in the previous defined universe of discourse,
- a set of predicates: composed of (i) all atomic and molecular aggregation relationships (i.e. p/a and o/r arcs), (ii) instance generalization and class generalization relationships (i.e. c/i et g/s arcs), (iii) usual mathematic predicates : $<, >, \leq, \geq, =, \neq$, and the v (value) predicate.

For example, over the previous database schema, we can define a general integrity constraint which states that if the vehicle power is greater than 10 and the person's age is less than 20, then the contract premium is at least equal to 5000.

```
IC1:  ∀P ∀C ∀V ∀G ∀S  ∀M ∀VG ∀VS ∀VM
      [i(PERSON,P) ∧ i(VEHICLE,V) ∧
       i(CONTRACT,C) ∧ i(Age,G) ∧
       i(Power,S) ∧ i(Premium,M)
         ∧ o(C,P) ∧ o(C,V)
         ∧ p(P,G) ∧ v(G,VG) ∧ VG<20
         ∧ p(V,S) ∧ v(S,VS) ∧ VS>10
         ∧ p(C,M) ∧ v(M,VM)]
         ->[VM≥5000]
```

As these constraints are specified using the same semantic arcs as for describing the static data structure, they can be represented by a semantic network in which each variable or constant is represented by a node. Variable nodes are considered as instances of object classes. The quantifier corresponding to each variable is represented as a complementary information of the arc I relating a variable to its class. For example, i(Person,x,∀) describes a variable x universally quantified over the class Person. As the order of the quantifiers is meaningful in a given formula, an indice is associated with the quantifier. For example, i(Person,x,∀,1). Finally, new binary arcs (inf, sup, equ, einf, esup, diff) are added to the semantic network to represent the predicates: $<, >, =, \leq, \geq$. To give more meaning to this representation, we must complete each predicate to specify whether it belongs to the left hand side or to the right hand side of the rule representing the integrity constraint.

```
IC1:  i(PERSON,P,∀,1,left,IC1)
      i(VEHICLE,V,∀,2,left,IC1)
      i(CONTRACT,C,∀,4,leftright,IC1)

      i(Age,G,∀,5,left,IC1)
```

```
i(Power,S,∀,6,left,IC1)
i(Premium,M,∀,7,right,IC1)

o(C,P,left,IC1)
o(C,V,left,IC1)

p(P,G,left,IC1)  v(G,VG,left,IC1)
inf(VG,20,left,IC1)

p(V,S,left,IC1)  v(S,VS,left,IC1)
sup(VS,10,left,IC1)

p(C,M,left,IC1)  v(M,VM,left,IC1)
sup(VM,5000,right,IC1)
```

The following schema illustrates the representation of the constraint **IC1**. The lower part represents the static data schema, the upper part represents the behavioral schema. In this latter one, we have separated the rule left hand side part and the right hand side part; although some nodes appear in the both parts. When the constraint doesn't have an implication symbol, the semantic network doesn't have a left hand side.

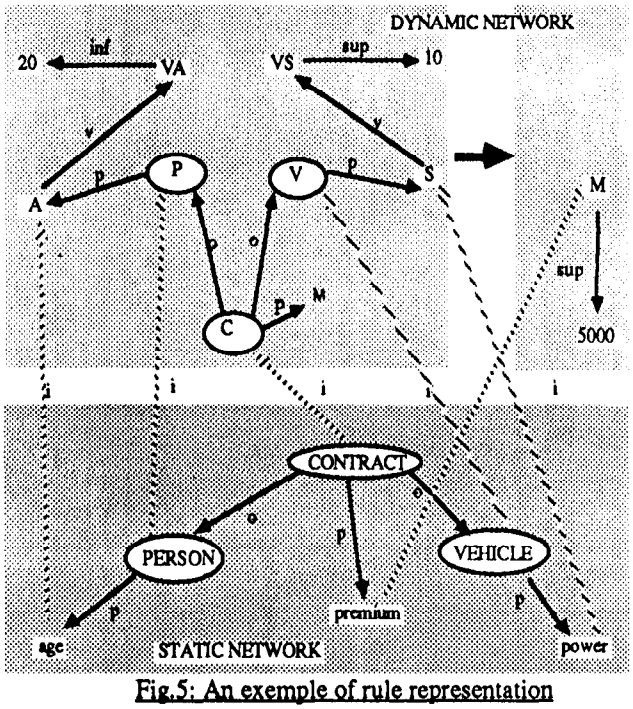

Fig.5: An exemple of rule representation

4.3. The semantic object oriented language

The Morse language exposed in the previous section is a formal language to represent the detailed description of a conceptual schema. This language is not intended to be used by end-users. Consequently, we need a friendly user interface to specify data structures and constraints. This subsection describes the declarative language offered to specify an application and its integrity constraints.

a) Specification of data structures

(1) Each set of p or o predicates which defines the structure of a molecular object class is replaced by the following statement, if the structure is composed of atomic objects:

$$\begin{aligned}&\mathbf{X(A_1:dom_1,\ldots,A_n:dom_n)}\\&\mathbf{<=>}\ i(NA,A_1,dom_1),\ \ldots,i(NA,A_n,dom_n),\\&\quad i(NM,X)\\&\quad p(X,A_1),\ \ldots,\ p(X,A_n)\end{aligned}$$

or by the following if the structure is composed by molecular objects:

$$\begin{aligned}&\mathbf{X(Y_1,\ldots,Y_n)}\\&\mathbf{<=>}\ o(X,Y_1),\ldots,\ o(X,Y_n),\ i(NM,X)\end{aligned}$$

or by the following statement if the structure is either composed of atomic objects and molecular objects.

$$\begin{aligned}&\mathbf{X(A_1:dom_1,\ldots,A_n:dom_n,Y_1,\ldots,Y_m)}\\&\mathbf{<=>}\ i(NA,A_1,dom_1),\ \ldots,i(NA,A_n,dom_n),\\&\quad i(NM,X)\\&\quad p(X,A_1),\ \ldots,\ p(X,A_n)\\&\quad o(X,Y_1),\ldots,\ o(X,Y_n)\end{aligned}$$

If the cardinality constraints are specified, we shall have the following description:

$$\begin{aligned}&\mathbf{X(\{A_1:dom_1\}[am_1,an_1],}\\&\quad\ldots,\\&\quad\mathbf{A_n:dom_n,[am_n,an_n],}\\&\quad\mathbf{\{Y_1\},[rm_1,rn_1],}\\&\quad\ldots,\\&\quad\mathbf{Y_m,[rm_n,rn_n])}\\&\mathbf{<=>}\ i(NA,A_1,dom_1),\ \ldots,i(NA,A_n,dom_n),\\&\quad i(NM,X)\\&\quad p(X,A_1,[1,N][am_1,an_1]),\\&\quad\ldots,\\&\quad p(X,A_n,[1,N][am_n,an_n]),\\&\quad o(X,Y_1,[1,1][rm_1,rn_1]),\\&\quad\ldots,\\&\quad o(X,Y_m,[1,1][rm_n,rn_n])\end{aligned}$$

(2) Each set of generalization arcs can be declared as follows:

g(X,Y)	<=>	X : Y
g(X,Y1),...,g(X,Yn)	<=>	X : Y1,...,Yn
g(X1,Y),...,g(Xn,Y)	<=>	X1,...,Xn : Y

Example of an external specification:

```
PERSON(ssn[1,1], name, {surname}, age, ADDRESS).
ADDRESS(n°, street, code)
VEHICLE (number, colour, type, power).
CONTRACT( PERSON, VEHICLE, premium).
CLIENT : PERSON.
```

b) Specification of general constraints

The external interface to specify general constraints must allow the user to specify easily his integrity constraints defined over the external description of the data structures (i.e. previous data language). Each integrity constraint is specified as a production rule. The external language must have the same expressive power as the Morse formal language, but must be more concise and more easy to learn and to use. The external constraint language is built from the Morse formal language as follows:

(1) The alphabet of the external language is roughly the same as that of the internal language; except that "∧" and "->" symbols are respectively replaced by "and" and the two keywords "if" - "then" to distinguish between the left part and the right part of a given rule. The quantified variables ∀x et ∃x are respectively replaced by (x] and [x] to alleviate the absence of the mathematical symbols in common keyboards.

(2) The domain of interpretation of the external language is the same of that of Morse language: we distinguish names of atomic object classes (NA) and molecular object classes (NM), atomic and molecular object identifiers (respectively IA and IM) and the values of atomic objects (VA).

(3) The following restriction is made for variables: the scope of each defined variable is the set of instances of a specific class. We use the notation **x/class_name** to represent this declaration.

(4) The only allowed predicates are: $<, >, \leq, \geq, =, \neq$. These predicates apply only on atomic values.

(5) To access object identifiers and objects values through another object, the following functions are defined:

a) A value v_y of an atomic object y through another object x is delivered by the function "." defined as follows:
$I_X \times I_Y \longrightarrow V_Y$

$(x,y) \longrightarrow x.y=v_y / i(X,x) \wedge p(X,A) \wedge p(x,a_i) \wedge v(a_i,va_i)$

where I_X and V_Y are respectively the set of all identifiers of the class X and the class Y, and where V_Y is the set of all values of the class Y.

b) The access to an object identifier through another object is done by the function "->" defined as following:

$I_X \times N_C \longrightarrow I_O$
$(x,Y) \longrightarrow x\text{->}Y = i_y / i(Y, i_y) \wedge o(x,i_y)$

were I_X is the set of instances of X, N_C the set of all class names (i.e. NM+NA) and I_O the set of all object identifiers (i.e. IM+IA), and were x, Y, i_y be respectively elements of these categories.

To facilitate the rule expression, we introduce the following compositions of functions :
i) "x->y->z..." which is equivalent to:
"x->Y = y and y->Z = z and...".
ii) "x->y.z = v" which is equivalent to :
"x->Y = y and y.z = v".

(6) The only allowed terms are constant terms, variable terms and functional terms obtained by "." et "->" function symbols.

(8) The well-formed formulas are those elaborated with conjunction (and) and one implication (If...Then).

Example 1 : If a student has at least one mark less than 16, then his honors is not a first class.

```
IC2 : (student/Student) [mark/Mark] (honors/Honors)
      If student.mark<16
      Then student.honors ≠ "first class".
```

Example 2 : For each contract relating a person and a vehicle, if the age of the person is less than 20 and the power of the vehicle greater than 10, then the premium of the contract is at least equal to 5000.

```
IC3:(person/Person) (vehicle/Vehicle)
    (contract/Contract) (age/Age)
    (power/Power)(premium/Premium)

    If   contract->person.age<20
         and contract->vehicle.power>10
    Then contract.premium≥5000.
```

4.4. Code generation from integrity constraints

This subsection deals with O2 code generation from logical formulas describing integrity constraints. Before this generation process, a semantic control of each formula is done. Then we discuss the method definition and attachment, at the operational level.

a) Consistency checking of integrity constraints

The consistency checking of the constraints aims to verify in one hand the semantics of the constraints and in other hand their compatibility with the static database schema. It is composed of the following steps:
- Each constraint variable must be defined over an existing class of the static database schema,
- For each function symbol there must correspond an aggregation arc in the static semantic network,
- Arguments of the same predicates have compatible types,
- No predicate is subsumed by another predicate,
- Check wether different predicates of the same formula are contradictory or not, redondant or not,
- As we have not considered the exception handling, no constraint has to be contradictory with another one. We only check the consistency of the set of constraints, but some works could also be done on the problem of satisfiability of this set of constraints [Bry 86].

b) Methods definition and attachment

An integrity constraint is a first order formula specified on a semantic network. To give an interpretation to this formula (by assigning one of the logical values: true or false) with respect to the application universe of discourse represented in the database, we must generate one or several enforcement procedures depending on different kinds of updates expected for the database (insert, delete, modify). For example, from the following constraint which asserts a classical referential constraint,

```
RC:  (person/PERSON)
     (agency-name/Agency-name)
     [agency/AGENCY] [name/Name]
     person.agency-name = agency.name.
```

we may generate two enforcement procedures:
- one procedure M1 triggered by the insertion of a person (or the modification of his agency_name), which checks whether the referenced agency exists in the agency class or not,

- one procedure M2 triggered by the deletion of an agency (or the modification of its name), which checks whether referencing persons exist or not.

Then, we notice that from one constraint specification, we may generate different controle procedures, attached to different objects. We call each of these procedures a constraint-method. As the example shows, each constraint-method is attached to a specific class. A given constraint-method attachment is characterized by the following tuple: (Constr_Name, Class_name, Set_of_updates) where set of updates can be {insert, delete, modify,...}. Then an integrity constraint specification may be characterized by a set of attachments of this form. For example, the set of attachments characterizing the previous constraint RC is the following:

```
RC_A:    {(M1,PERSON,{Insert,Modify}),
          (M2,AGENCY,{Delete, Modify})}
```

The code generation of constraint-methods from a logical constraint specification needs the knowledge of:
1) Classes involved in the constraint specification (known through variable declaration), methods needed.
2) For each involved class, update operations which trigger the methods implementing the constraint.

Identifying involved classes

We first suppose that the distinction between identified objects and value-object (with respect to O2 concepts) is already done. Each class of identified objects involved in the constraint needs a method in order to check the constraint after an update done on an object of this class. We name the method by the concatenation of the name of the constraint and the name of the class. For example, from the constraint:

```
IC2: {employee/Employee}
     {probationer/Probationer}
     {salary-1/Salary} {salary-2/Salary}
employee.salary-1 > probationer.salary-2.
```

will generate two methods, IC2_employee, attached to the Employee class, IC2_probationer, attached to the probationer class. Even if the two methods are generated from the same constraint, the corresponding bodies are not the same. The method generated for the class Employe will check that the salary of an employee is greater than the salary of each probationer (it will be triggered after an update over employeee). The method generated for the class Probationer will check whether or not his salary is less that the salary of every employee (it will be triggered after an update over Probationer).

If several variables of a class appear in a constraint, one method is generated for each variable. For example, the constraint :

```
IC3: {employee1/Employee}, {employee2/Employee}
     {age1/Age} {age2/Age}
     {salary1/Salary} {salary2/Salary2}
If employee1.age1 > employee2.age2
Then employee1.salary1>employee2.salary2"
```

will generate two methods for the employee class :
IC3_employee1 : for one given employee if his age is greater than another employee's age, his salary is greater too,
IC3_employee2 : for one given employee, each time his age is lower than an other employee's age, his salary is lower too.

An optimization step will verify that the two generated methods on the same class are not similar. This happens when the constraint is perfectly symmetrical in relation to the two variables).
In the process described in this section, we have not considered the case where several different logical formulas may generate a unique constraint-method. We just focused on the case where a formula may generate one or several methods.

Identification of an Operation which Triggers the Verification of an Integrity Constraint

For each couple (method, class) previously built, we have to determine if it is concerned by a constraint on insertion, modification, or deletion. Our approach consists in considering on one hand, the type of quantifier applied to the variables of the class and on the other hand, the position of the predicate defined on these variables (left hand side (LHS) or right hand side (RHS) of the rule). In each case, we will show which operations (insertion, modification, or deletion) could induce a violation of an integrity constraint, and thus, require the enforcement of the constraint. This method is very close to those used for constraints simplification in the relational model [Nicolas 82] [Bry 88] [Quian 88] [Borla 90]. Because we operate at the design step and not during the step of integrity checking by the SGBD, we can't take into account the instances of the base, nor the queries, but only the syntaxe of the constraint.

Let us for example consider the case of a rule where variables are ranging over the class X. The rule is of the form : "($\exists$ x/X (P(x)) => Q". At any moment in time, for example, before an operation which will update the database, the assertion "($\exists$ x/X P(x)) => Q" is necessarily valid since it corresponds to an integrity constraint. The truth table presented in figure 6 shows that the formula "($\exists$ x/X P(x)) => Q" can be valid in three different states; these correspond to lines 1,3 and 4 of the table.

$\exists$ P(x)	Q	$\exists$ P(x) => Q
T	T	T
F	T	T
F	F	T

Fig. 6: Truth table for ($\exists$P(x)) => Q

Let us now consider the impact of an operation applied to an object of a class X over lines 1, 3 and 4 of the truth table. An operation on the object x of class X can change the truth value of the predicate ($\exists$x P(x)) (see figure 7). If the truth value of $\exists$x P(x) changes while it was in a state which corresponds to lines 1 or 3 of the table, the database remains in a coherent state (we only change from line 1 to line 3 or vice-versa). However, if the predicate was previously in a state which corresponds to line 4, and if the operation applied to an object x changes the truth value of the predicate $\exists$x P(x) from false to true, then the integrity is violated because the predicate assumes the truth value of line 2.

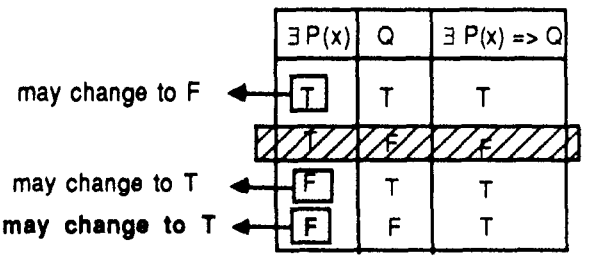

Fig.7 : Changes which may occur after an operation over an object of class X

We will now try to identify the operations which can, when applied to objects of class X, change the value of the predicate ∃x P(x) from false to true. This logical change may occur if before the operation, no instance of class X satisfied the predicate P, and after the operation, an object of X satisfies the predicate P. This can only happen under the two following operations:

- an object which satisfies the predicate P has been inserted into X,
- an object of class X which didn't satisfy P has been modified to a value which satisfies P.

We conclude by stating that the integrity constraint must be applied to class X in the case of an insertion or a modification of an object of the class X.

c) The generator of CO2 code

The generator of CO2 code is composed of a set of mapping rules which transform objects, relationships and integrity constraints of Morse toward objects and methods of O2. As each Morse object may satisfy one or several integrity constraints, each corresponding O2 object may satisfy one or several methods called "constraint-methods". These latter methods are particular in the sense they are activated by other methods which realize the encapsulation of the object. Then each update operation on a given object should activate by message passing the set of constraint-methods associated to this object. This set of constraints is called the "object-integrity". It can be itself considered as a general constraint-method associated to an object. Thus each update operation has to know only one general constraint-method instead of knowing the set of all specific constraint-methods.

The CO2 code generator is composed of two parts: (i) one part generates the definition of the object data structure and the constraint-methods signatures, (ii) the other part generates the body of the object-integrity method and the bodies of the corresponding specific constraints-methods. An example of code generation could be the following:

```
add class PERSON
type tuple(Name:string,Age:integer,
           Addr:setof(ADDRESS))
method ObjectIntegrity:set(string) is public
       NulleValue:boolean
       UniqueValue:boolean
       BoundedSet(min:int,max:int):boolean
       IC1Person:boolean
       IC2Person:boolean

body ObjectIntegrity:set(string)
     in class PERSON CO2
{ o2 set(string) ENS;
  SetRes = set();
  if (!([self NulleValue] ))
     {SET += set("NulleValue");};
  if (!([self UniqueValue] ))
     {SET += set("UniqueValue");};
  if (!( [self BoundedSet] ))
     {SET += set("BoundedSet");};
  if (!( [self IC1Person] ))
     {SET += set("IC1Person");};
  if (!( [self IC2Person] ))
     {SET += set("IC2Person");};
  return (SetRes);
}
```

In this example, "NulleValue", "UniqueValue" and "BoundedSet" come from the mapping of the MORSE structures. IC1Person and IC2Person comes from the mapping of the integrity constraints IC1 and IC2 explicitly given by the user as rules.

Each object-integrity method (i.e. the method ObjectIntegrity of the class Person in the previous example) returns a set type value. This set (e.g. SetRes in the previous example) contains the names of the constraint-methods which were not satisfied during the update operation. Depending on whether this set is empty or not, the programmer can commit or not the update operation. For example, the generator provides a new insertion method which inserts an object in the extension of its class. In the definition of this method, the corresponding "Object_Integrity" method must be activated to be sure that the update is allowed with respect to the integrity constraints :

```
add method insert:boolean
in class CO2 PERSON
body insert:boolean in class CO2 PERSON
{    o2 set(string) ENS
     SetRes = [self Integrity];
     if (SetRes==(o2 set(string)) set())
       {PERSON+=set(self);return(true);}
     else {printf("Integrity constraints
                   not respected: ");
           display (SetRes);
           return (false);}
}
```

In the previous code generation, the cost of the integrity checking process is not considered. Constraint-methods are specified in such a way they semantically correspond to the declarative assertions of the semantic network. The experience in traditional databases has shown that integrity checking is a very expensive process [Simon 84]. If we want to avoid the multiple scanning of the same class, we have to merge in the same procedure the different constraint-methods which have been defined for this class and don't modify the objects.

5. The Database Design Prototype

A large number of database design tools are based on semantic data models. Secsi is one among others [Bouzeghoub 85]. It was devoted to the design of relational databases. Starting from the Secsi Expert System environment, we have implemented a prototype for all the methodology previously developped. Much of the experience we got from this previous project (e.g. interactive acquisition of knowledge, completeness of specifications, consistency

checking) is reused in the new design tool prototype [Bouzeghoub 90].

During the analysis step, an aid is provided in order to refine incomplete specifications. To simplify the language, a few syntactic sugars can be introduced. The absence of a variable declaration, the usage of class names instead of class variables are examples. The control phase in the compilation process can be vued as a design specification phase where rules can be corrected by the system (with the help of the user) instead of brutally rejecting the rule. This phase occurs after the syntactic analysis of the constraint and before its semantic analysis.

Omitting variables : The user can omit variables. For example he can specify :

```
IC7 : If Person.Age < 20
      Then Person.salary < 1000.
```

In this case, the system must introduce variables and their corresponding quantifiers. By default, the quantifier is $\forall$ and its scope is the whole formula. The preceding formula is then translated into the following one:

```
IC7' : {person/Person} {age/Age} {salary/Salary}
       If person.age < 20
       Then person.salary < 1000.
```

This transformation supposes that the integrity constraint is mono-object. However, this must be confirmed by the designer.

Omitting links : We can envisage that the user may introduce integrity contraints on multiple objects without specifying the predicates which will serve as links between these objects, for example :

```
IC8 : If Person.Age<18 and Vehicle.Power>10.
      Then  Contract.Premium > 5000.
```

In this case, the links between objects are found in the static graph, and the integrity constraint can be rewritten as :

```
IC8': If Contract->Vehicle.Power > 10
         and Contract->Person.Age < 18
      Then Contract.Premium > 5000.
```

Many links may exist between two objects. For example, if a constraint holds on vehicles and persons, the link between them can either represent the fact that a person drives a car or the fact that a person owns a car. So the designer has to choose one of the reformulations proposed by the system.

Similarly, each time there is a lack in the static schema, it may be completed to be consistent with the integrity constraints. For example, the constraint "Person.Age>0" may create in the static semantic network an object 'person' which is composed by an object 'age' whose type is 'integer'.

The aid provided by the design tool is glaring at sight of the final results. The previous constraint, IC8, can be expressed with two lines by the user (because of the refinement made by the tool and because of the conciseness of the rule langage). The generated code is about twenty lines (because of the procedural aspect of the langage used). One object type specified by one line including cardinality constraints, may generate two pages in CO2 code because some cardinalities have to be expressed by methods, like any integrity constraint. In addition to the class declarations and the methods describing the integrity constraints, the tool generates other useful methods necessary for the manipulation of the objects. For example, we need a method called "ObjectIntegrity" which groups all the integrity constraints concerning a given object. This global method is called by any other operation defined on the object class concerned (e.g. an insertion operation creates an object instance, adds it into the extension of the corresponding class and then calls the "ObjectIntegrity" method which insure the integrity defined on the class.

6. Concluding Remarks and Current Extensions

In this paper, we have described a general framework for a CASE tool devoted to the design of object oriented databases. The design approach is based on two levels: the semantic object oriented level and the operational object oriented level. The first level is based on a semantic data model which was extended to represent more information about the behavior of objects (general integrity constraints and deductive rules). The second level is more operational, and is based on an existing object oriented DBMS. The design methodology described in this paper is implemented in the Secsi Expert system environment which already provides a design environment for relational databases.

This design tool is interfaced with O2 object oriented database system. It automatically generates a CO2 database schema and gives a very convenient way to populate the database and to check its constistency with respect to the constraint-methods generated. A syntactic analysis of specifications, an interactive acquisition aid of constraints and rules and a set of consistency checking rules are provided too. This design environment can be considered as a powerfull mean for validating user requirements against an image of the projected database application.

An extension concerning behavioral rules is in progress. We call "behavioral rule" an assertion which has in its right hand side some updates to be performed on the database; the left hand side has the same syntax and the same semantic than a constraint rule's left hand side. Further developments should mainly concern the formalization of the external language. Besides this formalization aspect, many other problems remain, like the efficiency of the code generation procedure and the code optimization.

References

[Bancilhon 87] BANCILHON F. "Les objectifs scientifiques du GIP Altaïr" , Rapport Altaïr 8/1987.

[Bernstein 82] BERNSTEIN P. & BLAUSTEIN B. "Fast Method for Testing Quantified Relational Calculus Assertions" ACM-SIGMOD Conf., Colorado, June 1982.

[Bertino 84] BERTINO E. & APUZZO D. "Integrity aspects in Database Management Systems" Proceed. of Internat. Conf. on Trends and Applications of Databases" IEEE-NBS, Gaithersburg, USA 1984.

[Borla 90] BORLA-SALAMET P. "Le contrôle de l'integrité Sémantique des bases de données Relationnelles et déductives" Phd thesis Paris VI, 1990.

[Bouzeghoub 84] BOUZEGHOUB M. "MORSE: a Functional Query Language and its Semantic Data Model" Proceed. of Internat. Conf. on Trends and Applications of Databases" IEEE-NBS, Gaithersburg, USA 1984.

[Bouzeghoub 85] BOUZEGHOUB M., GARDARIN G. & METAIS E. "SECSI: An Expert System for Database Design" 11th VLDB Conf., Stockholm Sweeden 1985.

[Brodie 81] BRODIE M. "On Modelling Behavioural Semantics of Databases" 7th VLDB Conf., Cannes, France 1981.

[Brodie 84] BRODIE M., MYLOPOULOS J., SCHMIDT Y. "On Conceptual Modelling: Perspectives from Artificial Intelligence, Data Bases and Programming languages" Springer-Verlag, NY 1984.

[Brodie 86] BRODIE M. & MYLOPOULOS J. "On Knowledge Base Management Systems" (editors) Springer Verlag, 1986.

[Bry 86] BRY F., MANTHEY R. "Checking Consistency od Database Constraints : a Logical Basis", 12th VLDB Conf, Kyoto, Japan, 1986.

[Bry 88] BRY F., DECKER H. "Preserver l'integrité d'une base de données déductive : unee méthode et son implémentation", 4ème journées BD3, Bénodet, France, 1988.

[Buchmann 86] BUCHMANN A.P., CARRERA R.S. & VASQUEZ-GALINDO M.A. "A Generalized Contraint and Exception Handler for an Object Oriented CAD-DBMS", in [OODB 86].

[Gustafsson 83] GUSTAFSSON M.R., BUBENKO J.A. & KARLSSON T. "A declarative Approach to conceptual information modelling" in OLLE,SOL,VERRIJN-STUART (eds): Information System Methodology: a comparative approach, North Holland Publ. Co 1983.

[Hagelstein 88] HAGELSTEIN T. "A declarative Approach to information system requirements" J. Knowledge Based Systems, 1(4) 1988.

[Hammer 81] HAMMER M.M. & McLEOD D.J., "Database Description with SDM: A Semantic Data Model" ACM TODS Vol6,N°3, Sept 1981.

[Haux 88] HAUX L. , C. LECLUSE, RICHARD P. "The CO2 V0.4 Language and Some Extensions, Release 3.1" Rapport interne Altaïr 4-88, octobre 1988.

[Lecluse 87] LECLUSE C., RICHARD P. & VELEZ F."An Object Oriented Data Model" C. Rapport Altaïr 10/ 1987.

[Lecluse 88] LECLUSE C,RICHARD P. & VELEZ F., "Modeling Inheritance and Genericity in Object Oriented Databases, Version 1" Rapport Altaïr 18/ 1988.

[Loucopoulos 89] LOUCOPOULOS O. & KARAKOS-TAS V. "Modelling and validating office information systems: an object and logic oriented approach" Software Engineering Journal, March 1989.

[Morgenstern 84] MORGENSTERN M. "Constraint Equations : Declarative Expression of Constraints with Automatic Enforcement". 1Oth VLDB Conference, Singapore, 1984.

[Nicolas 82] NICOLAS J.M. " Logic for Improving Integrity Checking in Relational Databases" Acta Informatica, July 1982.

[Olivé 89] OLIVE A. "On the design and implementation of information systems from deductive conceptual models" 15 th VLDB Conference, Amsterdam, 1989.

[OODB 86] Object-Oriented Databases, Proceed. of the 1st Internat. Workshop, IEEE Computer Society Press 1986.

[Quian 88] QUIAN X. "An effective Method for Integrity Constraint Simplification" 4th Conf. on Data Engineering, Los Angeles, California, 1988.

[Simon 84] SIMON E., VALDURIEZ V. "Design and Implementation of an Extendible Integrity Subsystem" Proceed. of ACM SIGMOD Internat Conf., Boston, USA, 1984.

[Smith 77] SMITH J.M. & SMITH D.C.P., "Database Abstractions: Aggregation and Generalization" ACM TODS June 1977.

[Tsalgatidou 90] TSALGATIDOU A., KARAKOSTAS V. & LOUCOPOULOS P. "Rule-Based Requirements Specification and Validation" Proceed. of the 2nd Nordic Conf. on Advanced Information System Engineering, Kista CAiSE90, in Lecture Notes in Computer Sciences (436), Steinholtz-Solvberg-Bergman (Eds), Springer-Verlag May 1990.

[Tucherman 85] TUCHERMAN L., FURTADO A. & Casanova M.A. "A Tool for Modular Database Design" 11th VLDB Conf, Stockholm , Sweeden 1985.