A Design Tool for Object Oriented Databases

Mokrane Bouzeghoub, Elisabeth Métais

Laboratoire MASI, Université P. et M. CurieCentre de Versailles,
45 avenue des Etats-Unis78000 Versailles

Farid Hazi, Laurent Leborgne

Infosys, 15, Rue Anatole France 92800 Puteaux La Défense

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 behaviour of objects. But this latter capability is generally provided through an algorithmic language which cannot be considered as a conceptual language. In this paper, we combine the two categories of data models and give a procedure on how to translate the conceptual model to the logical model.

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 behaviour 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.

This work was partly supported by GIP/ALTAIR, PRC/BD3 and INFOSYS

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 behaviour 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 behaviour 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 behaviour 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.

A large number of database design tools are based on semantic models. Secsi is one among others [BOUZ 85][BOUZ 86]. Many experience we got from this previous project Secsi (e.g. interactive acquisition of knowledge, completeness of specifications, consistency checking) is reused in the new domain of object orientation.

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. 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 model the O_2 model, which was developed by Altaïr project [BANC 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). A design tool prototype based on this approach has been developed and demonstrated [BOUZ 89].

Secsi is a database design assistant which is based on two models: a semantic data model, called Morse, and the relational data model. The semantic data model is built upon the usual concepts of aggregation and generalization which were refined in more basic constructors: aggregation of atomic objects, aggregation of molecular objects, classification of objects and generalization of classes. To enhance the semantic power of this model, several constraints are defined: domains, keys, cardinalities, dependencies, intersections and disjunctions.

2. The Semantic Data Model

A semantic netwok 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 [BOUZ 84]. In the following, such a semantic network data model is designated by the name Morse. The following subsections detail the different concepts used in Morse.

2.1. The objects of the model

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 molécular 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. In traditional databases, classes of atomic objects are practically never used; files containing only one field (or one column relation) is generally considered as irrelevant to the application. Database operations (retrieve, insert, delete) are generally defined over classes of molecular objects (file, relation). Then, in the classical data models, atomic objects exist only as properties or values to characterize molecular objects. In semantic data models and in object oriented data models, atomic objects can exist (and then be identified) independently of the molecular objects to which they are related.

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.

2.2. The semantic links

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* [SMIT 77]. Specific refinement 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* and *molecular aggregation.* Generalization is refined as *instance generalisation* and *class generalization.*

a) The atomic aggregation *(or aggregation of atomic objects)* permits the construction of a new molecular object X by juxtaposition of a sequence of atomic objects $A_1 \ldots A_n$ which are generally, but not necessarily, of different domains. The molecular object is related to each of its atomic components by a couple of binary arcs $\mathbf{a(A_i,X)}$ and $\mathbf{p(X,A_i)}$ which represent the reverse directions of the same binary relationship relating a molecular object to its atomic component.

```
a(Number, VEHICLE),                    p(VEHICLE,Number),
a(Type, VEHICLE),            or        p(VEHICLE,Type),
a(Power, VEHICLE),                     p(VEHICLE,Power),
a(Color, VEHICLE).                     p(VEHICLE,Color).
```

b) The molecular aggregation *(or aggregation of molecular objects)* permits the construction of a new molecular object Y by juxtaposition of a sequence of other molecular objects $X_1 \ldots X_n$. Each semantic relationship is represented by a couple of binary arcs $\mathbf{r(X_i,Y)}$ and $\mathbf{o(Y,X_i)}$ which represent the reverse directions of the same binary relationship which relates two molecular objects.

```
r(VEHICLE,ORDER),            or        o(ORDER,VEHICLE),
r(CLIENT,ORDER).                       o(ORDER,CLIENT).
```

c) The instance generalization *(or generalization of instances)* is often called **classification.** It permits to build a new class of object instances X by union of other object instances $O_1 \ldots O_n$. It's a way to define a class by extension. As the amount of objects in a given class could be very high, this abstraction is not often used in database schemas; then all schema objects are considered as classes defined intensionaly by their basic domains or their aggregations. This kind of abstraction is represented in the semantic network by the pair of arcs $\mathbf{c(O_i,X)}$ and $\mathbf{i(X,O_i)}$ (for classification/instanciation) which represent the reverse directions of the same binary relationship.

```
c(C1,Color)                            i(Color,C1)
c(C2,Color)                  or        i(Color,C2)
c(C3,Color)                            i(Color,C3)

c(v1, VEHICLE)                                 i(VEHICLE, v1)
.....                                or        .....
c(vn, VEHICLE)                                 i(VEHICLE, vn)
```

d) The class generalization *(or generalization of classes)* permits to build a new class of objects X as a union of other classes $X_1 \ldots X_n$ by concentrating only on their common properties (components). This kind of abstraction is represented by a pair of arcs $\mathbf{g(X_i,X)}$ and $\mathbf{s(X,X_i)}$ (generalization/specialization) which represent the reverse directions of the same binary relationship. These arcs allow to build hierarchies of classes.

```
g(CLIENT, PERSON),                    s(PERSON,CLIENT),
g(AGENT,CLIENT),                      s(CLIENT,AGENT),
g(PRIVATE_PERS,CLIENT),               s(CLIENT,PRIVATE_PERS).
```

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.

As for all abstractions there are two equivalent representations (equivalent reversed arcs), we use only one specification which subsumes the other (for example p, o, c and g) except if constraint specification is needed for the implicit arc.

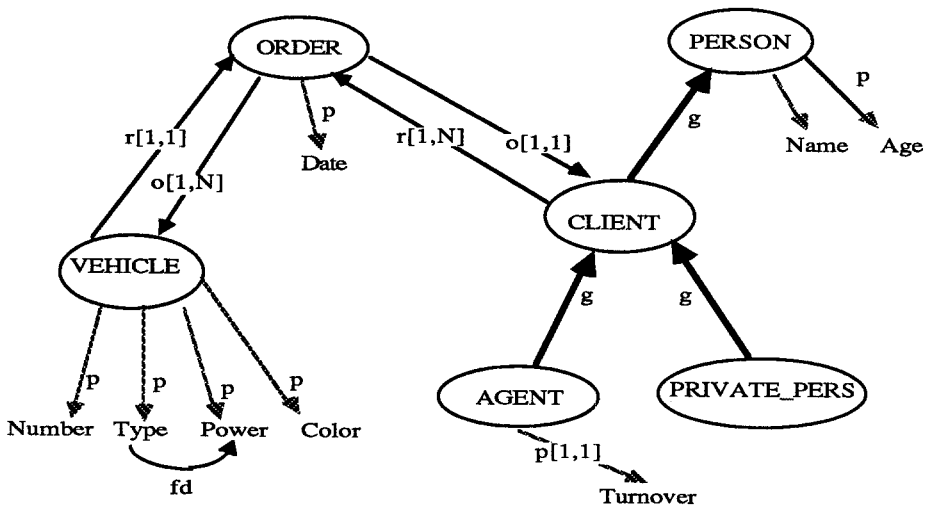

Fig.1: An example of a semantic network

2.3. Integrity constraints

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]),          and        p(VEHICLE,Power,[1,1]),
a(Color,VEHICLE,[0,N]).                     p(VEHICLE,Color,[1,3]).

r(VEHICLE,ORDER,[0,1]),          and        o(ORDER,VEHICLE,[1,10]),
r(CLIENT,ORDER,[1,N]).                      o(ORDER,CLIENT,[1,1]).

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

Graphically, a given semantic network can be represented as portrayed in figure 1.

3. The Object Oriented Data Model

The O_2 data model belongs to the category of the so-called object oriented data models [LECL 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 O_2 expressions (called CO_2) [HAUX 88] or an SQL like declarative query language (called LOOQ).

3.1. Objects and types

In the O_2 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 tuple ([...]) and set ({...}) constructors. For example:

(i_5, [name:"John", age:22, v:i_6]) is an object having a tuple value,
(i_6, {v_1,v_2,v_3}) is an object having a set value.

Objects can mutually reference each other. For example,

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

Intuitively a *class* is a mean for representing a set of objects with their behaviour. A class is composed of two parts: (i) a type which contains the structure that characterises 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 *tuple* and *set* constructors could be composed to create more elaborated types (e.g. sets of tuples or tuples of sets). For example:

```
Person =[name:string,age:integer,vehicles:{[number:integer,color:string]}]
```

The O_2 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 specification , 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]]
```

The object identity makes possible the sharing of objects. For example:

```
Person = [name:string, age:integer, vehicles:Vehicles]
Vehicles = {[number:integer, color:string]}
Garage = [code:string, address:string, vehicles:Vehicles]
```

where Person and Garage may share same objets of the class Vehicles. 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 [LECL 88].

A method is a procedure which is associated to a type in order to describe the behaviour 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.

3.2. Programming in CO_2

The CO_2 language is an embedded database language (CO_2) 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 CO_2:

```
add class   Person with extension                    /* type declaration */
 type tuple(name:string,age:integer,address:string,children:set(Person))

add class Agent inherits Person                      /* hierarchy of classes */
  type tuple (code:string, salary:integer)

  method category: string is public                  /* method declaration */

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

The keyword **inherits** defines a hierarchy of types. The keyword **with extension** creates a named value which contains the instances of the class and then permits set operations on this class. The keywords **type** and **method** respectively define the object data structure and its associated method signature. Its following keyword **is public** makes the object-integrity method visible from anywhere. The keyword **body** introduces the procedure which implements the method. Its following 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 CO_2 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.

4. Mapping from the Semantic Level to the Operational Level

The CO_2 model describes both static aspect (data structures) and 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 behaviour 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 CO_2 by different syntactic forms.

The semantic data model Morse concerns only 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. Objects identity is explicitely handled only for molecular objects. Indeed, in the traditional databases, we make a strict

dichotomy between molecular objects which are generally identified with an associative manner using attributes, and the atomic objects which exist only as attribute values for molecular objects.

In the following we are only interested on the mapping from Morse to CO_2 and not for the reverse mapping. First we consider the structural mappings between the two models, then we study the representation of constraints with methods, and finally 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 [BOUZ 88].

4.1. The mapping between objects

An atomic object defined in Morse is equivalent in O_2 to either an identified atomic object or to an atomic value (non identified object) inside an other object. A molecular object defined in Morse is equivalent to either an identified tuple structured object or to a tuple value in O_2. A class of objects defined in Morse is partly equivalent to a class of objects defined in O_2. Indeed, and as stated before, Morse classes describe only the static aspect of the objects, while O_2 classes describe their behaviour too, thanks to methods. Figure 2 summerizes the correspondance between the Morse objects and the O_2 objects.

MORSE CONCEPTS	O2 CONCEPTS
Atomic object	Atomic object / atomic value
Molecular object	Structured object / tuple value
Class	Class
Subclass	Subclass
Instance	Object
Object identifier	Object identifier

Fig. 2: Correspondance between the Morse objects and the O2 objects

4.2. The mapping between constructors

Both atomic and molecular aggregation defined in Morse are equivalent to the tuple constructor of the O_2 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 O_2. 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(Postcode,integer)
```

will be mapped into O_2 as:

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

which can be described in CO_2 by the following statements:

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

add class Address
    type tuple(Number:integer,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 an other object, i.e. Address.

The classification/instanciation defined in Morse is partly equivalent to an O2 class defined **with extension**. 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 O_2. 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 refinement on its structure. This makes the generalization/specialization more general than a partial order of types which is defined in O_2.

MORSE CONCEPTS	O2 CONCEPTS
Atomic aggregation	Tuple constructor
Molecular aggregation	Tuple constructor
Classification / Instanciation	Class defined by extension
Generalization / Specialization	Inheritance hierarchy

Fig. 3: Mapping between the Morse and the O_2 constructors

The inheritance is defined in Morse as a logical property which propagates components and constraints of generic classes to their subclasses. In the O_2 model, there is a uniform formalisation of hierarchies of types and inheritance (partial order of types). Figure 3 summerizes the different mappings between the Morse constructors and the O_2 constructors.

4.3. 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 O_2 object. Except for the usual domains which are represented by basic types in O_2 (integer, real, boolean, string), all the other Morse integrity constraints are represented by methods in the O_2 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. Figure 4 summerizes the different mappings between the Morse constraints and the O_2 concepts.

MORSE CONCEPTS	O2 CONCEPTS
Domain	basic type / method
Cardinality	set constructor + method
functional dependency	method
key	method
intersection / disjonction	method

Fig. 4: Mappings between the Morse constraints and the O2 concepts

a) Methods implementing cardinality constraints:

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 by which we will implement these constraints. As we have

several situations, we will only focus on three 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 O_2 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 integer) 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 O_2 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:integer, max:integer):boolean
     in class PERSON CO2
     {     O2 set(Address) x;
           x = (self->Addr);
           if ((min =< count(x)) && (count(x) =< max))
           {return (true);}  else return (false);
     }
```

Case 3: Unique value (key): If we specify cardinalities for the reverse arcs of the semantic network, we obtain other kind of constraints like unique values or keys:

```
a(Name, PERSONNE,[1,1]).
```

This constraint can be represented into CO_2 as follows:

```
add class PERSON     with extension
  type tuple (Name:integer)
  method  Unique_value:boolean

  body Unique_value:boolean in class PERSON CO2
     {     O2 PERSON p;
           integer RES;
           RES = 1;
           for (p  in  PERSON  when p->Name == self->Name)
           {RES = 0};
           if (RES == 1) {return (true);} else return (false);
     }
```

b) Methods implementing functional dependencies

In the relational data model, functional dependencies are used to represent elementary facts between attributes, and then serve as a basis for the normalisation process. In the Morse semantic data model, functional dependencies are just considered as constraints between atomic objects within a molecular object. In an object oriented data model, these constraints can be implemented as methods checking the consistency of the object values. For example,

```
p(VEHICLE,Number)            df(VEHICLE,lhs(Type),rhs(Power))
p(VEHICLE,Type)
p(VEHICLE,Power)
```

can be implemented as following in O_2:

```
add class VEHICLE   with extension
            type tuple(Number:integer,Type:string,Power:integer)
            method Funct_dependency : boolean
            body Funct_dependency: boolean in class  VEHICLE CO2
               {O2 VEHICLE v;
                integer RES;
                RES = 1;
                for (v in VEHICLE when (strcmp (self->Type, v->Type))
                                       && !(self->Power == v->Power))
                      {RES = 0});
                if (res == 1) {return (true);} else return (false);
                }
```

We shall see later that they can be used in the similar way of the relational model to what can be considered as object definition.

4.4. The object identity and the object sharing

In the Morse semantic data model, everything is considered as an object. Each object has a unique representation, then objects can be shared between different other related objects. In the O_2 object oriented data model, there are objects and values; objects are sharable while values are not. So,

when mapping a Morse schema into an O_2 schema, we have to decide whether a Morse object can be considered as an O_2 object or as an O_2 value. This decision mainly depends on the user's desire in the way to implement his database. He can arbitrarily decide whether a given Morse object is an O_2 object or value. For example, for the mapping of the following Morse schema (figure 5), he can envision many solutions:

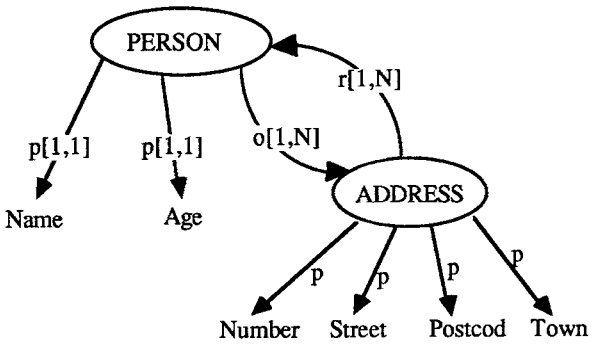

Fig.5: Morse objects

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

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

In this case each person's addr is a reference to a set of addresses.

Solution 3: Three O2 objects whose one is an identified set:

```
PERSONNE=[Name:string, Age:integer, addr:ADDRESSES]
ADDRESSES: {ADDRESS}
ADDRESS=[Name:integer, Street:string, Postcod:integer, Town:string]
```

There are two kinds of objects describing addresses: the first one (ADDRESSES) describes sets of addresses, the second one (ADDRESS) describes tuples whose each corresponds to a given address.

Solution 4: 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.

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 from the following parameters:

- *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.
- *Functional dependencies* defined between atomic objects of the semantic network: as in the relational model, a set of functional dependencies can determine a group of Morse atomic objects which may correspond to an O_2 object. In this case, we can just highlight these groups but the final decision remains to the human designer.
- *Keys* defined for molecular objects: generally keys are used to provide an associative access to objects. It is generally considered as an external way of identifiing objects.
- *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.

4.5. The generator of the CO_2 code

The generator of CO_2 code is composed of a set of mapping rules which transform objects, relationships and constraints of Morse toward objects and methods of O_2. As each Morse object may satisfy one or several integrity constraints, each corresponding O_2 object or value 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 CO_2 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. The two parts consist of dynamicaly filling a predefined frame which is organized into slots containing

keywords of the CO_2 language (figure 6).

add class	class_name_1 [**inherits** class_2] **with extension**	
	type	(component_1 : type_1, component_2 : type_2, ... component_n : type_n)
	method	method_signature_1 ... . method_signature_n
	body	signature_method_1 **in class** class_name_1 CO2 method_body_1
	body	signature_method_n **in class** class_name_1 CO2 method_body_n

Fig.6: The code generator frame

An example of code generation could be the following:

```
add class PERSON
   type tuple(Name:string,Age:integer,Addr:setof(ADDRESS))
        with extension
   method  Integrity:set(string) is public
           Nulle_value:boolean
           Unique_value:boolean
           Bounded_set(min:integer, max:integer) : boolean

   body Integrity:set(string) in class PERSON CO2
        {  o2 set(string) ENS;
           SetRes = set();
           if (!( [self  Nulle_value] )) {SET += set("Nulle_value");};
           if (!([self Unique_value] )) {SET += set("Unique_value");};
           if (!( [self  Bounded_set] )) {SET += set("Bounded_set");};
           return (SetRes);
        }
```

Each object-integrity method (i.e. the method Integrity 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 wether this set is empty or not, the programmer can commit or not the update operation. For example, we define a new insertion method which creates an object and assigns a value to each of its attributes. In the definition of this method, we must activate the corresponding integrity constraint

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 expansive process. 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. This problem is not addressed in this paper.

5. Extending the Semantic Data Model to Represent General Constraints

This section intends to extend the Morse semantic network to represent more generalized integrity constraints. These general integrity constraints should be any first order logic formula whose variables refer to the content of the semantic database. Before presenting this extension, 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 represent real databases but just to give a formal abstract representation in order to correctly specify integrity constraints.

5.1. 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,P_name,string)          p(PERSON,Name,[1,1][1,N])
i(NA,Age,integer)            p(PERSON,Age,[1,1][1,N])
i(NA,number,integer)         p(VEHICLE,Power,[1,1][0,N])
i(NA,Power,integer)          p(CONTRACT,Premium,[1,1][1,1])

i(NM,PERSON)                 o(CONTRACT,PERSON,[1,1][1,N])
i(NM,VEHICLE)                o(CONTRACT,VEHICLE,[1,1][1,1])
i(NM,CONTRACT)               g(CLIENT, PERSON)
```

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. In the following is an example of 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(PERSON,P3)

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)

p(P1,N1)                        p(V1,I1)                      o(C1,P1)
p(P1,A1)                        p(V1,W1)                      o(C1,V1)
                                                              p(C1,M1)
p(P2,N2)                        p(V2,I2)                      o(C2,P1)
p(P2,A2)                        p(V2,W1)                      o(C2,V2)
                                                              p(C2,M2)
p(P3,N2)                        p(V3,I3)
p(P3,A2)                        p(V3,W2)
```

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.

5.2. The representation of general integrity constraints

A general integrity constraint is a first order closed formula, restricted to only conjunction connectors and at most only one implication symbole. The following expressions are allowed

constraints: P(X), P(X) & Q(Y), P(X) —> Q(Y), P(X) & Q(Y) —> R(X,Y). 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), and (iii) usual mathematic predicates: $<, >, \leq, \geq, =, \neq$.

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: $\forall P\ \forall C\ \forall V\ \forall G\ \forall S\ \forall M\ \exists VG\ \exists VS\ \exists VM$
$[i(PERSON,P) \wedge i(VEHICLE,V) \wedge i(CONTRACT,C)$
$\wedge i(Age,G) \wedge i(Power,S) \wedge i(Premium,M)$
$\wedge o(C,P) \wedge o(C,V)$
$\wedge p(P,G) \wedge v(G,VG) \wedge VG<20$
$\wedge p(V,S) \wedge v(S,VS) \wedge VS>10]$
$\rightarrow [p(C,M) \wedge v(M,VM) \wedge VM \geq 5000].$

We can also state that the age of every person is greater than 17.

IC2: $\forall P\ \exists G\ \exists VG\ \ i(Person,P) \wedge i(Age,G) \wedge p(P,G) \wedge v(G,VG) \wedge VG>17$

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,$\forall$) 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,$\forall$,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,right,IC1)         v(M,VM,right,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 right hand side part; although some nodes appear in the both parts.

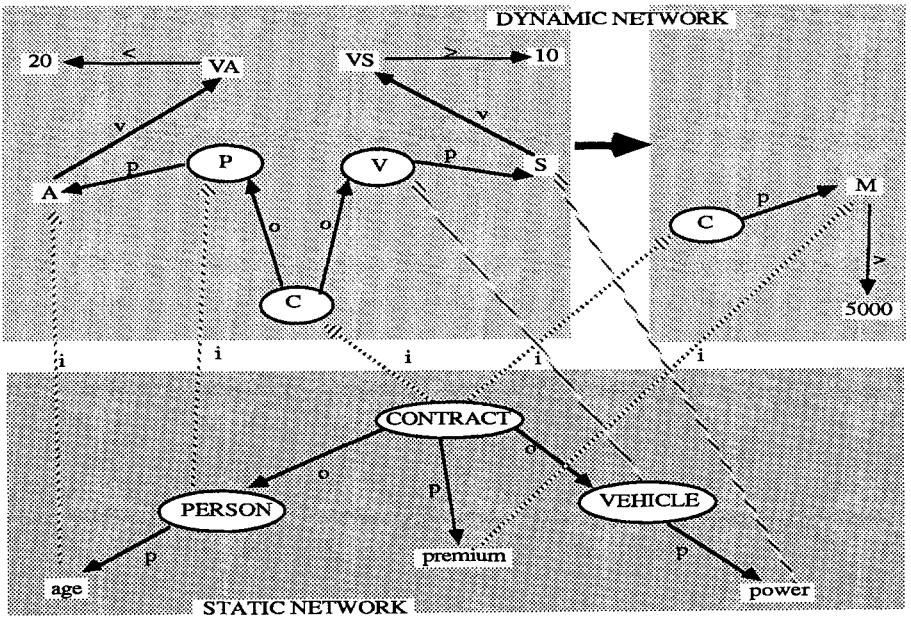

Fig.7: An exemple of rule representation

As for the database extension, this representation is an abstract representation for a better understanding and consistency checking of integrity constraints.

5.3. The semantic object oriented language

The Morse language is a formal language to represent the detailed description of a conceptual schema. This language is not intended to be used by end-users nor to implement real databases. Consequently, we need two things: a friendly user interface to specify data structures and constraints, and a real database managment system to implement the database schema. This subsection describes the former requirement, the latter one is supported by the O_2 system after mapping Morse specifications into O_2 specifications.

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:

```
X(A1:dom1,...,An:domn)
<=> i(NA,A1,dom1), ...,i(NA,An,domn), i(NM,X)
    p(X,A1), ..., p(X,An)
```

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

```
X(Y1,...,Yn)
<=> o(X,Y1),...,  o(X,Yn), i(NM,X)
```

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

```
X(A1:dom1,...,An:domn,Y1,...,Ym)
<=> i(NA,A1,dom1), ...,i(NA,An,domn), i(NM,X)
    p(X,A1), ..., p(X,An)
    o(X,Y1),...,  o(X,Yn)
```

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

```
X({A1:dom1}[am1,an1],
                ...,
                An:domn,[amn,ann],
                {Y1},[rm1,rn1],
                ...,
                Ym,[rmn,rnn])
<=> i(NA,A1,dom1), ...,i(NA,An,domn), i(NM,X)
    p(X,A1,[1,N][am1,an1]),
      ...,
    p(X,An,[1,N][amn,ann]),
    o(X,Y1,[1,1][rm1,rn1]),
      ...,
    o(X,Ym,[1,1][rmn,rnn])
```

(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
```

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 fist order assertion or a production rule. This specification must be made at any place in the application description. 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 "$\wedge$" 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 $\forall x$ et $\exists x$ 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 value of an atomic object is delivered by the function "." defined as a composition of two elementary functions f1 and f2 defined as follows: let I_X be the set of instances of X, P_X the set of atomic components of X, IA the set of all atomic identifiers and VA the set of all atomic values, and let x, at, a_i, va_i be respectively elements of the previous categories:

f1: I_X x P_X —> IA

(x,at) ——> x.at = a_i / i(at, a_i) $\wedge$ p(x,a_i)

➡ x.at = f2(f1(x,at))

f2: IA —> VA

a_i ——> va_i / v(a_i,va_i)

(5) The access to a a molecular object through another molecular object is made by the function "–>" which delivers only the molecular object identifier. This function is defined as following: let I_X be the set of instances of X, O_X the set of molecular components of X, and IM the set of all molecular object identifiers, and let x, mol, m_i be respectively elements of these categories.

I_X x O_X —> IM

(x,mol) ——> x->mol = m_i / i(mol, m_i) ∧ o(x,m_i)

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

(7) The only allowed predicates are: $<, >, \leq, \geq, =, \neq$.

(8) The well-formed formulas are those of the first order predicate calculus, elaborated with the conjunction (and) and the implication (If...Then).

Example 1 : The salary of any employee is les than that of all managers.

IC1: `{m/Manager} {e/Employee} e.salary < m.salary.`

Example 2 : Each student's mark is between 0 and 20.

IC2: `{s/Student} {m/mark} s.m≥0 and s.m≤20.`

Example 3 : If a student has at least one mark less than 16, then his honors is not a first class.

IC3: `{s/Student} [m/mark] If s.m<16 Then s.honors ≠ "first class".`

Example 4 :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.

```
IC4: {p/Person} {v/Véhicle} {c/Contract}
     If c->Person=p and c->Vehicle=v and p.age<20 et v.power>10
     Then c.premium≥5000.
```

To facilitate the rule expression, we can introduce the following composition of functions:

```
If x->y->z   Then .....
```

which is equivalent to:

```
If x->Y = y and y->Z = z   Then .....
```

In the same way, the following:

```
If x->y.Z = 'v'   Then .....
```

composition is equivalent to:

```
If x->Y = y and y.Z = 'v'   Then .....
```

With these compositions, the example 4 can be writen more simply as follows:

```
IC4:  {p/Person} {v/Véhicle} {c/Contract}
      If c->p.age<20 and c->v.power>10   Then c.premium≥5000.
```

5.4. Code generation from general integrity constraints

This subsection deals with O_2 code generation from logical formulas describing integrity constraints. 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 focus on the case where a formula may generate one or several methods. Before this generation process, a semantic controle of each formula is done. Then we discuss the method definition and attachment.

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,
- As we have not considered the exception handling, no constraint has to be contradictory with another one.

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 the application universe of discourse represented in a database, we must generate one or several enforcement procedures depending on different kinds of updates envisioned for the database (insert, delete, modify). For example, from the following constraint expression which asserts a classical referential constraint,

```
RC: {p/PERSON} [a/AGENCY]
    If p.Agency_name = "n"   Then  a.name = "n".
```

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: (`Const_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 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) object classes involved in the constraint specification (known through variable declaration),
2) for each involved class, update operations which trigger this constraint (given by the end user or generated from buiseness rules).

Having this knowledge, the process of generating a CO_2 procedure from a logical formula is quiet simple. The same recipient frame described in the previous section is instanciated to generate CO_2 methods.

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 behaviour ob objects (general integrity constraints and deduction rules). The second level is more operational, and is based on an existing object oriented DBMS called O2. 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 O_2 object oriented database system. It automatically generates a CO_2 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 a set of consistency checking rules are also provided too. This design environment can be considered as a powerfull mean for validating user requirements against an image of the projected database application.

The new development mainly concerns the semantic checking of general integrity rules, a decision procedure for object identification, and a more efficient code generation procedure. The current work extends the system to aid in the acquisition and represention of buiseness rules from which it may generate the complete behaviour of a given database. Buiseness rules are expressed as a generalization of integrity rules, and then represented by production rules having in their right hand side database operations.

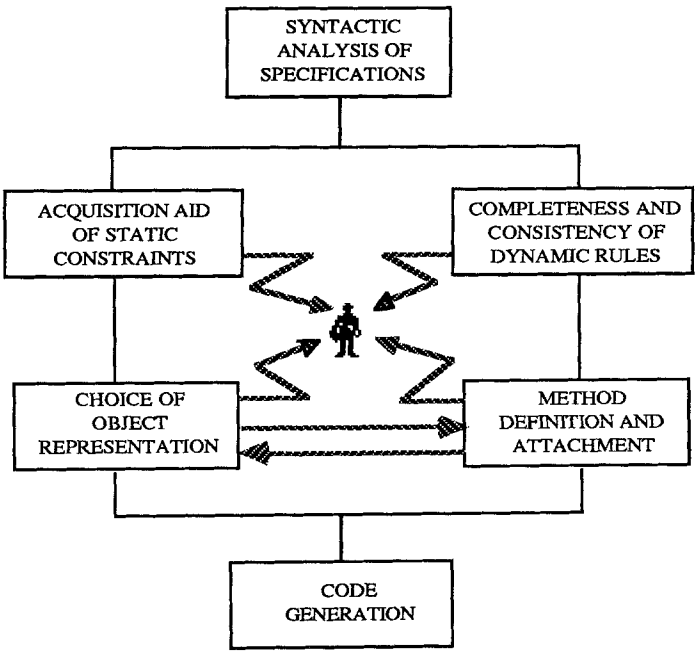

Fig.8: The architecture of the design tool

REFERENCES

[BANC 85] BANCILHON F., KIM W. & KORTH H.F., "A Model of CAD Transactions" 11th VLDB Conf., Stockholm 1985.

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

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

[BERT 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.

[BORG 85a] BORGIDA A. "A Language Features for Flexible Handling of Exceptions in Information Systems" ACM TODS Vol10, N°4, Dec. 1985.

[BORG 85b] BORGIDA A. "Accomodating Exceptions in Databases, and Refining the Schema by Learning from Them" 11th VLDB Conf., Stockholm, Sweeden 1985.

[BOUZ 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.

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

[BOUZ 86] BOUZEGHOUB M. "SECSI: un système expert en conception de systèmes d'information", Thèse de doctorat de l'université P. et M. Curie, mars 1986.

[BOUZ 88] BOUZEGHOUB M., METAIS E , MARAUX F., HAZI F., " Transformation du modèle MORSE en modèle O_2", Rapport de spécification, tâche 2 phase 1, Infosys-Masi-Altaïr, décembre 1988.

[BOUZ 89] BOUZEGHOUB M., METAIS E, MARAUX F., HAZI F., "Conception d'une base de données orientée objets à l'aide d'un modèle sémantique" Journées Bases de données avancées, PRC/BD3, Genève septembre 1989., Rapport MASI N°307, Univ. Paris VI Nov 1989.

[BOUZ 89] BOUZEGHOUB M., METAIS E, F., HAZI F., LEBORGNE L. "Aide à la spécification de l'intégrité sémantique dans les bases de données orientées objets" Rapport de spécification, tâche 1 phase 2, Infosys-Masi-Altaïr, septembre 1989.

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

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

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

[BUCH 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]

[CREM 83] CREMERS & DOMANNG, "An Integrity Monitor for the Database System Ingres", 9th VLDB Conf., Florence 1983.

[GUST 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.

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

[HAMM 75] HAMMER M.M. & McLEOD D.J., "Semantic Integrity in Relational Database Systems" 1st VLDB Conf., Framingham, USA Sept. 1975.

[HAMM 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, P.RICHARD. "The CO2 V0.4 Language and Some Extensions, Release 3.1" Rapport interne Altaïr 4-88, octobre 1988.

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

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

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

[MYLO 80] MYLOPOULOS J., BERNSTEIN P.A. & WONG H.K.T. "A Language Facility for Designing Database Intensive Applications" ACM TODS Vol-15, N°2, 1980.

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

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

[SIMO 84] SIMON E. & VALDURIEZ P. "Efficient Alorithm for Integrity Control in Database Machines" Proceed. of Internat. Conf. on Trends and Applications of Databases" IEEE-NBS, Gaithersburg, USA 1984.

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

[STONE 75] STONEBRAKER M. "Implementation of Integrity Constraints and Views by Query Modification" ACM-SIGMOD Conf., 1975.

[TSIC 82] TSICHRITZIS D. & LOCHOVSKY F. " Data Models" Prentice Hall 1982.

[TUCH 83] TUCHERMAN L., FURTADO A. & Casanova M.A. "A Pragmatic Approach to Structured Database Design" 9th VLDB Conf., Florence, Italy, 1983.

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

[VALL 88] VAN ASSCHE F., LAYZELL P.J., LOUCOPOULOS P. & SPELTINCK G. "Information System Development: a rule based approach", J. Knowledge Based Systems, 1(4) 1988.