

# MANAGEMENT OF COMPLEX OBJECTS AS DYNAMIC FORMS

Michel ADIBA<sup>1</sup>, Christine COLLET<sup>2</sup>

(1) *Laboratoire Génie Informatique* (2) *Centre de Recherche BULL*  
*Grenoble University BP 53X 38041 GRENOBLE Cedex*  
*electronic mail: adiba@imag.imag.fr, collet@imag.imag.fr*

## ABSTRACT

Traditionally, a form is either a paper document or an electronic object. It is used to describe, to structure and to facilitate the information flow inside an organization, or at the user interface level. From this well known concept, we propose an advanced form model. This model integrates three main areas of current database research: (1) Multimedia aspects, (2) Structural aspects related to the non first normal form models and (3) Dynamic aspects related to the Object Oriented approach.

The form paradigm that we propose offers a formal and homogeneous approach for describing and manipulating structural, dynamic and interface aspects of new database applications (Office Automation, Medical, CAD, ...). This approach provides a better control over object integrity and allows for the integration of various database objects: flat relations, multimedia documents and other forms. A formal definition of our model has been done and a prototype is currently under implementation on a workstation using a relational DBMS as a basic data manager.

## 1. INTRODUCTION

A Form is a well known and widely used kind of document for exchanging information, in a structured and non ambiguous manner. It was at the origin of several methods used in the organizations in order to facilitate information flow.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

In the database field, most of the research and development efforts consider the form at the interface level to facilitate man and machine interaction. The concept of electronic form appears immediately when people want to computerize their applications. [TSI 82, GIB 84]. Today, several tools added on top of a given relational DBMS (e.g. 4th Generation Language), offer a form concept. Through forms displayed on the screen, the user describes, manipulates, extracts informations from the database and associates to them specific formats (report, diagram, ...).

We classify these tools as:

1/ End-user oriented, for instance, QBF, VFE, RBF, GBF of INGRES [STO 86]

2/ Programmer oriented, in this case, the form is an object used as an application generator. For instance, IAF [ORA 84], ABF for INGRES [DOU 87], FADS [ROW 82, ROW 85]. For example, IAF allows to specify several screens and to model some dialog but the approach is rather artificial because of the use of the SQL language to manipulate the database but also to express specific operations on the form itself.

Previous work on forms can be found mainly in the area of Office Automation. In such applications, the form concept was very largely used in order to add to a DBMS a set of tools, namely electronic mail, word processing, editors, ... Office is seen as a large database, shared by several users [GIB84]. A form is a very convenient object both at the data model level [LUM 82, SHU 83, SHU 85, TSI 82, BAR 84], or at the methodological level for (1) integrity constraint control [FER 82, GEH 83], (2) office procedures automatization [ZLO 77, ZLO 82, DEJ 80, TSI 82]. In all these approaches, the use of form shows that it is really an object:

♦ easy to understand: allowing to interpret naturally the office world;

♦ "ergonomic": allowing a rational organization for data, actions and constraints;

♦ interactive: allowing better information exchange between all the actors of the organization.

---

*This research has received the support of the French Research Programm PRC/BD3.*

However, most of the approaches mentioned above are only considering one of these aspects. In the framework of multimedia databases and from our experience in the TIGER project [ADI 87a, VEL 84], we choose to have a more global view and we defined a model integrating these three aspects [COL 87].

Our approach is motivated by the fact that usually, DBMS are designed in a bottom-up process where a specific data model (e.g. relational) is implemented and then, user interfaces are built on top of this data manager in order to implement applications. For multimedia database systems, and specially in an office automation environment, we propose a rather top-down approach. Starting with the classical and well known notion of form as a tool for user interaction, we propose to extend this concept in several directions in order to design a complete system.

First we allow multimedia data to be stored in certain form fields. Second, we assimilate a form to a specific complex object, i.e. it is not only an object at the interface level but it becomes an object inside the database, this object has a hierarchical structure. Third, and because of the interactivity, the user is allowed to do some specific actions upon data displayed on the screen, letting the system reacting accordingly. This is very similar to the Object Oriented approach if we declare to the system (1) the specific actions that a user is allowed to do on a given object (a given form type), and (2) the sequence of operations that the system must do according to a given user action. We propose to capture the behavior of these complex "forms" by defining specific rules which belong to the form schema. In order to implement such a model, we need a (complex) object manager with dynamic capabilities [BDV 87].

To define our model, we took a similar approach than the one developed for non first normal form relational models [ABI 84, FIS 83, JAE 82, ROT 85a] or complex objects [ABI 87a, HUL 87]. The complete and formal definition of our approach can be found in [COL 87], however, we give here the main concepts that we propose, and particularly we introduce the form model as a complex object model. We describe briefly the interface and form presentation approaches together with the main characteristics of the implementation. Section 2 presents informally the form model. Section 3 gives the structural aspects of what we call the Abstract Form (FA) and section 4 gives an idea about the extended algebra which has been defined in order to build FA-expressions and to express form manipulation. Section 5 indicates how we treat the dynamic aspects through the notion of "rules" and section 6 gives some elements about interface and presentation aspects.

## 2. A MODEL FOR DYNAMIC FORMS

Figure 1 is an attempt to show how the different concepts of our form model are related together. A complex form is considered at different levels, by analogy with the ANSI/SPARC architecture.

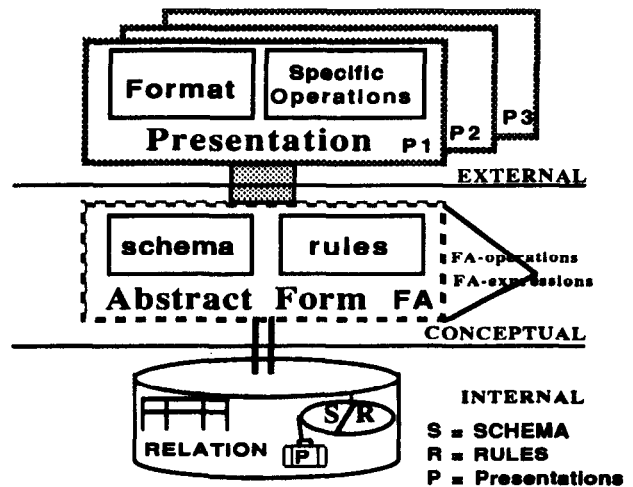


Figure 1 : Form representation levels

1. At the internal level of a database (or more generally an object manager), the form is described and stored in terms of the database model. Our prototype, for instance use a relational DBMS for that. This choice is for rapid prototyping only and we are thinking of an Object Oriented DBMS for that purpose [LEC 87, BDV 87].

2. At the conceptual level, the notion of **Abstract Form (FA)** is a class of forms having the same structural and semantic properties. (e.g. the travel expenses form). When we have to consider an Abstract Form from a semantic point of view, we have to distinguish between structure and contents.

**FA** gives the specification of a forms class. It is twofold: first the **schema** which describes the form structure and second the **rules** which express dynamic aspects of the application. The method for rule definition is based upon the use of FA-expressions similar to (extended) algebraic expressions in the relational model. FA-expressions (see section 4 and 5) are built using different FA-operations for querying, updating and doing specific calculations (the power of FA-expressions is analogous to the power of an SQL-like language or an extended SQL-like [PIS 86, ROT 86]). A rule is composed of an activation condition (WHEN) and BEFORE and AFTER clauses. The meaning is similar to an on-condition: WHEN specific actions will occur on the form (see presentation below), then execute the BEFORE and eventually the AFTER clauses and accept or reject the actions (see

section 5). Hence, a FA is a set of occurrences which obey to a forms class specification (FA). The FA and FA notions "live together" and the term FA will be used to refer to both structure (or schema) and contents of an Abstract Form.

We think that this is a very important point because we are trying to express complex objects semantic and dynamicity: other complex object models proposed so far : NFN relations models [FIS 83, SCH 84, ROT 85a, PIS 86], the model from the SPECDOQ system [KIT 84], VERSO [BAN 82, ABI 84], TIGRE [VEL 84], or the one by [ABI 87] are only considering static and structural aspects and therefore dynamicity must be expressed through specific programs.

3. At the external level called **presentation** level, we can have several presentations for a given FA. A presentation is twofold: a **format** and a **list of specific operations** which are allowed on the form. A format expresses how the form is to be displayed on the screen and it may change from one user to another, allowing different formats for a given FA. Here also, we want to stress this point because, compared to traditional complex object models, we are proposing a notion of object presentation, at the interface level.

Besides the format, a presentation contains the list of specific operations. They correspond to operations that the user is allowed to invoke (e.g. create a form, retrieve, update... see section 6). A specific operation is always associated to a format and concerns a group of users. Hence, the list of these operations for a format represents the dynamic aspects from an external point of view (the user). The invocation of a specific operation for a given form occurrence, can be seen as the opening of a context (e.g a transaction) for executing elementary actions on the form elements (filling a field, modifying another one). Each of these actions are considered as events and their occurrence may trigger the execution of specific rules defined in the schema. Like this, we are modeling the behavior of the object (form) when it is used. In summary, we can say that our form notion is the result of the association of one FA and one presentation. This results in a complex object that the user manipulates from his/her workstation. The system is then responsible for mapping these manipulations onto actions on database objects.

A form example is given in figure 2 and will be used throughout this article. It is used in a library environment for managing book loans. Each member which can borrow several books, has a name, a sex, an address. Each loan is described by the book reference, title, loan date, and return date. Also, we have the total number of books on loan. To register a new member in the library, the librarian will

invoke the CREATE operation which allows for filling the different fields. When the user considers that all the occurrence is complete, he invokes the VALIDATE operation which validates the creation and a new member is registered inside the database.

During this dialog, a specific rule may be activated, for instance to verify if the new member is not already inside the database. From the CREATE operation to the VALIDATE operation, the form is manipulated, at the external level, inside a working area distinct from the database.

Figure 2 : Form Example

### 3. ABSTRACT FORM (FA)

As we have seen, an Abstract Form represents a class of forms (orders, invoices, library member cards) which have the same structure and semantic. As we said a distinction have to be done between the design of an Abstract Form (FA) and its contents (FA). A FA is composed of (i) a schema and (ii) a set of rules. A FA is a set of occurrences, constructed using a schema. In the following, we define more precisely what we mean by schema and occurrence. The second component of a FA (the rules), will be described in section 5, after the presentation of query and update FA-operations (section 4).

#### 3.1 SCHEMA

Schema notion can be compared to the type notion for structured objects. We use the formalism proposed in [ABI 87]. We give a special attention to hierarchical schemas, cyclic schemas are not considered here. A schema is built

using four constructors, i.e. (1) the **tuple** constructor which allows the aggregation of any number of elements, (2) the **set** constructor allows the aggregation of elements of the same kind, (3) the **list** constructor allows the ordered aggregation of a bounded number of elements of the same kind and (4) the **choice** constructor which allows a choice between different elements.

### Preliminaries

We assume the existence of an infinite set of values called **domains**. An element of a domain is called an atomic object. For multimedia database systems, many domains should be considered: integer, real, string, text, image, voice, etc. To simplify the presentation here, we consider only one domain, namely **D**. We consider a particular domain, denoted by  $\square$  to allow constants in a schema. In addition of punctuation symbols (":", "[", "(", ...), we assume the existence of an infinite set of symbols called **elements** (in other models they are called attributes). We also assume the existence of three null values [ROT85b]:

- $\partial$ : nonexistent (or does not exist) null (*dne*),
- $?$ : unknown null (*unk*),

$\phi$ : no-informative null.  $\forall D, \phi \in D$

The value  $\partial$  in a form field means that this field is not applicable for this occurrence. The value  $?$  denotes an unknown value and  $\phi$  correspond to a no-informative null value (e.g an empty set is considered as a set containing  $\phi$  [ROT 85b]).

#### 3.1.1 DEFINITION

The definition of a schema **S** consists of two components, a name and a description (that could be a domain). Any schema **S** has the form **A:da**. The first component **A** is the name of **S** and is denoted by **name(S)**. The second component **da** is the description of **S** denoted by **des(S)**.

The set **S** of allowed schemas is defined using the following rules:

- ◆ if **A** is an element, then  $A: \square \in S$
- ◆ if **A** is an element and **D** a domain, then  $A:D \in S$
- ◆ if **A**, **A1**, **A2**, ..., **An** are distinct elements and if  $S_i = A_i: d_i \in S, i \in [1..n]$ , then
  - ◆  $A: [ S_1, S_2, \dots, S_n ] \in S$  *tuple*
  - ◆  $A: S_1 | S_2 | \dots | S_n \in S$  *choice*
  - ◆  $A: \langle S_1 \rangle_n \in S$  *list of at most n element*
  - ◆  $A: \{ S_1 \} \in S$  *set*
- ◆ if  $S_1 \in S, (S_1)_{dneallowed}$  and  $(S_1)_{unkallowed} \in S$

The following examples illustrate the notion of schema.

**Example 1:**  $S_1 = \text{address}: [ (\text{street}: \text{string})_{unkallowed}, \text{zip}: \text{integer}, \text{city}: \text{string} ]$  *tuple*  
 $S_2 = \text{sex}: \text{man}: \square \mid \text{woman}: \square$  *choice*  
 $S_3 = \text{loans}: \{ \text{loan}: [ \text{ref}: \text{string}, \text{title}: \text{string}, \text{begin}: \text{time}, \text{return}: \text{time} ] \}$  *set*

For the schema **S1** of the above example 1, **name(S1)**= address gives the name of **S1** and its description is **des(S1)**= [ (street: string)*unkallowed*, zip: integer, city: string ] .

Because of the recursive nature of schema definition, it should be clear that to any schema **S** we can associate the set {**S1**, **S2**, ..., **Sn**} of schemas which compose **S**. A schema **Si** of this set is called an "element schema" and its first component **name(Si)** is called an element of **S**. For example, we can associate to the schema "address", the set { street: string, zip: integer, city: string }. "street" and "city" are elements of the schema "address". Knowing an element **E** of a schema **S**, the function **sch\_elem** gives the schema of **E**.

As proposed by other people [SCH 84, ABI 86, HUL 87], we can represent schemas by trees. The cross (x) represents the tuple constructor, the plus (+) the choice constructor, the star (\*) the set constructor and the star subscript n (\*n) the list constructor. The trees of Figure 3 show the schemas **S1**, **S2** and **S3** of Example 1.

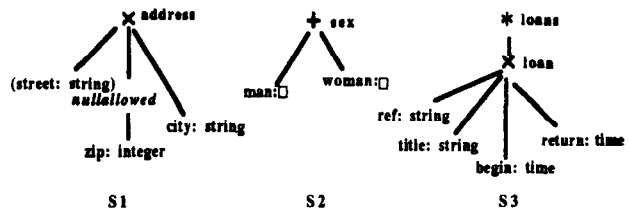


Figure 3: Tree schemas

#### 3.1.2 DOMAIN

An object built using a schema is an **occurrence**. The set of possible occurrences for a given schema **S** defines the domain of **S**.

##### Definition

The domain of a schema **S**, denoted **dom(S)** is defined by:

- ◆  $S = A:D, \text{dom}(S) = \{ A:a \mid a \in D \}$
- ◆  $S = A:\square, \text{dom}(S) = \{ A:\square \} = \{ A \}$
- ◆  $S = A: [ S_1, S_2, \dots, S_n ], \text{dom}(S) = \{ A: [ s_1, s_2, \dots, s_n ] \mid \forall i \in [1..n], s_i \in \text{dom}(S_i) \}$
- ◆  $S = A: S_1 | S_2 | \dots | S_n, \text{dom}(S) = \{ A: s \mid \exists i \in [1..n], s \in \text{dom}(S_i) \}$

- ◆  $S = A: \langle S1 \rangle_n$ ,  
 $\text{dom}(S) = \{A: \langle s1, s2, \dots, sk \rangle \mid \text{for } 0 \leq k \leq n$   
and  $\forall i \in [1..k], si \in \text{dom}(S1)\}$
- ◆  $S = A: \{ S1 \}$ ,  
 $\text{dom}(S) = \{A: \{s1, s2, \dots, sk\} \mid \text{for } 0 \leq k$   
and  $\forall i, j \in [1..k], i \neq j, si \neq sj$  and  $si, sj \in \text{dom}(S1)\}$

A complete definition of the domain of a schema  $S$  with the option *dneallowed* (respectively *unkallowed*) is given in [COL 87]. This definition shows that a *dne* (*unk*) occurrence is composed of *dne* (*unk*) sub-occurrences. We also give simplified representations used for *dne*, *unk* occurrences, and no-informative (empty) occurrences. For example,  $A: \langle \phi \rangle$  represents an empty list i.e an occurrence built from a schema  $S = A: \langle S1 \rangle_n$ . The empty set is denoted :  $\{ \phi \}$ .

The following examples illustrate the notion of occurrence.

(1) address: [ street: *10 av couchetard*, code: *38100*, city: *Grenoble*]

address: [ street: ?, code: *38402*, city: *St Martin d'hères*]

These occurrences have been built from the schema "address" of Example 1. The second occurrence have the sub-occurrence "street: ?" which means that now "street" is unknown.

(2) sex : man:

sex : woman:

These occurrences have been built from the schema "sex" of Example 1. They do not have the same schema, because "sex" is a *choice* schema.

(3) loans: { loan: [ ref: *A6*, title: *L'amant*,  
begin: *20/12/1987*, return: *31/12/1987*],  
loan: [ ref: *A5*, title: *Les eaux brûlées*,  
begin: *20/12/1987*, return:  $\phi$  ] },

This occurrence have been built from the schema "loans" of Example 1.

#### Notation :

In the formalism used for schema declaration, we have to note the importance of elements. For a given occurrence, elements allow to distinguish and to name its sub-occurrences. Let us consider the occurrence  $O = \text{address: [street: } 10 \text{ av couchetard, zip: } 38100, \text{ city: } Grenoble]$ , then "O.street" denotes the sub-occurrence "street: *10 av couchetard*" which is called an element value. For the occurrence  $O = \text{"sex:man:}\square\text{"}$  (built from the schema  $\text{sex : man:}\square\text{ | woman:}\square\text{}$ ), "O.woman" denotes an undefined sub-occurrence. For the occurrence  $O$  of example (3) above, the element "loan" indicates any sub-occurrence of the set "loans". The use of "loan", to indicate a particular sub-occurrence, will be possible in some quantified expressions

on a set. For example, it is possible to write :  $\forall I \in O, O.\text{loan} = I$  and "loan" designates successively the two sub-occurrences of  $O$ . Other notations are given in [COL 87], specially to locate a particular sub-occurrence of a list. In the context of an occurrence  $O$  built from a schema  $S$ , an element of  $S$  represents (generally) a sub-occurrence of  $O$  (value of element) if it exists. From a given occurrence it is almost always possible to determine the corresponding schema. Obviously exceptions come from the choice constructor and from empty occurrences.

### 3.2 FA

A FA corresponds to a set of occurrences stored at a given time in the database. Occurrences are built from  $\underline{FA} = \langle S, R \rangle$ . The component  $S$  is one schema of  $S$  and  $R$  is a set of associated rules (see section 4). Formally a FA is defined by :


$FA = \{ O \mid O \in \text{dom}(S) \text{ and } R [O] \}$  where  
 $R [O]$  denotes a successful execution of rules for  $O$ .

The difference between  $\underline{FA}$  and FA is similar to the distinction between relation schema and relation in the relational model and more generally between data type and data. The notions of  $\underline{FA}$  and FA "live together" and we misuse the term "FA" to describe the pair ( $\underline{FA}$ , FA). When we want to make the distinction clear, we use SCHEMA for  $\underline{FA}$  and VALUE for FA.

As an example of FA, consider Figure 4 which shows informations of a set of forms presented in Figure 2. The VALUE of FA "member" is represented by table of Figure 4(b). Each line of this table gives an occurrence built from the schema "member" of Figure 4(a). Some rules related to this schema will be presented further.

The following notations are used. Consider the FA  $\Gamma (\langle S, R \rangle, \Delta)$ ,  $\text{sch}(\Gamma) = S$  and  $\text{val}(\Gamma) = \Delta$ .

We extend the function name (on a schema) to be directly applicable to FA  $\Gamma$  by the following definition :  $\text{name}(\Gamma) = \text{name}(\text{sch}(\Gamma)) = \text{name}(S)$ . The notion of element, element schema and element value can be used for a FA.

The FA "member" has the elements "name", "address", "sex", "loans" and "total". In the context of the occurrence " member: [name : *balou*, picture : , address: [street: *10 av couchetard*, zip: *38100*, city: *Grenoble*], sex: man: , loans : {  $\phi$  }, total: 0 ] "

to the element "address" correspond :

- ◆ the element schema "address: [street: string, zip: integer, city: string]"
- ◆ the element value "address: [street: *10 av couchetard*, zip: *38100*, city: *grenoble*]"

## 4. FA operations

This section presents the operations provided to manipulate and update FA. The "form algebra" and its formalism proposed here do not define a user friendly language or interface. We try to capture the elementary operations that are really necessary to retrieve, create, delete and modify occurrences. We present them in the following using the FA "member" of Figure 4.

We first present a set of (recursive) algebraic operators in order to have an homogenous framework for FA manipulation and consultation. A FA is a natural extension of a non first normal form ( $-1NF$ ) relation. Therefore, we construct our operations in a way similar to  $-1NF$  relational models [JAE 82, SCH 84, ROT 85a, ROT 85b, PIS 86, FIS 83, ABI 84], and also to structured object models [ABI 87, BAN 88]. The update operations are defined later (section 4.2).

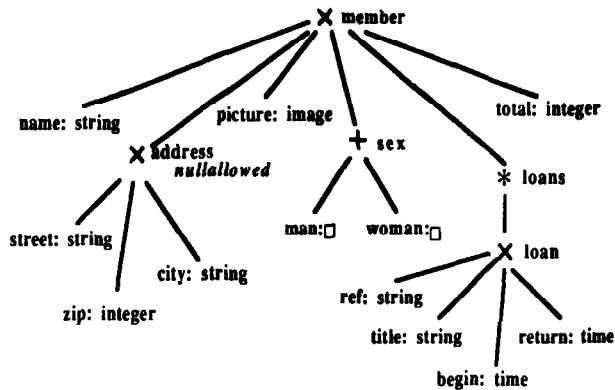


Figure 4(a) : Schema of the FA "member"

## 4.1 Query FA-operations

As for algebraic languages, we consider that the information we want to retrieve could be represented by a FA built from successive applications of (unary or binary) FA-operations.

As mentioned earlier, a FA  $\Gamma$  is defined by a couple  $\langle S, R \rangle$ ,  $\Delta \subseteq \text{dom}(S)$ . The component  $R$  (rules) is not relevant for the operations definitions and will be discussed later. Therefore, in the following we consider a simplified definition of a FA :

$\Gamma(S, \Delta)$  is a FA,  $S \subseteq S$  and  $\Delta \subseteq \text{dom}(S)$ .

In [COL 87], we prove that in the context of an occurrence  $O$  (of a FA  $\Gamma$ ), an element schema  $S_E$  together with an element value  $\Delta_E$  forms a FA  $\Gamma_E$  which in turn may have elements. So we are able to apply the definitions and notations introduced in section 3 in a nested manner. The operations proposed to consult and manipulate FA may be applied also in a nested manner, i. e repeatedly to a given FA but also on any of its sub-FA. This approach is similar to the one proposed in [SCH 84] for  $-1NF$  relations algebra. Functions of the main operations are summarized in Figure 5.

A complete and precise definition of each operation can be found in [COL 87]. The filtering operations are used to choose some occurrences in a FA (**selection**), to transform occurrences "pruning" some edges (**prune**), to **rename** some elements of FA and to arrange occurrences in a certain order (**sort**). The set operations : **union**, **intersection**, **difference** and **product** have their usual meaning. The **nest** and **unnest** operations are extensions of nest and unnest operations in the  $-1NF$  relational model [JAE 82, SCH 84, FIS 83, ROT 85a]. We proposed also some other operations allowing

× member

name	picture	× address			+ sex	* loans	total
		street	zip	town			
balou		10 av couchetard	38100	Grenoble	man: <input checked="" type="checkbox"/>	loan: $\emptyset$	0
geser		[ ? ]			woman: <input checked="" type="checkbox"/>	loan:[ref:A1, title:El tunel, begin:01/11/1987, end:30/11/1987] loan:[ref:A5, title:Les eaux brûlées, begin:15/11/1987, end: $\emptyset$ ] loan:[ref:A2, title:Cocaine, begin:01/11/1987, end:15/11/1987]	3
dobey	$\emptyset$	50 cr J. Jaurès	38100	Grenoble	woman: <input checked="" type="checkbox"/>	loan:[ref:A1, title:El tunel, begin:02/12/1987, end: $\emptyset$ ] loan:[ref:A2, title:Cocaine, begin:02/12/1987, end:15/12/1987]	2
nay		[ ? ]			man: <input checked="" type="checkbox"/>	loan:[ref:A6, title: L'amant, begin:20/12/1987, end:31/12/1987]	1

Figure 4(b) : VALUE of the FA "member"







value, we could introduce marks on  $\phi$  or  $\emptyset$  null as in [ROT 85b].

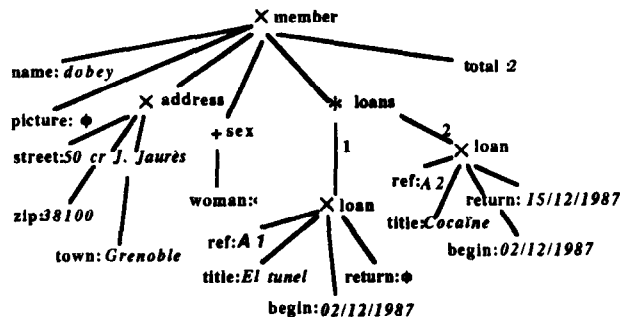
## 4.2 Updates FA-operations

The table of Figure 8 presents the update FA-operations that we have defined. The operations : **create**, **modify** and **delete** apply on an occurrence and **assign** and **remove** apply on an element. As presented informally in section 2, the FA manipulation takes place at the external level and any creation will be done using a working context for the new occurrence. This workspace will keep track of any action on the (temporary) form, e.g. assign, remove, rule activation, etc. As soon as the user validates all these actions, and if every rule was executed correctly, the new form occurrence will effectively be stored into the database with a unique identifier.

OPERATIONS	FUNCTION
create	create an occurrence of FA
modify	modify an occurrence of FA
delete	delete an occurrence of FA
assign	assign a value to an element of occurrence
remove	remove a value to an element of occurrence

Figure 8 : Update FA-operations

The **assign** and **remove** operations are used to modify the elements in an occurrence. They are not defined recursively. An occurrence could also be represented as a tree. The following tree represents the third occurrence of FA "member" ( table (b) of Figure 4).



To update a node of the tree (a value of element), we have to designate it. The notion of path is used here to refer to an element : it may be thought of as a "path" through the tree. A path is always linked to a specific occurrence. The syntax for a path is as follows :

$\langle \text{path} \rangle := \langle \text{occurrence-id} \rangle . \langle \text{element-chain} \rangle$ .

As examples of paths, suppose the above occurrence is identified as the  $id_{01}$  occurrence, then we can have:

$id_{01}.name$ ,  $id_{01}.address.city$ ,  $id_{01}.sex.man$ ,  
 $id_{01}.loans$

Paths are evaluated from left to right. For example " $id_{01}.name$ " refers to the name of the member  $id_{01}$ . " $id_{01}.sex.man$ " is undefined and " $id_{01}.loans$ " is the set of loans done by the member  $id_{01}$ .

### Examples of update operations :

**assign( $id_{01}.name$ , 'toto')** : replace the previous value of the element "name" for occurrence  $id_{01}$  (which may be null, i.e "name: $\phi$ ") by value "name:toto".

**assign( $id_{01}.sex$ , 'woman')** : assign the value "sex:woman" to the element "sex" of the occurrence  $id_{01}$ . This element have a *choice* schema and this indicates the schema name which replaces the previous one, if any.

**assign( $id_{01}.loans(3).return$ , '24/12/1987')** : assigns a value to the element "return" of the third "sub\_occurrence" of set "loans". The subscript is used to designate sub\_occurrences.

The **remove** operation is in a sense the reverse operation for **assign**. It has the following syntax **remove( $\langle \text{path} \rangle$ ,  $\langle \text{value} \rangle$ )**. In the case of an *aggregate* element E or a *choice* element E, the remove operation replaces the value of the element (E:v) with E: $\phi$ . If the element has a *set* or a *list* schema, then  $\langle \text{value} \rangle$  is removed from the set or list which is the value of the element designated by  $\langle \text{path} \rangle$ .

## 4.3 FA\_expressions

The FA-operations presented above could be used for FA query and update. To express other operations, these functions are not sufficient. We would like (i) to express arithmetic and logical calculus, (ii) to express some algorithmic operations, (iii) to use the result of a query as a temporary FA etc. In order to extend the power of FA-operations, we introduced FA-expressions. FA-expressions have to be considered as a formalism which is part of our model and not as a user language. More precisely, an FA-expression is constructed using :

- ◆ Query and Update FA-operations,
- ◆ the operation FA which specifies a temporary FA. The specification involves the use of a FA-variable : i.e a name local to the transaction in which it has been defined and bound to a set of occurrences. There are two methods to define the set of occurrences : query FA-operations and extension. The temporary FA could be seen as a cursor and then could be manipulated accordingly (with the iteration operation : for).
- ◆ functions for handling data values related to data types (integer (+, \*...), text, time), users functions (corresponding to particular programs) and programming functions such as : "if then else", "for each", ...

In FA-expressions, variables local to a transaction may appear. They are denoted by an identifier with a question mark. They are assigned during the expression execution. They naturally could be referred in other expressions.

#### Examples of FA-expressions :

**create**(member, ?a); create a new occurrence ?a of FA "member" : ?a will take the value of the occurrence identifier.

**assign**(?a.total, *count*(?a.loans)); for any occurrence ?a, assign to element "total" the result of the function *count* applied on the set of "loans".

**FA**(LATE, **selection**(member, **selection**(loans,  $\exists$  **selection**(loan, and (= (return.  $\phi$ ), longer (subtime ('now', begin), 1month))))));

**for** ( (LATE, ?r), **print** (?r.name); );

The first FA-expression (operation FA) creates a temporary FA containing all members which still have books on loan for more than a month. The second expression prints the name of these members.

## 5 RULES

Here, with forms, we do not try to solve all the problems related to object dynamicity management. Concerning this area, a lot of research has been done. Some propose to introduce in data models (i) time [ADI 86, ADI 87b, BUR 87] (ii) events and operations [ANT 81, LIN 87, BAN 87]. Complex objects dynamicity is treated here in the precise framework of the interaction between a form and a user.

Rules are the second component of a FA <S,R> (meaning of a FA). They encode the semantic tied up to the schema S and give a **dynamic behavior** to the object (form) presented to the user at the interface level. The form reacts to the users actions, realizes controls and FA-operations. The rule concept enhances the schema concept in the same way (i) methods enhance the class concept in object oriented languages [GOL 83, LEC 87], (ii) facets (procedural attachments) in frames [BEN 85, FIK 85, VIG 85]. Also the rule notion we defined here is an extension of the "triggers" proposed in [GIB 84]. Many examples of rules are given in [COL 87], showing that the rule notion should be used to express :

- ◆ several integrity constraints for database (complex) objects,
- ◆ objects status : mandatory, no modifiable, locked, ...
- ◆ calculation of value element, either local to a given form occurrence or global when values belong to other database occurrences.
- ◆ update translation in the case of virtual elements (view constructor [COL 87]).

- ◆ exceptions : the concept of semantic tolerance introduced in [ESC 87] gives a new approach to model and handle exceptions in databases [BOR 85]. We show that it is possible to introduce some "tolerance" in rules in order to soften controls and accept exceptional data.

A rule is defined by : **DEF\_R** <rule name>

**WHEN** <evt1>; <evt2>; ...<evtn>;

**BEFORE** <FA-expressions>

**AFTER** <FA-expressions>

The **WHEN** clause, identifies a list of events which activate the rule. The main difficulty here is to define precisely what is an event [ANT 81]. In our framework, an event will be an activation of one update FA-operations : create, modify, delete, assign and remove.

*Rule1* (Figure 9) is intended to control assignation of values to begin and return dates for a book loan. We find two events (ev1 and ev2). An event occurs when the corresponding operation is invoked by the user. So, *Rule1* will become active when ev1 occurs and ev2 occurs or vice versa. In this way, a rule activation is controlled by one event, the last one which occurs. The corresponding operation for that event is called an activation operation. In our example, when both values for ?b and ?r are given by the user to the system, they will be accepted only if ?b precedes ?r in time. A rejection (stop-event) causes the non acceptance of the update operation (assign) and the system may wait until consistent values are given.

**WHEN** assign(?m.loans(?i).begin, ?b); - ev1  
assign(?m.loans(?i).return, ?r); - ev2

**BEFORE**

if ( precede (?r, ?b), message('the return loan date can't precede the loan date'); stop\_event;)

**Figure 9 : Rule1**

In general, every operation on a FA is controlled in order to determine the rules which are concerned by the corresponding event. For a given rule R, as soon as all the events of its **WHEN** clause occur, then R is activated. If a **BEFORE** clause exists, the corresponding FA-expressions are performed. Usually, they are used to verify integrity constraints. If one constraint is not satisfied, we provide the "stop-event" operation to force the rule to fail. For instance, in the **BEFORE** clause of *Rule1*, there is only one FA-expression which permits to verify that "the return date of a book on loan do not precede the loan date". If the rule R did not fail or if there was no **BEFORE** clause, then we consider the **AFTER** clause. This one may contain FA-expressions which have to be executed. Two cases may occur : (1) correct execution of all these expressions result in a successful rule R. (2) if one of these expressions is an event for another rule R' and if R'

fails, then, to avoid side effects, we rollback all the operations performed since the beginning of R. For instance in *Rule2* (Figure 10) : when the event "assign(loans(?i).ref, ?r)" occurs, then the BEFORE is executed. It verifies first if the requested book is in the LIBRARYCATALOG inside the database. If so, it verifies that the book is not already on loan. If the book is available, the assign is accepted and the AFTER clause is executed. This results in assigning to the form the book title which was extracted from the database by the execution of the first selection of the BEFORE clause. In this way title will be displayed on the screen.

```

WHEN assign(?a.loans(?i).ref, ?r);
BEFORE FA(CAT,
  FA(CAT, selection (LIBRARYCATALOG,
    =(serial-number, ?r)));
  if( non( exist(CAT)), message('no book corresponding
    to this serial number');
    stop-event;)
  FA(NOTRETURN, selection (member, (selection
    (loans,  $\exists$  selection(loan, and (=ref,?r), =(return,  $\phi$  ) ))))
  if( exist (NOTRETURN), message('this book is already
    on loan'); stop-event; )
AFTER for ( (CAT, ?c),
  assign(?a.loans(?i).title, ?c.title); );

```

Figure 10 : Rule2

**Remark :**

An important step during rules specification is to examine their behavior and their correctness . This is not a trivial problem. First a rule must not be inconsistent. In our case, this will require to do proof of FA-expressions. A more formal treatment of this problem is related to program proving and of course it is not our intention to solve this problem here. A second problem is when two (or more ) rules are activated by the same operation . If we accept such situation, we need a general policy for the activation of such rules. As in [GIB 84], the policy is that rules which have at least a common event in their WHEN clause should have FA-expressions (in the BEFORE or/and AFTER clauses) that operate upon mutually exclusive objects. A third problem is the detection of loops. It is easy to define - a rule which have a loop : when it is activated, it will reactivated itself indefinitely - or a rule which can form a loop with existing rules. We can handle a dependency graph for "before" and "after" of the existing rules. All the problems introduced above still belongs to the research area. We think, however that our approach is sufficient for several interactive applications where the users apply a methodological approach for specifying dynamic aspects through rules.

**6 PRESENTATION**

We give in this section some elements about the concept of presentation. Remember that a FA can have different presentations. A presentation have two components: a format and a list of specific operations.

**6.1 FORMAT**

The format is the specification of how a FA (or only a part of it) presents itself to users on a particular output. We limit ourself to formats for displaying on screen. A format is defined by the primitive : **DEF\_FT <SCHEMA\_FT> FOR <FA> WITH <users>**.

<SCHEMA\_FT>describes a set of boxes. Their organization reflects the schema of the <FA>.In the FOR clause, we give a name of FA or an FA\_expression. Finally, in the WITH clause is given a list of "users" allowed to use the format. In SCHEMA\_FT, we find three types of information described for templates [TSI 82] : (i) box appearance, (ii) box - element mapping, (ii) box positioning. The approach chosen here is similar to the one proposed in JANUS [CHA 81], but we do not use a programming language to describe a format. Each box is described by attributes. The following table gives an exhaustive list of attributes .

BOX ATTRIBUTE	FOR
name	box name
title	box title
title position	above, side
type	simple, multiple
appearance	background
dimension	height, width
box position	above, under, inside a box
contents	element-chain / constant
justification	contents justification : centered, left justified ...
borders	thicked, dotted, ...
description	text for help
display status	always / condition
expandable	true / false

**STANDARD FORMAT**

To help the programmer during form specification, we provide a mechanism to generate a standard format for each FA. For that, some informations must be given first : format name and name of the corresponding FA, its type (Simple to display one occurrence, Multiple to display several occurrences, Mn to display a menu or a choice). This type is used to determine the control functions associated with the format. Then the mechanism will be able to define a set of boxes according to the schema of the specified FA. For each schema, a box is defined with :

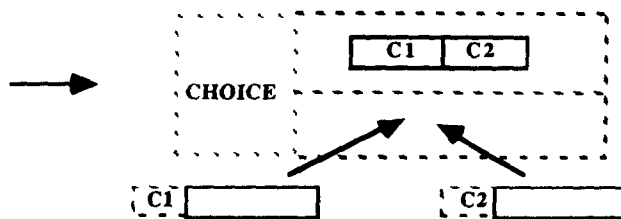
- an internal *name* ,
- a *title* which is the name of the schema. Its position depends of the box which includes it (by default left to the box)

- a *justification* : left and right justified
- a *standard appearance*
- a *description* which is the element schema
- display status : always

The others attributes are defined using mapping-rules, described in [COL 87]. As example of mapping-rules, the following attributes are defined for a *choice* schema :

- type : simple
- dimension : calculated in accordance with the dimension of the biggest box defined for schemas which compose the choice and the "menu" box (see below).
- borders : nothing
- format : horizontal. The box have two horizontal zones. The first zone have a "menu" box. Its contents gives the possible choices (names of schemas which composed the choice schema). The second zone is reserved to display the box defined for the schema which would be chosen. The size of this zone is equal to the one of the biggest box associated to one of the schemas which compose the choice. When one of these schemas defines a constant (domain = □), no box will be specified.

*SCHEMA\_FA* = choice : C1: string | C2 : integer



## 6.2 SPECIFIC OPERATIONS

A specific operation is related to a format and concerns one "user". It specifies the actions (events on FA through the format) which could be accepted from the user. A specific operation gives a first level of control for the users actions without activation of rules and then, open a context where rules are considered (see section 2 and 5). The names of specific operations will define the menu used to manipulate a form, resulting from the connection of a presentation with a FA.

## 7 CONCLUSION

We presented here a generalization of the **FORM** concept. Considered mainly in Office systems, the form is not only an interface object but is considered as a complex and dynamic object at different levels of database definition, manipulation and interaction. We think that our proposition constitutes an homogeneous framework for describing and managing structural and dynamic aspects for

complex (and multimedia) objects as forms. This work is related to several research areas: complex or structured object models, extended algebra, knowledge representation, data dynamicity, database interfaces. We tried to integrate several aspects in order to offer proper tools for the design and the implementation of new database applications.

A first implementation of this model is already in progress using on an Apollo workstation a relational DBMS with a C interface and with specific interface tools . We choose a subset of operations on forms and we are trying to implement our system in an incremental manner by taking advantage of the meta level of the model. Offering menus and interfaces for the definition of form schemas is done through specific "system" forms to which they correspond specific "system" rules, here coded in the C language. This is a very important point because, for instance an interface with a pop-up menu can be model as a form with a choice between several elements. This approach appears to be very promising.

Complex objects are mapped onto flat relations in a conventional (although, not efficient) manner [VAL 86] and rules are implemented in C+SQL. One of the main difficulty is to separate clearly (1) external and conceptual levels in order to be independent of any specific interface tool [COU 87];and (2) conceptual and internal levels in order to consider in the future, object oriented DBMS [LEC 87, BDV 87]. A more detailed discussion on these aspects will appear in a following paper.

We are considering several directions of research as a continuation of this work. First, we think that more work is needed on dynamic aspects for complex forms. Our notion of rule must be extended and need more formal considerations. It should be compared to a strict Object Oriented approach with encapsulation in order to find a good level for expressing set of objects manipulations. This will help for implementing extended database query and manipulation languages through form interfaces.

Second, we want to define in a more systematic manner how to design a complex object manager integrating structural and dynamic capabilities. This work is part of a new important project that we are currently defining. Two main directions are considered, one concerns knowledge representation and manipulation and the other one is related to multimedia aspects of complex objects. We mentioned only this problem at the beginning of the paper but we are far away from real "multimedia" complex forms!!

**Acknowledgments** : The authors wish to thank C. Delobel and S. Abiteboul for their helpful and constructive comments on this research and the anonymous referees for their pertinent remarks.

## REFERENCES

- [ABI 84] S. ABITEBOUL, N. BIDOIT : *Non First Normal Form relations : an algebra allowing data restructuring* . Proc. ACM SIGACT-SIGMOD Symposium on Principles of Database System (1984). Journal of Computer and Sciences (December 1986).
- [ABI 86] S. ABITEBOUL, R. HULL : *Restructuring of semantic database objects*. Proc Int. Conf. on theory of databases. Rome, September 1986. To appear in theoretical Computer Science.
- [ABI 87] S. ABITEBOUL, S. GRUMBACH : *Bases de Données et objets Structurés*. Techniques et Sciences de l'Informatique. December 1987.
- [ADI 86] M. ADIBA, N. BUI QUANG : *Historical Multimedia Database*. VLDB Conference. Kyoto, Japan (August 86)
- [ADI 87a] M. ADIBA : *Modelling Complex Object for multimedia databases*. ENTITY RELATIONSHIP APPROACH. S. Spacapietra (Editor)
- [ADI 87b] M. ADIBA, N. BUI QUANG N., C. COLLET : *Aspects temporels, historiques et dynamiques dans les Bases de Données* . Techniques et Sciences de l'Informatique. December 1987.
- [ANT 81] V. ANTONELLIS, B. ZONTA : *Modelling events in Database Application design*. Proc 7th VLDB Conf. Cannes, September 1981.
- [BAR 84] F. BARBIC, M. CARLI, B. PERNICI, G. BRACCHI : *A tool for form definition in office information systems specification* . New Applications on Databases edited by G. Gardarin. On Proc ICOD-2 Conf. Workshop Cambridge University, September 1983.
- [BEN 85] A. BENSALD : *Un modèle de données relationnel étendu*. Thèse Docteur Ingénieur. INPG de Grenoble, June 1985.
- [BAN 82] F. BANCILHON, P. RICHARD, M. SCHOLL : *Verso : a relational Back end Database Machine*. Proc. International Workshop on Database Machines. San Diego (1982).
- [BAN 87] F. BANCILHON, S. KOSHAFIAN : *A calculus for Complex Objects*. Proc of ACM Symp. on PODS, Boston, March 1986.
- [BAN 86] F. BANCILHON, T. BRIGGS, S. KOSHAFIAN, P. VALDURIEZ : *FAD, A powerful and Simple Database Language*. Proc of 13th VLDB Conf. Brighton 1987.
- [BDV 87] F. BANCILHON, V. BENZAKEN, C. DELOBEL, F. VELEZ : *The 02, VO Object Manager Interface*. Technical Report Altair 11-87. September 1987.
- [BOR 85] A. BORGIDA, K.E WILLIAMSON : *Accommodating Exceptions in Databases, and Refining the Schema by Learning from Them*. Proc. of VLDB Conf. Stockholm, August 1985.
- [BUR 86] R. BURSENS, J. GUYOT : *Temps + Dynamique + référentiel = Histoire* . Troisièmes Journées Bases de Données Avancées. Port-Camargue, May 1987.
- [CHA 81] D.D. CHAMBERLIN, J.C KING, D.R SLUTZ, S.J.P TODD, B.W. WADE : *JANUS , An interactive system for document preparation*. Proc. ACM. Symposium on text manipulation. June 1981.
- [COL 87] C. COLLET : *Les Formulaires complexes dans les bases de données multimédia*. Thèse de doctorat de l'USTMG. Grenoble. November 1987.
- [COU 87] J. COUTAZ : *PAC, an object Oriented Model for Dialod Design*. Human Computer Interaction INTERACT'87. H.J Bullinger and B. Shackel (Editors). North-Holland.
- [DOU 87] A. DOUCET, C. LEPENANT : *Langages de quatrième génération et générateurs d'interfaces* . Rapport technique GIP Altair No 3-87 (IN2-INRIA-LRI). March 1987
- [DEJ 80] P. DE JONG : *The system for business automation (SBA) : A unified application development system*. Proc. IFIP Conference (1980)
- [ESC 87] C. ESCULIER : *Inheritances with exceptions : an approach based on semantictolerance* IFIP88 Conference. Canton Chine (1988).
- [FER 82] J.C FERRANS : *SEDL : A language for specifying integrity constraints on office forms*. ACM SIGOA Conf. on office information systems. Phyladelphia, June 1982.
- [FIK 85] R. FIK, T. KEHLER : *The role of frame based representation in reasoning*. Communication of ACM. Vol 28, No 9. September 1985.
- [FIS 83] P. FISHER, S.T THOMAS : *Operators on Non-First-Normal-Form Relations* . Proc of the 7th Int. Computer Software Application Conference, Chicago. 1983.
- [GEH 83] N.H GEHANI : *High level form definition in office information systems*. The Computer Journal. Vol 26, No 1, 1983.
- [GIB 84] S. GIBBS : *An object-Oriented Office Data Model* . Technical Report CSRG-154. January 1984.
- [GOL 83] A. GOLDBERG, D. ROBSON : *SMALLTALK-80. The language and its implementation*. Addison Wesley Publishing Company. 1983.

- [GUT 87] R.H.GUTING, R.ZICARI, D.M.CHOY: *An algebra for structured office documents*, IBM Research Report, RJ5559, San Jose, March 87
- [HUL 87] R. HULL : *Resarch on typed complex database objects*. Databses edited by J. Paredeans. University of Antwerp VIA, Belgium.
- [JAE 82] B. JAESCHKE, H.J SCHECK : *Remarks on the Algebra of non first normal form relations*. Proc PODS, Los Angeles. 1982
- [KIT 84] H. KITAGAWA, M. GOTOH, S. MISAKI, M. AZUMA : *Form Document Management System SPECDOQ - Its Architecture and Implementation*. ACM SIGOA on Office Information System. Toronto, June 1984.
- [LEC 87] C. LECLUSE, P. RICHARD, F. VELEZ : *O<sub>2</sub>, an object Oriented Data Model*. Technical Report Altaïr 10-87. September 1987.
- [LIN 87] J. LINGAT, P. NOBECOURT, C. ROLLAND : *Behavior Management in Database Applications*. Proc of 13th VLDB Conf. Brighthon . September 1987.
- [LUM 82] V.Y LUM, D.M CHOY, N.C SHU : *OPAS : an office procedure automation system* . IBM Reserch Lab. RJ3394. San Jose, February 1982.
- [ORA 84] Oracle IAF : *Users Manual for IAF application Design*. Oracle corporation. Menlo Park. California. Original issue : October 1984.
- [PIS 86] P.PISTOR, F. ANDERSEN : *Designing a generalized NF2 model with an SQL-type language interface* . Proc 12th VLDB Conf. Kyoto. August 1986.
- [ROT 85a] M.A ROTH, H.F KORTH , A. SILBERSCHATZ : *Extended Algebra and calculus for  $\neg$ INF Relational Databases* Technical Report - Computer Science Dept. - University of Texas Austin (1985).
- [ROT 85b] M.A ROTH, H.F KORTH : *Null values in  $\neg$ INF Relational databases* . Technical Report - Computer Science Dept. - University of Texas at Austin (1985).
- [ROT 86] M.A ROTH, H.F KORTH, D.S BATORY : *SQL/INF, A query Language for  $\neg$ INF Relational Databases*. Technical Report - Computer Science Dept. - University of Texas at Austin (1986).
- [ROW 82] L.A ROWE, K.A SHOENS : *A form application development system*. Proc. ACM SIGMOD Conf. orlando, May 1982.
- [ROW 85] L.A ROWE : *"fill-in-the-form" programming* . Proc 11th. VLDB Conf. Stockholm, August 1985.
- [SCH 84] H.J SCHEK, M.H SCHOLL : *An algebra for the relational model with relation valued attributes* . Technical Report. Technische Hochschule Darmstadt. 1984.
- [SHU 83] N.C SHU, H.K WONG, V.Y LUM : *Forms Approach to Requirements Specification for Database Design*. ACM SIGMOD. International Conf. on management of data. San Jose May 1983.
- [SHU 85] N.C SHU : *FORMAL : A Forms-Oriented, Visual-Directed Application Development System* . IEE Transactions on software engineering, 1985.
- [STO 86] M. STONEBRAKER : *The Ingres Paper, Anatomy of a relational Database System* Addison-Wesley Publishing Company.
- [TSI 82] D. TSICHRITZIS : *Form management* Communications ACM. Vol 25, No 7. July 1982.
- [VAL 86] P. VALDURIEZ, S. KHOSHAFIAN, G. COPELAND : *Implementation Techniques of Complex Objects*. VLDB Conference. Kyoto, August 1986.
- [VEL 84] F. VELEZ, M.LOPEZ : *Modelling and handling generalized data in the TIGRE Project*. 8th Honeywell International Computer Science Conference Bloogmington, 1984.
- [VIG 86] P. VIGNARD : *Représentations de connaissances - mécanisme d'exploitation et d'apprentissage* . INRIA collection Didactique. 1986.
- [ZLO 77] M. ZLOOF : *Query-By-Example : a database language* . IBM systems Journal. Vol 16, No 4, 1977.
- [ZLO 82] M. ZLOOF: *Office-By-Example : a business language that unifies data and word processing and electronic mail* . IBM systems Journal. Vol 21, No 3, 1982.