Towards a Formal Approach for Object Database Design

P. Poncelet	M. Teisseire	R. Cicchetti	L. Lakhal
University of Nice-Sophia Antipolis	*Digital Equipment Ferney Voltaire*	*IUT Aix-en-Provence University of Aix-Marseille II*	*ESSTIN University of Nancy I*

I3S - CNRS - URA 1376 - 250 avenue A. Einstein -Sophia Antipolis - 06560 Valbonne - FRANCE
E-mail: poncelet@opaline.unice.fr - Tel: (33) 92 94 26 22 - Fax: (33) 92 94 28 98

Abstract

This paper focuses on a formal approach* for advanced database modeling and design. It is based on the IFO_2 model, an extension of the semantic model IFO defined by S. Abiteboul and R. Hull. It preserves the acquired strengths of the semantic approaches, whilst integrating concepts of the object paradigm. To model an IFO_2 schema, the structural part of the model including concepts such as alternative, composition, grouping for building complex objects and semantic constraints is formally specified. Furthermore, the definitions of update facilities necessary to modify and perfect IFO_2 schemas are specified through change rules. Finally, in order to design a database schema, an IFO_2 schema is translated, in an automatical way, into an existing target (implementable) model. As an illustration, we present a translation from the IFO_2 model into the O_2 one. The result is a new coherent and formal approach which is useful in overcoming some of the difficulties in the specification and design of object-oriented applications.

* This work, supported by the PRC-BD3 and an External European Research Project in collaboration with Digital Equipment, comes within the scope of a larger project whose aim is to realize an aided system for advanced application modeling and design.

Proceedings of the 19th VLDB Conference, Dublin, Ireland, 1993.

1 Introduction

Modeling needs for new applications and flaws in the relational model have led to the definition of more powerful models which are extended relational [1] or object-oriented [5], [6] and [15]. The generic term for associated systems, of which certain prototypes are described in [26], is Advanced Database Management Systems. As a consequence, current research work is focusing on the definition of new modeling and design approaches able to satisfy the needs of both traditional and advanced applications [7], [11] and [13]. The presented research work fits into this context: a new approach whose three main aspects are the following ones. Firstly, a formal object model IFO_2 [28] is defined for advanced database modeling as an extension of the semantic model IFO proposed by S. Abiteboul and R. Hull [2]. Its objective is actually to reconcile apparently opposed ideas: an optimal data representation and a complete real world modeling. IFO_2 attempts to preserve the acquired strengths of semantic approaches, whilst integrating concepts of the object paradigm [4]. Secondly, structural update primitives are formally proposed through change functions to offer an incremental specification of IFO_2 schemas. They are crucial for they assist the designer to take into account real world evolutions or to rectify a part of his schema without redefining the whole. They also play a part in the merging of existing sub-schemas and so they may be seen as one important element in a view-integration process. Finally, in order to design object database schema, a set of transformation rules translates an IFO_2 schema into an implementable one.

The aim of this paper is to describe our approach in contrast with the related works and particulary to present:

1. The structural part of the IFO_2 model.
2. The associated structural update facilities through change rules.
3. The formalization of the translation rules from an IFO_2 schema to an O_2 one (according to the

established O_2 model [16]) to justify (and illustrate) our approach.

Finally, we briefly give some aspects of the implementation of the system.

2 Related Works and Proposal

Before presenting our approach and in order to highlight its contributions, it would be interesting to provide a brief survey of modeling and design approaches. Among them, there are two main trends.

The first group involves semantic currents. They are based on conceptual (or semantic) models for real representation. Their principle is to offer the users concepts powerful enough to achieve, from the real world, the most complete specification possible. The resulting schema is then translated into a logical or implementable one. We may quote [11], [17], [27] and [30]. However, the classical models in this group generally suffer from the lack of concepts (object-identity, reusability,...) which are efficient for advanced application modeling. Furthermore, in this trend, structural updating capacities are not always proposed, and when they exist, they are described in an intuitive way.

The second class encompasses object-oriented currents. Their major goal is to capture the dynamic aspects of applications [21] and [25]. In contrast with the first class, these approaches do not offer enough structural concepts (often limited to those of implementable object models) for a complete real world modeling. Generally, additional methods are used to express semantic structural constraints. These trends do not respect the independence between the source and target models. Furthermore, they involve an optimized representation of data, i.e. type-oriented, when an attribute-oriented modeling is advisable for the conceptual level [13]. The implication for the database designer is the necessity of specifying preliminary representation choices. These choices sometimes cut off parts of the real world being modelled.

The object models provide database evolution mechanisms (three trends have been defined in [3]) but they do not deal with conceptual schemas, and their objectives differ from ours. However, they are interesting for they pinpoint two levels to be taken into consideration: the IS_A hierarchy and the composition hierarchy. For instance, we may quote: the Mosaico system where algorithms are defined for type insertions into a lattice [19]; the Esse project where algorithms ensure consistent updates of an O_2 database schema [9] [32]; the Gemstone [22] and Orion [14] systems, the Sherpa [20], Farandole2 [3] and Cocoon [29] projects where rules for the schema evolution are stated.

We would suggest an approach based on the formal model IFO_2 which is both type and attribute oriented. The IFO_2 model objective is to integrate the object paradigm whilst retaining IFO modeling strengths. It boosts modeling abilities and appears more suitable for advanced application design than object models.

To modify and perfect an IFO_2 schema, formal structural update facilities are offered. These changes are formally taken into consideration through update functions [23]. When the schema seems to be complete for the designer, it would be carried out, automatically, into a target model, by using a transformation function.

We assert that it is essential to have a really rigorous approach as IFO_2. The object paradigm allows and encourages a modular modeling of the real world. So, object modeling can sometimes look "anarchistic" and therefore difficult to handle [31]. In order to avoid such problems, a formal approach leads to a schema which is non-ambiguous, without omissions, modifiable and easily reusable. Moreover, it has the advantage of faciliting not only the comparison of different designs but also the verification of updates on specifications without further validations.

First of all, we present the IFO_2 model. Update facilities are then explained and defined through change rules.

3 The IFO_2 Model

IFO_2 adopts the philosophy of the semantic model IFO. Two main extensions are realized. Firstly, an explicite definition of the object identifier which is object value independent, is integrated. To achieve this, all manipulated elements of IFO are re-defined to consider the object paradigm. Secondly, to fully meet our "conceptual" objectives, the modeling power of IFO must be enhanced. Then, the concepts of alternative, composition and grouping for building complex objects have been integrated. The connectivity and existency constraints are explicitly specified.

In the next sections, we propose a part of formal definitions of the IFO_2 model. Instance and attached object concepts are not presented, the interested reader can refer to [28]. Firstly, the object and type concepts are described as well as the different constructors. The fragment notion and IFO_2 schema are then detailled.

3.1 Object and Type

In the IFO_2 model, an object has a unique identifier which is independant of its value. Furthermore, the domain of a type describes the possible values for its objects. The figure 1 shows the components of the type 'Name'.

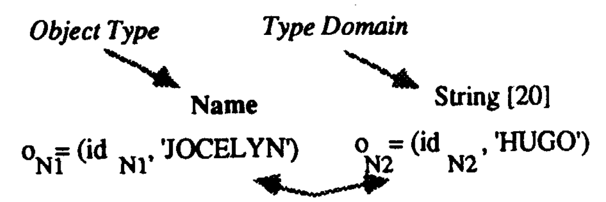


Figure 1 - Object Type Example

Definition 1: **TO** is an infinite set of *object types* such that:

$\forall \tau \in$ **TO**, **Dom** (τ) is an infinite set of symbols, including the empty set, called the **value domain** of τ, **Did** (τ) is an infinite set of symbols called the **identifier domain** of τ. Objects of type τ are defined by a pair (id, value) such that:

$\forall$ o, o' of type τ, $\exists$ (id, id') $\in$ Did$(\tau)^2$, $\exists$ (value, value') $\in$ Dom $(\tau)^2$ such that: if o=(id, value), o'=(id', value') and id $\neq$ id' then o $\neq$ o'.

The infinite set of objects of type τ is called **Obj** (τ).

3.1.1 Printable and Abstract Types

There are three basic types (shown in the figure 2):

1. A **printable** type (TOP), used for I/O application (Input/Output are therefore environment-dependent: String, Integer, Picture, Sound, ...), which are comparable to attribute type of the Entity/Relationship model [27];
2. An **abstract** type (TOA) which would be perceived as entity in the Entity/Relationship model;
3. A **represented** type (TOR), defined in the section 3.1.3, which handles another type through the IS_A specialization link. This concept is particularly interesting when considering modularity and reusability goals. The designer may defer a type description or entrust it to somebody else, while using this type for modeling a part of the application schema.

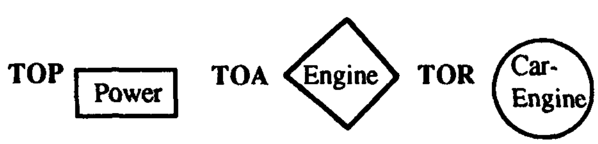


Figure 2 - Basic Type Examples

Definition 2: Let **TOP** be an infinite set of *printable types*, let **TOA** be an infinite set of *abstract types*, two disjoint subsets of **TO**, such that:

1. $\forall \tau \in$ **TOP**, dom (τ) is an infinite set of symbols;
2. $\forall \tau \in$ **TOA**, dom (τ) = {$\emptyset$}.

An abstract type actually represents an entity without internal structure but nevertheless identifiable and having properties, hence its value domain is empty.

3.1.2 Complex Types

The IFO_2 model takes into account five type constructors and makes a distinction between an exclusive and a non-exclusive building. These constructors may be recursively applied according to specified rules for building more complex types.

For example (see the figure 3), 'Address' is built up from 'Street', 'Number' and 'Zipcode' types and 'Wheels' is composed with the 'Wheel' type obtained from 'Axle' and 'Tyre' types.

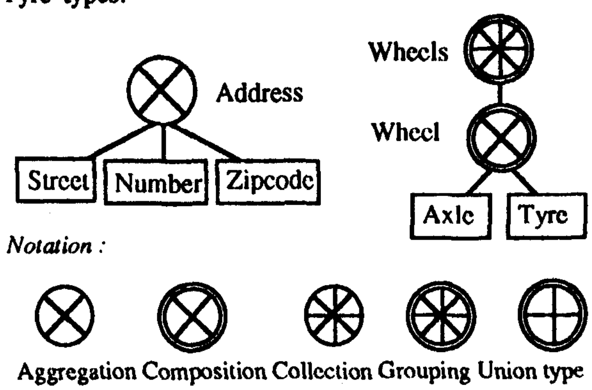


Figure 3 - Type constructors

Aggregation and Composition Types

Aggregation and composition represent the aggregation abstraction of semantic models [12] defined by the Cartesian product. It is a composition, if and only if, each object of an aggregated type occurs only once in an object construction of aggregation type.

Definition 3: Let **TOTA** be an infinite set of *aggregation types*, let **TOTC** be an infinite set of *composition types*, two disjoint subsets of **TO**, such that:

$\forall \tau \in$ **TOTA** $\cup$ **TOTC**, $\exists$ $\tau_1, \tau_2, ..., \tau_n \in$ **TO**, n > 1, such that:

Dom (τ) $\subseteq$ Obj (τ_1) x Obj (τ_2) x ... x Obj (τ_n),

τ is structurally defined as:

$\forall$ o $\in$ Obj (τ), $\exists$ $o_1 \in$ Obj (τ_1), $o_2 \in$ Obj (τ_2), ..., $o_n \in$ Obj (τ_n) such that:

o = (id, [$o_1, o_2, ..., o_n$]);

if $\tau \in$ **TOTC** then $\forall$ o' $\in$ Obj(τ) with o $\neq$ o $\exists$ $o'_1 \in$ Obj(τ_1), $o'_2 \in$ Obj(τ_2), ..., $o'_n \in$ Obj(τ_n) such that o' = (id', [$o'_1, o'_2, ..., o'_n$]) with $\forall$ i $\in$ [1..n], $o_i \notin$ {$o'_1, o'_2, ..., o'_n$}.

Collection and Grouping Types

They represent the *set-of* constructor of object models with an exclusivity constraint for the grouping.

Definition 4: Let TOSC be an infinite set of *collection types*, let TOSG be an infinite set of *grouping types*, two disjoint subsets of TO, such that:

$\forall \tau \in$ TOSC $\cup$ TOSG, $\exists ! \tau' \in$ TO such that:
Dom $(\tau) \subseteq$ P(Obj (τ) where P(Obj (τ')) is the powerset of Obj (τ),
τ is structurally defined as:
$\forall o \in$ Obj (τ), $\exists\ o_1, o_2, ..., o_n \in$ Obj (τ') such that:
$o = (id, \{o_1, o_2, ..., o_n\})$
if $\tau \in$ TOSG then $\forall o' \in$ Obj (τ) with $o \neq o$
$\exists\ o'_1, o'_2, ..., o'_n \in$ Obj (τ') such that:
$o' = (id', \{o'_1, o'_2, ..., o'_n\})$ with $\forall i \in [1..n]$,
$o_i \notin \{o'_1, o'_2, ..., o'_n\}$.

Alternative Types (Union Types)

Stucturally different types can be handled in a uniforme way through the alternative type concept. This constructor represents the IS_A generalization link enhanced with a disjunction constraint between the generalized types.

Definition 5: Let TOUT be an infinite set of *union type types*, a subset of TO, such that:

$\forall \tau \in$ TOUT, $\exists\ \tau_1, \tau_2, ..., \tau_n \in$ TO, $n > 0$ such that:
Dom $(\tau) \subseteq$ Dom $(\tau_1) \cup$ Dom $(\tau_2) \cup ... \cup$ Dom (τ_n),
τ is structurally defined as:
$\forall i, j \in [1..n]$ if $i \neq j$ then
Obj $(\tau_i) \cap$ Obj $(\tau_j) = \emptyset$, Obj (τ) = Obj $(\tau_1) \cup$ Obj $(\tau_2) \cup ... \cup$ Obj (τ_n),
with $\forall o \in$ Obj (τ), $\exists !\ k \in [1..n]$ such that:
$o = o_k$, $o_k \in$ Obj (τ_k).

3.1.3 Represented Types

The definition of represented types takes into account the multiple inheritance since a represented type may have several sources.

Definition 6: Let TOR be an infinite set of *represented types*, a subset of TO, such that:

$\forall \tau \in$ TOR, $\exists\ \tau_1, \tau_2, ..., \tau_n \in$ TO, $n > 0$ called source(s) of τ such that , Obj $(\tau) \subseteq$ Obj $(\tau_1) \cup$ Obj $(\tau_2) \cup ... \cup$ Obj (τ_n) with $\forall o \in$ Obj (τ), $\exists\ o_i \in$ Obj (τ_i) such that $o = o_i$.

3.1.4 Types

From basic types and constructors, it is possible to define a type, as a tree, in a general way.

Definition 7: A *type* $T \in$ TO is a directed tree $T = (S_T, E_T)$, where E_T is a set of type edges. T is such that:

1. The set of vertices S_T is the disjoint union of eight sets TOP, TOA, TOR, TOTA, TOTC, TOSC, TOSG, TOUT.
2. If $T \in$ TOA then T is root of type.
3. The leaves of the tree are printable or represented types.

An abstract type cannot be used in a built type since its role is to describe a real world entity which is not defined by its internal structure but through its specified fragment properties.

3.2 IFO2 Fragment

The types could be linked by functions (simple, complex (i.e. multi-valued), partial (0:N link) or total (1:N link)) through the **fragment** concept. The aim of the fragment is to describe properties (attributes) of the principal type called heart. The figure 4 describes the fragment of heart 'Person' having 'Name', 'Address' and 'Vehicle' as properties. For each vehicle associated to a person, there is a contract insurance number, this is called a nested fragment. First Names are not always known for a person.

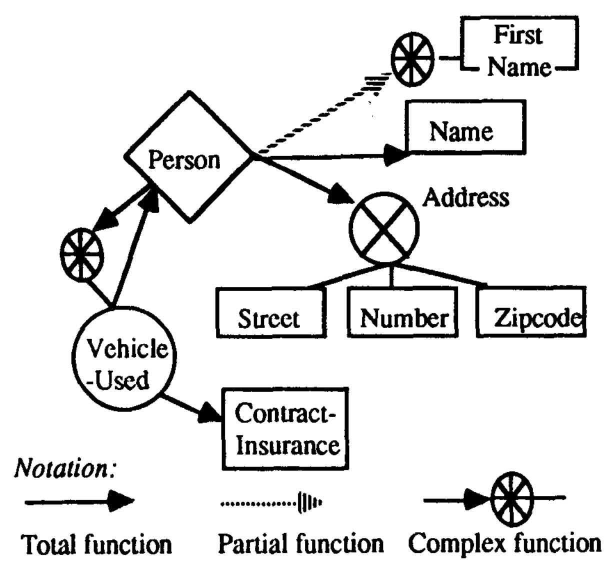


Figure 4 - The Fragment 'Person'

Conventions: we call partial a function in which some elements of the domain have no associated elements in the codomain. Otherwise, it is called total. The kind of handled graph is: G = (X, U) where the set of vertices X is the set of types T of TO and the set of edges U is composed with: simple edges (simple functions) and complex edges (functions applied on a TOSC, called complex functions: an image of an object is a set). The edge is called either partial or total if the associated function is either partial or total.

Definition 8: An *IFO2 fragment* is a graph F = (V_F, L_F), with V_F the set of types T ∈ 𝒯𝒪 and L_F the set of fragment links, defined such that:

1. There is a direct tree H = (V_F, A) such that:
   1.1. The root of H is called heart of fragment.
   1.2. The source of an edge is either the heart root or the root of a target type of a complex edge whose source is the heart root.
2. For each edge linking the heart to a represented type, there is a reciprocal total edge.

The IFO_2 fragment is called by its heart.

3.3 IFO_2 Schema

An IFO_2 schema is composed of n IFO_2 fragments: F_1, F_2, ..., F_n, n > 0, related by IS_A links according to two rules. The figure 11 illustrates one IFO_2 schema made up with five fragments 'Person', 'Employee', 'Vehicle', 'Car' and 'Engine'. They are linked with IS_A links through the represented types ('Vehicle_Used', 'Employee', 'V_Car', 'Truck_Engine' and 'Car_Engine').

3.3.1 Specialization Link

The IS_A link in the IFO_2 model is the specialization link of the semantic models [12]. It represents either the subtyping (inheritance) if the target is a fragment heart or the client/supplier concept [18].

Definition 9: Let τ' be a type of 𝒯𝒪ℛ and let T be a type of 𝒯𝒪, such that it is the (or one) source of τ' and a heart of a fragment, the link of head T and queue τ' is called an *IS_A link*. T is called the source of the IS_A link and τ' the target.

The figure 5 illustrates the specialization link between 'Vehicle' and 'Vehicle_Used'.

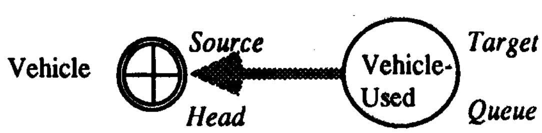


Figure 5 - Notation for Specialization Link

3.3.2 IFO_2 schema

Definition 10: An *IFO_2 schema* is defined as a graph G_S = (S_S, L_S) with S_S the set of types T ∈ 𝒯𝒪 of the graph such that:

1. L_S is the disjoint union of two sets L_{S_A} (fragment links) and $L_{S\_IS\_A}$ (IS_A links).
2. (S_S, L_{S_A}) is a forest of IFO_2 fragments, called the IFO_2 fragments for G_S.
3. (S_S, $L_{S\_IS\_A}$) follows these two schema rules:
   3.1. There is no IS_A cycle in the graph.
   3.2. Two directed paths of IS_A links sharing the same origin have to be extended to a common vertex.

The structural part of the IFO_2 model having now been formalized, we examine the supplied update facilities.

4 Updates on IFO_2 Schema

Due to space limitation, we just present updates on IFO_2 schema in an informal way. The interested reader may find more details in [23] where a functional approach is defined to formally ensure the structural consistency of IFO_2 updates.

4.1 Motivation

The problem with schema updates can be summarized by: how to modify a given schema whilst preserving a coherent representation? In other terms, our aim is to ensure that updates retain the schema consistency. In object model, consistency can be classified in structural consistency which refers to the static part of the database and in behavioral consistency relating with the dynamic part [32]. In this paper, we only deal with the structural case.

An IFO_2 schema is a couple (S_S, L_S) where L_S is composed by both fragment and IS_A links but not every arbitrary couple (S_S, L_S) is a correct schema. Thus, we have to make sure that the result of modifications is an updated schema which verifies the IFO_2 schema definition (*correctness*). Therefore, we give a set of schema invariants which are conditions to be satisfied by any valid schema. A similar approach is adopted by models such as Orion, O_2, Gemstone, Cocoon and Sherpa.
Some schema changes are quite simple, whereas others need a complete reorganization of the database. The latter can often be decomposed into a sequence of more elementary changes. The following taxonomy, figure 6, presents the schema update primitives in IFO_2, which is minimal and complete in the sense that all possible schema transformation can be built up by a combination of these elementary operations (*completeness*). Such a taxonomy can be found in models like Orion, Sherpa and Cocoon. The two former give three categories of operations: changing class definitions, i.e. instance variable or methods, modifying the class lattice by changing the relationships between classes and adding or deleting classes in the lattice. As the latter is based on type, function and classes, schema changes are respectively: type updates, function updates and class updates.

All schema structure changes, as for instance, a fragment insertion into the directed acyclic graph, may be expressed by a sequence of basic updates. For example, the fragment insertion may be done by: <(1.1) a type insertion, (3.1) zero or more IS_A link insertion and finally, (3.2) zero or more IS_A link deletion (in the case of a node insertion into the direct acyclic graph)>. The primitive (1.4) is necessary to preserve the schema in a valid state for it is not equivalent to the sequences <(1.2) (1.1)> or <(1.1) (1.2)> when the type has to be related to other ones. The consequence of applying such sequences may occur in a temporary invalid state: as instance, if we want to substitute a grouping component using <(1.2) (1.1)> or <(1.1) (1.2)>, the application of (1.2) (respectively (1.1)) carries out the schema in a forbidden state: a grouping without component (respectively a grouping with two components).

(1) Types updates
(1.1) Add a new type
(1.2) Delete a type
(1.3) Change a type
(1.3.1) its name
(1.3.2) its domain for printable type only
(1.4) Substitute a type

(2) Fragment link updates	(3) IS_A link updates
(2.1) Add a new fragment link	(3.1) Add a new IS_A link
(2.2) Delete a fragment link	(3.2) Delete an IS_A link
(2.3) Change the sort of fragment link	

Figure 6 - Taxonomy of possible updates in IFO2

Intuitively, in IFO2, a schema update is either a type insertion or a type modification in a fragment. The former case is defined as a type insertion which must be related to the schema. We can create a fragment, add a type to a fragment or relate a type to others. The latter is described with one or more operations on the concerned fragments which are themselves modifications on types. Operations like insertion of a sub-type into an existing one, deletion of a type and substitution of one type by another are thus possible.

4.2 Presentation

We present the schema invariants that the transformation process must maintain, and the necessary rules to provide a guidelines for supporting schema modifications. As we have just discussed, two rule categories have to be taken into consideration: insertion and modification.
In this section, we illustrate the introduced concepts using the following schema which is a sub-part of the figure 11.

4.2.1 Evolution Schema Invariants

The following schema invariants ensure that the change does not leave the updated schema in an inconsistent state. If the change would violate the invariants, it is rejected.

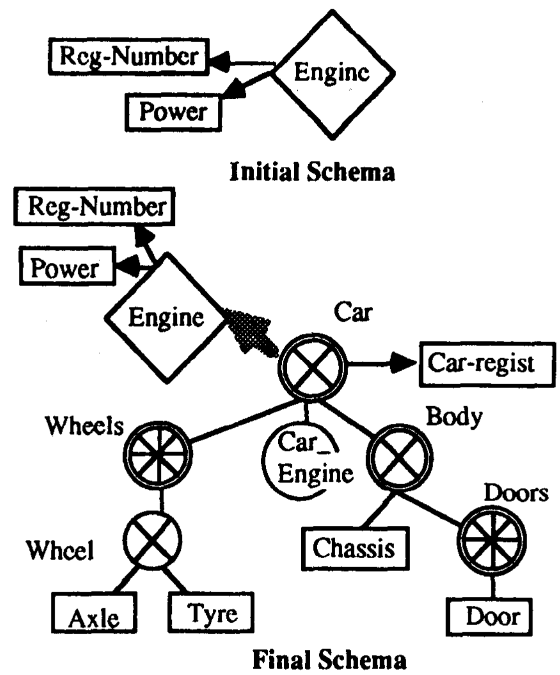


Figure 7 - An IFO2 fragment creation example

Invariants:

I.1: a type T has to follow the definitions 1 to 7 of an IFO2 type.

I.2: in the graph:
1. There is no IS_A cycle.
2. Two directed paths of IS_A links sharing the same origin have to be extended to a common vertex.

I.3: the source of an IS_A link must be a fragment heart and the target a represented type.

I.4: the source of an edge is either the heart root or the root of a target type of a complex edge whose source is the heart root and for each edge linking the heart to a represented type, there is a reciprocal total fragment edge.

I.5: a fragment cannot be isolated (except if it is the unique one), i.e. it has to be related to other ones through IS_A links.

For example, the type insertion of a Wheel's brother could not be possible because the invariant I.1 is not satisfied (a grouping has only one component).

4.2.2 Insertion Rules

A type insertion into a schema is either a fragment creation or a property insertion into an existing fragment. The insertion rules have to respect the schema invariants and therefore, some insertions are forbidden. For instance, the addition of the type 'Car' without adding the IS_A link from 'Engine' to 'Car-Engine' violates the invariants I.3 and I.5. The fragment properties may often be modified, so the following rules provide a guideline for supporting changes into the fragment.

Rule 1: Addition of a type into a fragment

R1.1: addition of the type (**primitive 1.1**).

R1.2: addition of the fragment link relating the fragment heart to the type (**primitive 2.1**).

Rule 2: Addition of a represented type or a type built up with represented types

R2.1: addition of the type into the appropriate fragment (**rule 1**).

R2.2: if the type is a represented one, addition of the reciprocal total fragment link (**primitive 2.1**).

R2.3: addition of the IS_A link(s) whose represented type(s) is(are) target(s) (**primitive 3.1**).

R2.4: if the type is a represented one and a fragment heart, addition of the IS_A link(s) which it is the source of (**primitive 3.1**).

R2.5: if the type is a represented one and a fragment heart, deletion of the IS_A link(s) relating the R2.3 source vertices to the R2.4 target vertices (**primitive 3.2**).

For example, the type insertion of 'Car' has to connect the type 'Car-Engine' to 'Engine'. As 'Car' is a leaf of the directed acyclic graph, there are no represented types whose source is 'Car' (rule 2). The schema components are thus obtained as follows. The original set of schema vertices {Engine, Power, Reg-Number} is increased with the 'Car' vertices {Car, Car_Engine, Wheels, Wheel, Axle, Tyre, Body, Chassis, Doors, Door} applying the R.1.1 statement. The fragment link set is not modified because the inserted type is a fragment heart. The application of R2.3 provides the IS_A link set composed by the link relating 'Car-Engine' to 'Engine'.
Now, the addition of 'Car-regist' as a 'Car' fragment property, following the rule 1, updates schema components such as: the type 'Car-regist' increases the set of schema vertices according to the R1.2 statement; the fragment link set is updated for 'Car-regist' is related to 'Car' and the IS_A link set is not modified because the inserted type is not a represented one.

4.2.3 Modification Rules

Schema invariant constraints can be violated by modifications. It is thus necessary to define rules to obtain a valid updated schema.
The following rule is used to prevent that the invariant I.5 is controlled.

Rule 3: IS_A link deletion condition

R3.1: an IS_A link can be deleted if and only it does not carry out one isolated part in the resulting schema.

Rule 4: Deletion of a type vertex

R4.1: if the vertex is a represented type, deletion of IS_A link(s) which it is the target of (**primitive 3.2**) and (**rule 3**).

R4.2: if the father is a grouping or a collection then deletion of the father vertex (**rule 4**) else if the father is an aggregation, a type union or a composition <u>and</u> there is a unique brother then substitution of the father type by the brother one (**primitive 1.4**) else deletion of the type whose root is the vertex (**primitive 1.2**).

Consider the deletion of the type 'Tyre'. As 'Wheel' is a composition of two elements, the deletion of 'Wheel' would provide an inconsistent type (a composition of a unique element violates the invariant I.1). The updates have thus to be send back in the 'Wheel' father level substituting the father type by the Tyre's brother (R.4.2). The schema vertex set is thus decreased with {Wheel, Tyre}. The following figure shows the updated type:

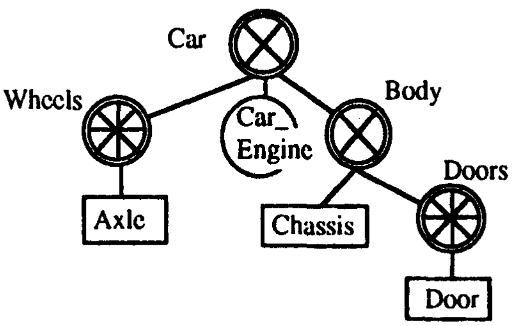


Figure 8 - The resulting type after type deletion

Rule 5: Deletion of a type into a fragment

R5.1: deletion of its related types which are not fragment heart (**rule 4**).

R5.2: deletion of the fragment links relating it to the previous types R5.1 (**primitive 2.2**).

R5.3: deletion of the fragment link relating the fragment heart to the type (**primitive 2.2**).

R5.4: deletion of the type (**primitive 1.2**).

A type deletion provides necessary operations deleting fragment links related to and related from the type (rule 5). For instance, the 'Car-regist' deletion needs to delete the fragment link from 'Car' to 'Car-regist' (R5.3) of the schema fragment link set.

Rule 6: Deletion of a represented type or a type built up with represented types into a fragment

R6.1: deletion of the IS_A link(s) whose represented type(s) is/are target(s) (**primitive 3.2**).

R6.2: if the type is a represented one and a fragment heart, deletion of the IS_A link(s) which it is the source of (**primitive 3.2**) and (**rule3**).

R6.3: if the type is a represented one and a fragment heart, addition of the IS_A link(s) relating the R6.1 source vertices to the R6.2 target vertices (**primitive 3.1**).

R6.4: if the type is a represented one, deletion of the fragment links relating it to the fragment heart (**primitive 2.2**).
R6.5: deletion of the type in the appropriate fragment (**rule 5**).

When a type is a represented one or is built up with represented ones, its deletion changes the schema IS_A links. This is done through the rule 6. For example, the deletion of the type 'Car' has to delete the IS_A link between 'Car_Engine' and 'Engine' (R.6.1).

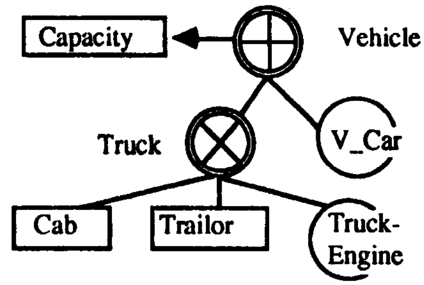
As the IFO2 model and its update capacities are now defined, we will examine the IFO2 translation into the O2 model [16].

5 Mapping an IFO2 Schema to an O2 Schema

This mapping follows the same principle as the realized transformation from MORSE to O2 [7]. As this application of a principle frame is on the "whole-object" from the "source" model level, the ways of translation are different. Therefore, we define checking methods and associated classes for composition, grouping and union type. We also consider multiple inheritance generically.
The formalization of the mapping will be given after having introduced it through an example.

5.1 Illustration

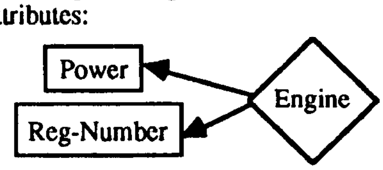
An IFO2 schema is translated into an O2 schema. Each fragment generates at least one O2 class (more if they are composition, grouping or alternative types).
We translate a part of the IFO2 schema described in figure 11 into an O2 schema. Therefore, we work successively on the two fragments "Vehicle" and "Engine".
The "Vehicle" fragment is mapped into a particular class. It represents either a "Truck" element or a "Car" element. To do this, adopting a similar principle as in [8], we use two boolean attributes "is_Truck" and "is_Car" which indicate the object type. We also need to define the classes "C_Truck", "C_Cab" and "C_Trailor" so that each object of these types has an identifier. Furthermore, a method checks the exclusive composition constraint in the class "C_Truck". As the represented types are not fragment heart, they are translated using the O2 composition. Otherwise, the associated class has to inherit the generic class: for instance, the class employee inherits the class Person.




```
class C_Vehicle
public type tuple (
  Vehicle : tuple (
    is_Truck : boolean, Truck : C_Truck,
    is_V_Car : boolean, V_Car : C_Car),
    Capacity : integer)
end;
class C_Truck
public type tuple (
          Truck : tuple ( Cab : C_Cab, Trailor : C_Trailor,
           Truck-Engine : C_Engine))
method Check_Compo_Truck : boolean
end;
class C_Trailor
public type tuple (Trailor : string) end;
class C_Cab
public type tuple (Cab : string) end;
```

Figure 9 - The fragment "Vehicle" and the associated O2 Classes

The "Engine" fragment is translated into a class with two attributes:




```
class C_Engine
public type (Power : string,
             Reg-Number : integer);
```

Figure 10 - The fragment "Engine" and the associated O2 Class

5.2 Transformation Functions

We suggest to define the transformation function which carries out an O2 schema from an IFO2 schema.

5.2.1 Transformation Function

Definition 11: Let $\mathcal{T}$ be the *transformation function* from the IFO2 model to the O2 one:

$$\mathcal{T}: IFO2 \rightarrow O2$$

$$\mathcal{T}(\text{IFO2 schema}) = \bigcup_{j=1}^{m} \mathcal{T}(\text{Fragment}_j)$$

where m is the fragment number of the IFO2 schema.

This transformation uses a $\mathcal{F}$ function (with three parameters: the function source type, the function target type and the kind of function (simple (partial or total) or complex (partial or total)*) which translates each fragment edge into the O_2 model. We adopt the following notation: for each t type of $\mathcal{TO}$, C_t is a class name and t is an attribute name in the O_2 model.

Definition 12: Let n be the edge number whose source is the fragment heart, $\mathcal{T}$ is defined as:

$\mathcal{T}$ (Fragment) =

1. if heart $\notin \mathcal{TOA} \cup \mathcal{TOR}$:
   class C_heart
   public type tuple (heart: $\mathcal{F}$(heart, □, □)

2. if heart $\in \mathcal{TOA}$:
   class C_heart public type tuple (

3. if heart $\in \mathcal{TOR}$:
   /* s is the number of IS_A links */
   class C_heart inherit C_t_1, ..,C_t_s
   public type tuple (

◊ $\overset{n}{\underset{i=1}{}}$ $target_i$: $\mathcal{F}$(heart, $target_i$, simple|complex))

◊ if heart $\in \mathcal{TOTC}$:
method public Check_Compo_heart : boolean

◊ if heart $\in \mathcal{TOTG}$:
method public Check_Set_heart : boolean

◊ if $target_i \in \mathcal{TOR}$ ∧ ($target_i$, heart, simple)
method public Check_Simple_$target_i$($\mathcal{F}$($target_i$, □, □)): boolean

∧ ($target_i$, heart, complex):
method public Check_Complex_$target_i$ ($\mathcal{F}$($target_i$, □, □), integer): boolean

◊ if $target_i \in \mathcal{TOTC}$
method public Check_Compo_$target_i$: boolean

◊ if $target_i \in \mathcal{TOTG}$:
method public Check_Set _$target_i$: boolean

◊ *end;*

---

* For the general case, partial and total terms will be omitted.

5.2.2 Types and Edges Transformation Function

Definition 13: The *$\mathcal{F}$ function* translates fragment types either in O_2 type or in O_2 class. It is defined as:

◊ if t $\in \mathcal{TOP}$:
$\mathcal{F}$(t, □, □) = *dom (t)*

◊ if t $\in \mathcal{TOR}$ ∧ t has only one source:
$\mathcal{F}$(t, □, □) = *C_t'* /* t' is the source of the IS_A link */

∧ t has m sources:
$\mathcal{F}$(t, □, □) = *C_t* /* there are m sources of IS_A links */
∧ *class C_t inherit C_t_1, ..., C_t_m*
/* in case of name conflict, O_2 is able to solve it by adding a prefix to the class name */

◊ if t $\in \mathcal{TOA}$:
$\mathcal{F}$(t, □, □) = *tuple* ($\overset{k}{\underset{i=1}{}}$ t_i: $\mathcal{F}$(t_i, □, □)) /* k is the number of related types */

◊ if t $\in \mathcal{TOTA}$:
$\mathcal{F}$(t, □, □) = *tuple* ($\overset{k-n}{\underset{i=1}{}}$ t_i: C_t_i , $\overset{n}{\underset{j=1}{}}$ t_j: $\mathcal{F}$(t_j, □, □))
/* k is the number of aggregated types and n the number of $\mathcal{TOR}$ types */

$\overset{k}{\underset{i=1}{\wedge}}$ *class C_t_i public type tuple* (t_i: $\mathcal{F}$(t_i, □, □)
if $t_i \in \mathcal{TOTC}$:
method public Check _Compo_t_i: boolean
if $t_i \in \mathcal{TOTG}$:
method public Check _Set_t_i: boolean
end;

◊ if t $\in \mathcal{TOTC}$:
$\mathcal{F}$(t, □, □) = *set* ($\mathcal{F}$(t', □, □)) /* t' is the collected type */

◊ if t $\in \mathcal{TOTG}$:
$\mathcal{F}$(t, □, □) = /* t' is the collected type */
if t' $\in \mathcal{TOR}$: *set* ($\mathcal{F}$(t', □, □))
if t' $\notin \mathcal{TOR}$: *set* (C_t')
∧
class C_t' public type tuple (t': $\mathcal{F}$(t', □, □))
if t' $\in \mathcal{TOTC}$:
method public Check_Compo_t': boolean
if t' $\in \mathcal{TOTG}$:
method public Check_Set_t': boolean
end;

◊ if $t \in$ TOUT:

$\mathcal{F}(t, \square, \square) = tuple\,(is\_t_i: boolean, t_i: C\_t_i)$ with $i=1 \ldots k$

$\wedge$ $class\ C\_t_i$ ($i=1 \ldots k$) /* k is the number of associated types */

public type tuple ($t_i: \mathcal{F}(t_i, \square, \square)$)

if $t_i \in$ TOTC:

method public Check_Compo_*t_i: boolean*

if $t_i \in$ TOTG:

method public Check_Set_*t_i: boolean*

end;

◊ $\mathcal{F}(heart, target, simple) = \mathcal{F}(target, \square, \square)$

For a complex edge, the target can be the source of p other edges (nested fragment) with $target2_i \neq heart\ \forall i \in [1,..,p]$ (edges whose target is the heart, have already been translated):

◊ $\mathcal{F}(heart, target, complex) =$

set (tuple(target: $\mathcal{F}(target, \square, \square)$, $target2_i: \mathcal{F}(target, target2_i, simple|complex)$))) with $i=1 \ldots p$

◊ if $\mathcal{F}$*(heart, target, simple total|complex total)* *then add an existence test for attribute target value in the Init heart method.*

◊ $\mathcal{F}(target, target2, simple) = \mathcal{F}(target2, \square, \square)$

$\mathcal{F}(target, target2, complex) =$ $set(\mathcal{F}(target2, \square, \square))$

◊ if $\mathcal{F}$*(target, target2, simple total|complex total)* *then add an existence test for attribute target2 value in the Init heart method.*

5.2.3 Structures of Generated Methods

This part describes the structure of some generated methods. These methods are added into classes as semantics constraint checkers.

Generally, the method checking the exclusive constraint for composition of a type $t \in$ TOTC is:

```
method body Check_Compo_t boolean in class C_t {
  /* k is the number of component types */
  if (!(self->Test_compo1(self->t.compo1)))
  return false; ...
  if (!(self->Test_compok(self->t.compok) return false;
  return true;};
```

The method verifying that one element belongs to an object of class *C_Class* is *:

```
method body Test_compoj (object: C_compoj):
boolean in class C_Classe
{ o2 C_Classe obj;   /* compoj is a component of type t */
  o2 boolean result=true;
for (obj in C_Classe_s where obj->C_Classe.compoj
== object) {result=false;}; return result;};
```

In the same way, the following method verifies the grouping of a type t whose collected type is t':

```
method body Check_Set_t : boolean in class C_t
{o2 C_t obj, C_t' obj_t'; /*test member of the set */
 o2 boolean result=true;
for (obj in C_t_s)
{for (obj_t' in obj->t where obj_t' in self->t )
{result=false;};
return result;};
```

The other checking methods are defined in the same way.

6 Implementation

In this section, we briefly indicate some aspects of the implementation of the IFO_2 system.

A first version of the IFO_2 editor is currently developped under Unix/XWindow (X11R5), with the help of the Aida/Masai (Release 1.5) programming environment, developped in object-oriented Le-Lisp (Release 15.24).

This editor, as illustred in figure 11, is made up of three tools:

1. A graphical view consisting of an editing panel, a tool panel and a workspace;
2. A selection panel of object types and existing link types;
3. An object editor enabling the textual representation of textual object as well as information which does not appear on the schema.

* For a C_Classe class, the C_Classe_s value is the instances set of the C_Classe class. For each new instance of C_Classe, it is necessary to modify the set which is defined as: Name C_Classe_s: set(C_Classe).

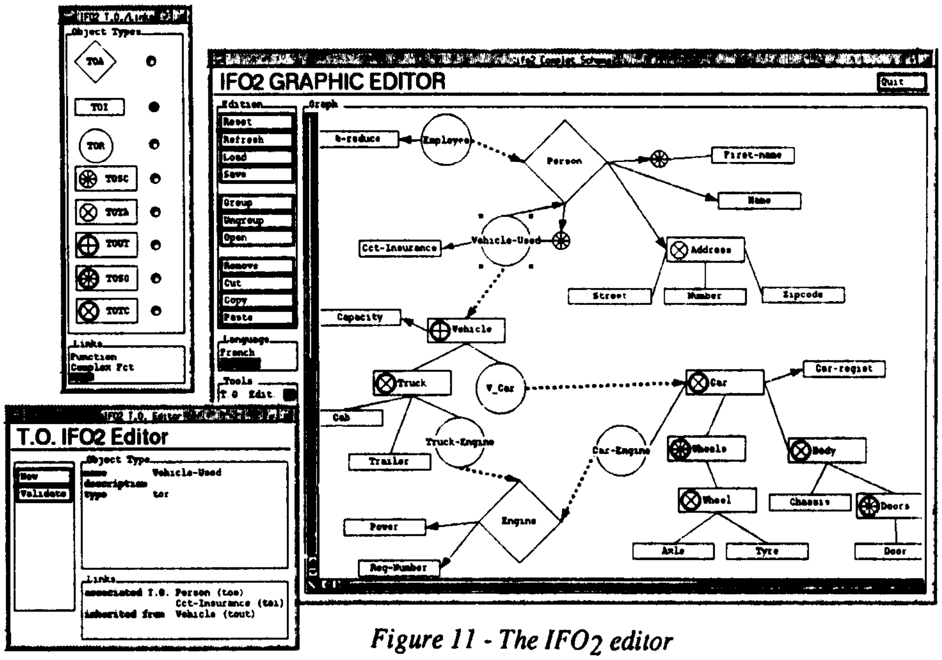


Figure 11 - The IFO$_2$ editor

The type definitions achieved with the editor are converted by a translator into O$_2$ descriptions. The descriptions may thus be used in the O$_2$ system. For instance, the methods described in the previous section have been implemented in the O$_2$ Database Management System (Release 3.3). The following example shows some generated methods associated to the class C_Truck:

```
method body Check_Compo_Truck: boolean in class C_Truck
{ if (!(self->Test_Cab(self->Truck.Cab))) return false;
if (!(self->Test_Trailor(self->Truck.Trailor))) return false;
if (!(self->Test_Truck_Engine(self->Truck.Truck_Engine)))
return false; return true; };
method body Test_Cab (object: C_Cab): boolean in class
C_Truck
{ o2 C_Truck obj;
  o2 boolean result=true;
  for (obj in C_Truck_s where obj->Truck.Cab == object)
{result=false;}; return result; };
/* named values declaration */
name C_Truck_s: set (C_Truck);
```

7 Conclusion

In this paper, we have formally defined an object model IFO$_2$ as well as its transformation into the O$_2$ model. We have also presented the offered updates facilities in an informal way. As a conclusion, first of all we would like to highlight its contributions so as to indicate the prospects of this work.

The first contribution, that of the IFO$_2$ whole-object, is the coherent and rigorous definition of the component elements of the model through the object identity concept.

The second strength of the model is the integration of constructors which are indispensable to the development of advanced applications, such as composition and grouping. The latter enables the constituant sets to be "physically" taken into account.

The most original aspect of IFO$_2$ is that it draws upon both elements which may be said conceptual, such as fragments and represented types, and implementable such as object identifiers. The case of multiple inheritance is a special case given that, at the conceptual level, no conflicts are involved while at the system level, all conflicts generated are explicitly processed. We have seen that IFO$_2$ inheritance may be multiple but does not require any prior management. The conflicts are processed according to the target model while the translation rules are defined.

Another advantage of IFO$_2$ is the way it can modulate and reuse parts of schema that have been developped, through the fragment concept. Therefore, it is possible to focus on only one part of the schema while reusing, through represented types, the already defined and validated components.

The fragment concept represents another advantage of IFO$_2$: namely the ease of integrating application dynamic through this structure. It enables the behavior of the heart type to be described naturally and above all makes it possible for behavior to be inherited through represented types.

Finally, IFO$_2$ is totally independent in relation to implementable models, while providing an ease of transformation rule definition towards different models due to its genericity. The translation of an IFO$_2$ schema into an O$_2$ one is a prime example of this. The formal rule definitions reduce data-loss and misinterpretation.

The presented update capacities is a strength of our approach. They ensure the integrity of the updated schemas. The result is a coherent and formal approach. The ambiguities and contradictions are then detected and different schemas may be compared. Furthermore, in a reusability goal, the security obtained through the consistency of handled informations is crucial.

The prospects of the presented work begin with the integration of modeling abilities for the application dynamic. The conceptual rules associated with the IFO_2 model advocate an attribute-oriented modeling and are principally based on the object behavior. Moreover, through "process" specification associated with the fragment, the most suitable optimized representation can be determined. According to us, dynamic and behavior will be integrated in the model using a formal approach based on the temporal logic [10] and [24]. Such an approach automatically validates the specified constraints whilst being easily understood by the users.

8 References

[1] Abiteboul S., Fischer P. C. and Schek H.-J. (Eds) - Nested relations and complex objects in databases. LNCS, Vol. 361, Springer-Verlag, 1989.

[2] Abiteboul S. and Hull R. - *IFO: A Formal Semantic Database Model*. ACM Transaction on Database Systems, Vol. 12, N° 4, December 1987, pp. 525-565.

[3] Andany J., Léonard M. and Palisser C. - *Management Of Schema Evolution in Databases* - Proceedings of the 17th International Conference on Very Large Data Bases, Barcelona, September 1991, pp. 161-170.

[4] Atkinson M.P., Bancilhon F., DeWitt D., Dittrich K., Maier D. and Zdonik S. - *The Object-Oriented Database System Manifesto*. Proceedings of the First Deductive and Object-Oriented Database - DOOD89 Conference, Kyoto, Japan, December 1989, pp. 223-240.

[5] Banerjee J., Chou H.-T., Garza J.F., Kim W., Woelk D., Ballou N. and Kim H. J. - *Data Model Issues for Object-Oriented Applications*. ACM Transactions on Office Information Systems, Vol. 5, N° 1, January 1987, pp. 3-26.

[6] Bertino E. and Martino L. - *Object-Oriented Database Management Systems: Concepts and Issues*. Computer IEEE, Vol 24, N°4, April 1991, pp. 33-47.

[7] Bouzeghoub M., Métais E., Hazi F. and Leborgne L. - *A Design Tool for Object Databases*. LNCS, Vol. 436, Steinholtz B., Solvberg A. and Bergman L. (Eds.), proceedings of the Second Conference on Advanced Information Systems Engineering, Springer-Verlag, 1990, pp. 365-392.

[8] Collet C. and Brunel E. - *Définition et manipulation de formulaires avec FO2*. TSI Technique et science informatique, Vol. 10, N° 2, 1991, pp. 97-124.

[9] Coen-Porisini A., Lavazza L. and Zicari R. - *The ESSE Project: An Overview*. Proceedings of the 2nd Far-East Workshop on Future Database systems, Kyoto, Japan, April 26-28, 1992, pp. 28-37.

[10] Fiadeiro J. and Sernadas A. - *Specification and Verification of Database Dynamics*. Acta Informatica, Vol. 25, 1988, pp. 625-661.

[11] Heuer A. - *A Data Model for Complex Objects Based on a Semantic Database Model and Nested Relations*. In [AbFi89], pp. 297-311.

[12] Hull R. and King R. - *Semantic Database Modelling: Survey, Applications, and Research Issues*. ACM Computing Surveys, Vol. 19, N° 3, September 1987, pp. 201-260.

[13] Hull R. - *Four Views of Complex Objects: A Sophisticate's Introduction*. In [AbFi89], pp. 87-116.

[14] Kim W., Bertino E. and Garza, J.F. - *Composite Objects Revisited*. Proceedings of the ACM SIGMOD Conference, June 1989, pp. 337-347.

[15] Kim W. - *Object-Oriented Databases: Definition and Research Directions*. IEEE Transactions on Knowledge and Data Engineering, Vol. 2, N° 3, September 1990, pp. 327-341.

[16] Lécluse C., Richard P. and Velez F. - *O2, An Object-Oriented Data Model*. Proceedings of the ACM SIGMOD Conference, June1988, pp. 424-433.

[17] Lyngbaek P. and Vianu V. - *Mapping a Semantic Database Model to the Relational Model*. Sigmod Record, Vol. 16, N° 3, December 1987, pp. 132-142.

[18] Meyer B. - Object-Oriented Software Construction - Prentice Hall International series in Computer Science, 1988.

[19] Missikoff M. and Scholl M. - *An Algorithm for Insertion into a Lattice: Application to Type Classification*. Foundations of Data Organization and Algorithms, LNCS, Vol. 367, Springer-Verlag, 1989, pp. 64-82.

[20] Nguyen G.T. and Rieu D. - *Schema Evolution in Object-Oriented Database Systems*. Data&Knowledge Engineering (North Holland) 4, 1989, pp. 43-67.

[21] Pernici B. - *Objects with Roles*. Conference on Office Information Systems, Cambridge, April 1990, pp. 205-215.

[22] Penney D.J. and Stein J. - *Class Modification in the Gemstone Object-Oriented DBMS*. OOPSLA'87, Proceedings, October 1987, pp. 111-117.

[23] Poncelet P. and Lakhal L. - *Consistent Structural Updates for Object Database Design*. LNCS, proceedings of the Fifth Conference on Advanced Information Systems Engineering, Paris, June 1993, to appear.

[24] Saake G. - *Descriptive Specification of Database Object Behaviour*. Data & Knowledge Engineering (North Holland) 6, 1991, pp. 47-73.

[25] Sakai H. - *An Object Behavior Modeling Method*. Proceedings of the 1st International Conference on Database and Expert Systems applications, 1990, pp. 42-48.

[26] Stonebraker M. (Ed.) - Special Issue on Database Prototype Systems. IEEE Transactions on Knowledge and Data Engineering, Vol. 2, N°1, March 1990.

[27] Teorey T. J.(Ed.) - The Entity-Relationship Approach. Morgan Kaufmann Publishers, 1990.

[28] Teisseire M., Poncelet P. and Cicchetti R. - *A Tool Based on a Formal Approach for Object-Oriented Database Modeling and Design*. Proceedings of the 6th International Workshop on Computer-Aided Software Engineering (CASE'93), IEEE Publisher, Singapore, July 1993, to appear.

[29] Tresch M. and Scholl M.H. - *Meta Object Management and its Application to Database Evolution*. Proceedings of the 11th International Conference on the Entity-Relationship Approach, LNCS N°645, Karlsruhe, Germany, October 1992, pp. 299-321.

[30] Twine S. - *Mapping between a NIAM conceptual schema and KEE frames*. Data & Knowledge Engineering, Vol. 4, N° 4, December 1989, pp. 125-155.

[31] Unland R. and Schlageter G. - *Object-Oriented Database Systems: Concepts and Perspectives*. Database Systems of the 90s, LNCS, Vol. 466, Springer-Verlag, 1990, pp. 154-197.

[32] Zicari R. - *A Framework for Schema Updates in an Object-Oriented Database System*. Proceedings of the 7th IEEE Data Engineering Conference, Kobe, Japan, April 8-12, 1991, pp. 2-13.