Colette Rolland François Bodart
Corine Cauvet (Eds.)

Advanced Information Systems Engineering

5th International Conference, CAiSE '93
Paris, France, June 8-11, 1993
Proceedings

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
W-7500 Karlsruhe, FRG

Juris Hartmanis
Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca, NY 14853, USA

Volume Editors

Colette Rolland
François Bodart
Corine Cauvet
C.R.I., Université de Paris I, Panthéon-Sorbonne
17 Rue de Tolbiac, F-75013 Paris, France

CR Subject Classification (1991): D.2, H.2

ISBN 3-540-56777-1 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-56777-1 Springer-Verlag New York Berlin Heidelberg

Printed in Germany

Typesetting: Camera ready by author
45/3140-543210 - Printed on acid-free paper

Preface

The University of Paris-Sorbonne is very proud to host this year the CAiSE'93 Conference on Advanced Information Systems Engineering.

This event is the fifth of a series of conferences initiated by Professor Janis Bubenko from the Swedish Institute for Systems Development in Stockholm, Sweden and Arne Sølvberg from the Norwegian Institute of Technology in Trondheim, Norway. Starting in 1989 in Stockholm, this series evolves from a Nordic audience to a truly European one. All these conferences have attracted international papers of high quality and indicated the need for an international conference on advanced information systems engineering topics.

The call for papers for CAiSE'93 was given international audience. The programme committee was chosen from very well reputed researchers in the international information systems engineering community, as well as key professionals in European industrial and consultant companies. One hundred and twenty papers have been submitted, representing authors from twenty-five different countries. Thirty-three papers of high technical quality have been accepted for presentation and discussion during the conference.

The spectrum of the contributions contained in the present proceedings extends from inevitable and still controversial issues regarding modeling of information systems, via development environments and experiences, to various novel views for some specific aspects of information systems development such as reuse, schema integration, and evolution.

The editors express the hope that these papers will contribute to improvements in both the understanding and practice of information system engineering.

A major challenge for any research community is to provide a forum for presentation and exchange of research results and practical experiences within the field. It is our hope that this year's conference may serve as a contribution to this end and will enhance communication between the information systems research community and the information systems professionals.

The CAiSE'93 conference would not have been possible without the efforts of the authors, the program committee members and the additional referees. They deserve our deepest thanks.

March 1993

Colette Rolland
François Bodart
Corine Cauvet

General Conference Chair

François Bodart, Faculté universitaire Notre Dame de la Paix, Belgium

Programme Committee Chair

Colette Rolland, Sorbonne University, France

Organising Chair

Corine Cauvet, INFORSID, France

Programme Committee

Andersen, Rudolf
Bodin, René Abel
Bouzeghoub, Mokrane
Brinkkemper, Sjaak
Bubenko, Janis
Cauvet, Corine
Chrisment, Claude
De Antonellis, Valeria
Falkenberg, Eckhard
Finkelstein, Anthony
Flory, André
Foucaut, Odile
Franckson, Marcel
Gardarin, Georges
Habrias, Henri
Hagelstein, Jacques
Hanani, Michael
Hori, Koichi
Jarke, Matthias
Jeffery, Keith
Kangassalo, Hannu
Kouloumdjian, Jacques
Layzell, Paul
Le, Frederic
Leonard, Michel
Lochovsky, Fred
Lockeman, Peter
Loucopoulos, Pericles
Luguet, Jacques
Mac Donald, Yan
Mannino, Michael V
Mylopoulos, John
Nilsson, Björn
Olive, Antoni
Oziard, Philippe
Pernici, Barbara
Prakash, Naveen
Proix, Christophe
Rault, Jean Claude
Rochfeld, Arnold
Sauge, Pierre
Sernadas, Amilcar
Shimojo, Shinji
Sølvberg, Arne
Spaccapietra, Stefano
Sutcliffe, Alistair
Tabourier, Yves
Tardieu, Hubert
Thanos, Costantino
Theodoulidis, Babis
van Assche, Frans
van de Riet, Reind P.
Vassiliou, Yannis
Wand, Yair
Wangler, Benkt
Wasserman, Anthony
Wieringa, R.J.
Wijers, Gerard M.
Wuwongse, Vilas
Zicari, Roberto

Additional Referees

Ambrosio, Ana Paula
Amghar Youssef
Auddino, Annamaria
Balsters, Herman
Brunet, Joël
Dupont, Yann
Engmann, Ralf
Finance, Béatrice
Grefen, Paul
Grosz, Georges
Gulla, Jon Atle
Harmsen, Frank
Joosten, Stef
Kraïem, Naoufel
Krogstie, John
Levreau, Gilles
Metais, Elisabeth
Parent, Christine
Plihon, Véronique
Raïs, Naïma
Ruggia, Raul
Schmitt, Jean-Roch
Sedes, Florence
Seltveit, Anne Helga
Sindre, Guttorm
Tari, Zahir
Teniente, Ernest
van de Weg, Rob
Wei Yang, Ming
Willumsen, Geir
Zurfluh, Gilles

Contents

Requirements Engineering

Conceptual Modeling I

Conceptual Modeling II

Schema Transformation

Reuse and Reliability

Animation and Verification

Development Environments

Panel Statement

Consistent Structural Updates for Object Database Design

P. Poncelet
Université de Nice - Sophia Antipolis

L. Lakhal
Université de Nancy I- ESSTIN

I3S - CNRS - URA 1376. Bât. 4 - 250 avenue A. Einstein
Sophia Antipolis - 06560 Valbonne - FRANCE
Tel: (33) 92 94 26 22 - Fax: (33) 92 94 28 98 - E-mail: poncelet@opaline.unice.fr

Abstract. This paper focuses on consistent structural updates for object database design and is included in a formal approach* for advanced database modeling and design. This approach is based on the IFO_2 model, an extension of the semantic model IFO defined by Abiteboul and Hull. It preserves the acquired strengths of the semantic approaches, whilst integrating concepts of the object paradigm. Structural part of the model including concepts such as alternative, composition, grouping for building complex objects and semantics constraints are defined. Furthermore, the definitions of consistent updates necessary to modify and perfect IFO_2 schemas are formally specified through change functions. The result is a new coherent and formal approach which is useful in overcoming some of the difficulties in the specification and design of advanced applications.

1 Introduction

Given the evolving complexity and size of applications, traditional models are proving to be restrictive. Indeed, the building of new applications requires more powerful constructors and a greater degree of modeling flexibility. Furthermore, the philosophy behind classical models does not fit in with new development tools - mainly object-oriented [5, 6] and extended relational database management systems [1, 24]. So, 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, 8, 11, 13, 20].

The presented research work fits into this context. We propose a new approach for which the three main aspects are the following:

(1) a formal object model IFO_2 [21, 22] is defined for advanced database modeling. It is an extension of the semantic model IFO proposed by Abiteboul and Hull [2]. Its objective is actually to reconcile apparently opposed ideas, firstly an optimal data representation and secondly a complete real world modeling. IFO_2 attempts to preserve the acquired strengths of semantic approaches, whilst integrating concepts of the object paradigm [4];

(2) structural update primitives are formally proposed through change functions to offer an incremental specification of IFO_2 schemas;

(3) finally, in order to design object database schema, a set of transformation rules translates an IFO_2 schema into an implementable one. To illustrate this conversion, the chosen target model is the established O_2 one [15].

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

Our approach offers the user concepts powerful enough to achieve from the real world the most complete specification possible. Its modeling capabilities may be compared with those of modeling semantics currents [12] rather than object-oriented models which are not expressive enough. Indeed, IFO_2 proposes the concepts of alternative, composition and grouping and explicitly expresses structural semantics constraints (connectivity and existency). However, the semantics currents suffer from the lack of concepts (such as object identity, reusability...) which are efficient for advanced application modeling. Furthermore, update facilities are not always proposed, and when they exist, they are described in an intuitive way.
In contrast with these approaches, we integrate, in IFO_2, the strength of the object models whilst boosting their modeling abilities and respecting independance between the specification and target models. Another advantage of our approach is to reduce the dataloss due to ambiguous vocabulary, particularly during the transfer between the conceptual and logical levels. Moreover, this brings about an optimization of the translation rules of an IFO_2 schema toward target models.
We maintain that it is essential to have a really rigorous model such as IFO_2. The object paradigm allows and encourages a modular modeling of the real world. Thus, object modeling can sometimes look "anarchistic" [27] and therefore difficult to handle. In order to avoid such problems, a formal approach leads to a schema which is non-ambiguous, without omissions, modifiable and easily reusable.

This paper particularly focuses on the structural update facilities provided in our approach. They are crucial for they assist the designer in taking real world evolutions into account or in rectifying 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. Nevertheless, as mentioned earlier, structural updates have not been examined with all the required attention, in semantics modeling approaches. If the object models provide database evolution mechanisms (three trends have been defined in [3]), they do not deal with conceptual schemas, and their objectives differ from ours. However, they are interesting for they pinpoint two levels which should be taken into consideration: the IS_A hierarchy and the composition hierarchy. For instance, we may quote:
(1) The Mosaico system where algorithms are defined for type insertions in a lattice [17];
(2) The Esse project where algorithms ensure consistent updates of an O_2 database schema [9, 28];
(3) The Gemstone [19] and Orion [14] systems, the Sherpa [18], Farandole2 [3] and Cocoon [26] projects where rules for the evolution schema are stated.

In the following sections, we first describe the structural part of the IFO_2 model and then we examine its structural updates. These two parts are presented both informally and formally.

2 The IFO_2 Model

To present the IFO_2 model, we first describe the object and type concepts as well as the different constructors. We then explain the fragment notion and IFO_2 schema.

2.1 Informal Presentation

IFO_2 is a formal object model which is both type and attribute oriented [13]. It 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 in consideration of the object paradigm. Secondly, to fully meet the objectives, the modeling power of IFO must be enhanced. Then, the concepts of alternative, composition and grouping for building complex objects have been integrated. Connectivity and existency constraints are explicitly specified.

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

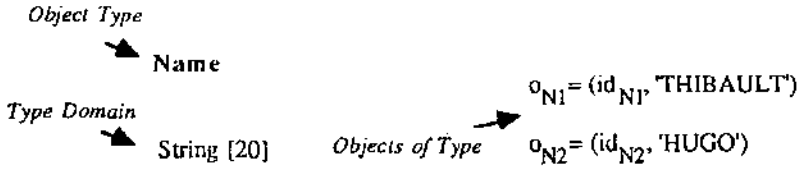

Fig. 1. Object Type Example

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 in the Entity/Relationship model [25];

(2) an **abstract** type (TOA) which would be perceived as entity in the Entity/Relationship model;

(3) a **represented** type (TOR) which allows the handling of 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.

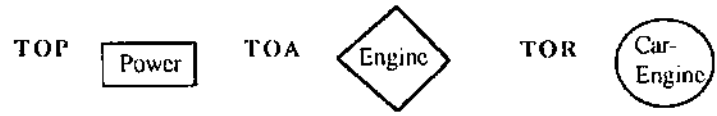

Fig. 2. Basic Type Examples

IFO_2 takes into account five constructors:

(1) **aggregation** and **composition**: they represent the *tuple* constructor of object models with an exclusive constraint for the composition (an object can take part in a unique construction);

(2) **collection** and **grouping**: they represent the *set-of* constructor of object models with an exclusive constraint for the grouping;

(3) **union type** (**alternative**): it allows the handling in the same way of structurally different types. This constructor represents the IS_A generalization link enhanced with a disjunction constraint between the generalized types.

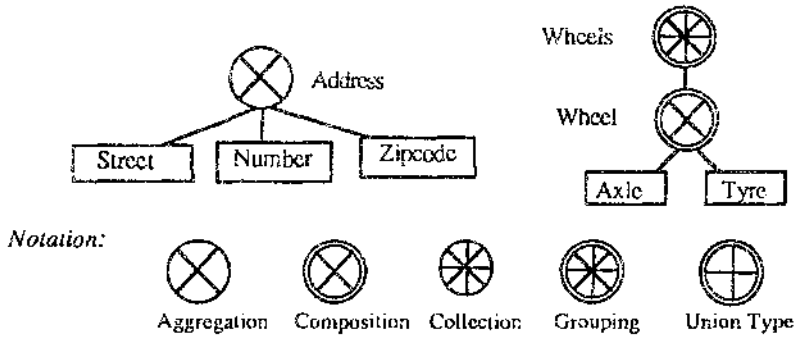

Fig. 3. Type Constructors

These constructors can be recursively applied according to specified rules for building more complex types. For example (see the figure 3), 'Address' is built from 'Street', 'Number' and 'Zipcode' types and 'Wheels' is composed with the 'Wheel' type obtained from 'Axle' and 'Tyre' types.

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. Firstnames are not always known for a person.

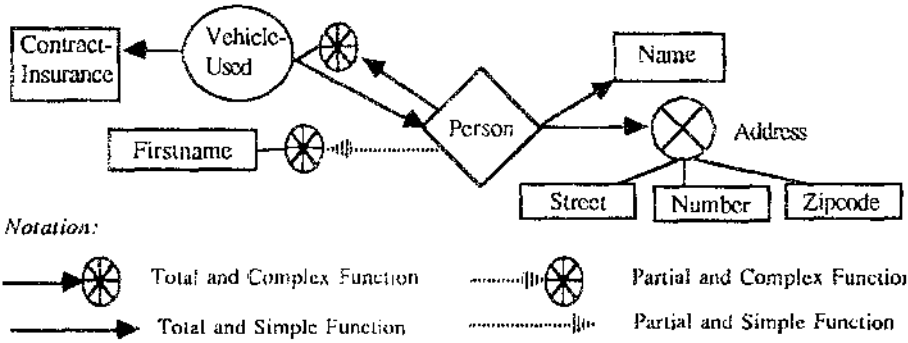

Fig. 4. The Fragment 'Person'

The objects involved in the description of heart objects are described in the fragment instance. The latter is achieved as follows: a function maps each heart object into property values (objects of other fragment types). From the fragment instance, we define for each fragment type its attached objects. Inheritance is then defined in a formal and generic way.

Finally, an IFO_2 schema is a set of fragments related by IS_A links according to two rules. The IS_A link in the IFO_2 model is the specialization link of the semantics models [12]. It represents either the subtyping (inheritance) if the target is a fragment heart or the client/supplier concept [16].

A schema instance is obtained by the union of associated fragment instances. These instances follow property propagation (through the attached heart objects) via the IS_A link (the represented type inherits the heart description). The figure 5 illustrates one IFO_2 schema made up of four fragments 'Person', 'Vehicle', 'Car' and 'Engine'. They are linked with IS_A links through the represented types ('Vehicle_Used', 'V_Car', 'Truck_Engine' and 'Car_Engine').

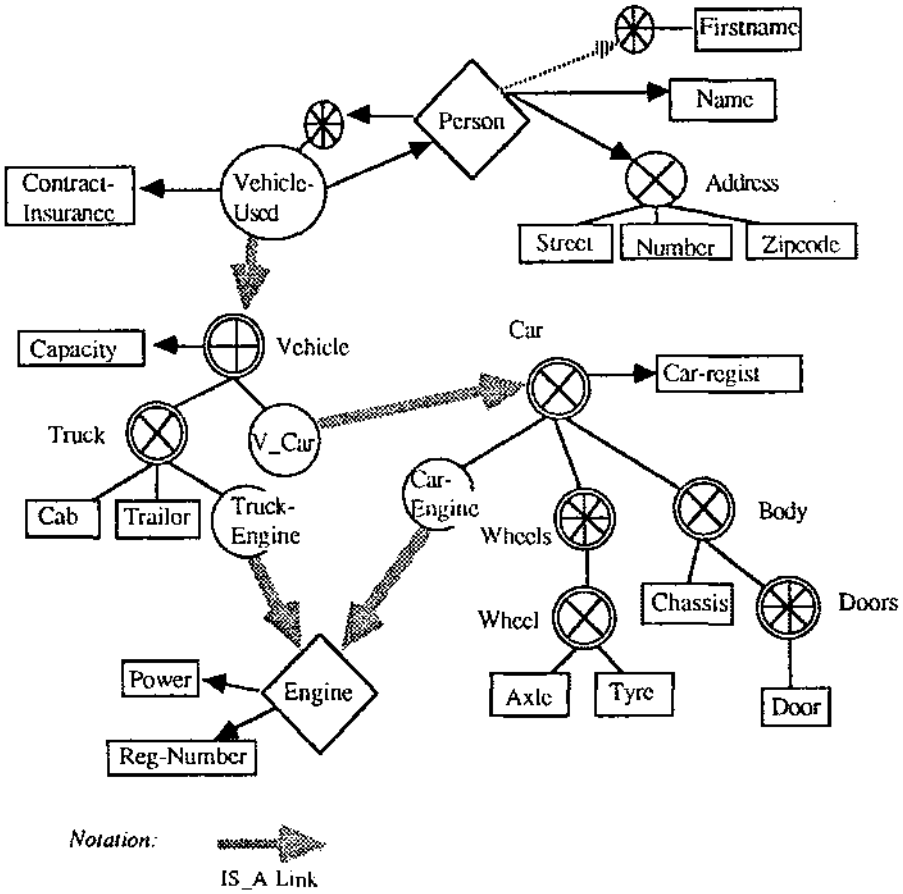

Fig. 5. An IFO_2 Schema

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 developed 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 toward different models due to its genericity. The translation of an IFO_2 schema into an O_2 schema is a prime example of this. The formal rule definitions reduce data-loss and misinterpretation.

In the next section, we propose part of the formal definitions which describe the introduced concepts. Instance and attached object concepts are not presented in this paper, the interested reader can refer to [21, 22].

2.2 Object and Type

Each object has an identifier independent of its value. We need then to define the concepts of value and identifier domains.

> $\mathcal{TO}$ is an infinite set of object types such that: $\forall \tau \in \mathcal{TO}$, **Dom** ($\tau$) 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 = (id, value) and o' = (id', value') of type τ, (id, id') $\in$ Did(τ)^2 with id $\neq$ id' and (value, value') $\in$ Dom (τ)^2. The infinite set of objects of type τ is called **Obj** (τ).

2.2.1 Printable and Abstract Types

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

> Let $\mathcal{TOP}$ be an infinite set of printable types, let $\mathcal{TOA}$ be an infinite set of abstract types, two disjoint subsets of $\mathcal{TO}$, such that:
> (1) $\forall \tau \in \mathcal{TOP}$, dom ($\tau$) is an infinite set of symbols;
> (2) $\forall \tau \in \mathcal{TOA}$, dom ($\tau$) = {$\emptyset$}.

2.2.2 Complex Types

The IFO_2 model takes into account five type constructors and makes a distinction between an exclusive and a non-exclusive building. Examples with different kinds of complex types can be found in [22].

2.2.2.1 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 aggregate type.

Let $\mathcal{TOTA}$ be an infinite set of aggregation types, let $\mathcal{TOTC}$ be an infinite set of composition types, two disjoint subsets of $\mathcal{TO}$, such that:

$\forall \tau \in \mathcal{TOTA} \cup \mathcal{TOTC}$, $\exists\ \tau_1, \tau_2, ..., \tau_n \in \mathcal{TO}$, $n > 1$, such that:
$Dom(\tau) \subseteq Obj(\tau_1) \times Obj(\tau_2) \times ... \times Obj(\tau_n)$,
τ is structurally defined as:
$\forall o \in Obj(\tau)$, $\exists\ o_1 \in Obj(\tau_1), o_2 \in Obj(\tau_2), ..., o_n \in Obj(\tau_n)$ such that:
$o = (id, [o_1, o_2, ..., o_n])$;
if $\tau \in \mathcal{TOTC}$ then $\forall o' \in Obj(\tau)$ with $o \neq o'$,
$\exists\ o'_1 \in Obj(\tau_1), o'_2 \in Obj(\tau_2), ..., o'_n \in Obj(\tau_n)$ such that:
$o' = (id', [o'_1, o'_2, ..., o'_n])$ with $\forall i \in [1..n], o_i \notin [o'_1, o'_2, ..., o'_n]$.

2.2.2.2 Collection and Grouping Types
As we have seen in section 2.1, a grouping is a collection with an exclusivity constraint.

Let $\mathcal{TOSC}$ be an infinite set of collection types, let $\mathcal{TOSG}$ be an infinite set of grouping types, two disjoint subsets of $\mathcal{TO}$, such that:

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

2.2.2.3 Alternative Types
Stucturally different types can be handled in a uniform way through the alternative type (type union) concept.

Let $\mathcal{TOUT}$ be an infinite set of type union types, a subset of $\mathcal{TO}$, such that:

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

2.2.3 Represented Types

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

Let $\mathcal{TOR}$ be an infinite set of represented types, a subset of $\mathcal{TO}$, such that:
$\forall \tau \in \mathcal{TOR}, \exists \tau' \in \mathcal{TO}$ called source of τ such that Obj (τ) = Obj (τ'),
with $\forall o \in$ Obj (τ), $\exists o' \in$ Obj (τ') such that $o = o'$.

2.2.4 Types

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

A type $T \in \mathcal{TO}$ 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 $\mathcal{TOP}$, $\mathcal{TO\mathcal{A}}$, $\mathcal{TOR}$, $\mathcal{TOT\mathcal{A}}$, $\mathcal{TOTC}$, $\mathcal{TOSC}$, $\mathcal{TOSG}$, $\mathcal{TOUT}$;
(2) if $T \in \mathcal{TO\mathcal{A}}$ then T is root of type;
(3) printable and represented types are leaves of the tree.
A type edge between two vertices t' and t" is denoted $E_T(t' \rightarrow t")$.

An abstract type cannot be used in a type building since its role is to describe a real world entity which is not defined by its internal structure but by its fragment specified properties. The figure 6 shows a type whose root is 'Car'.

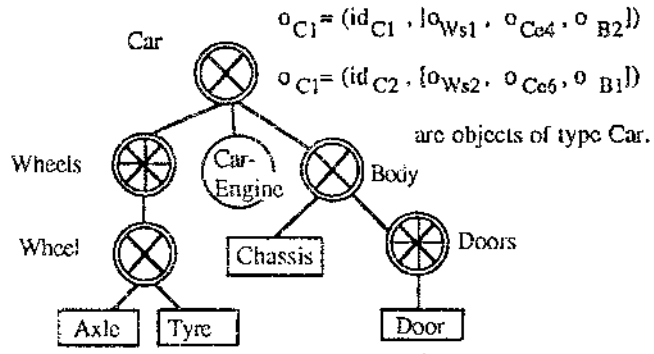

Fig. 6. The Type 'Car'

2.3 IFO_2 Fragment

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 $\mathcal{TO}$ and the set of edges U is composed with: simple edges (simple functions) and complex edges (functions applied on a $\mathcal{TOSC}$, 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.
The figure 5 shows four fragments: 'Person', 'Vehicle', 'Engine' and 'Car'.

An IFO_2 fragment is a graph $F = (V_F, L_F)$, with V_F the set of types T of $\mathcal{TO}$ 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.

A fragment link between two types T' and T" is denoted $L_{F(T' \rightarrow T")}$.

2.4 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 5 describes an IFO_2 schema.

2.4.1 Specialization Link

Let τ' be a type of $\mathcal{TOR}$ and let T be a type of $\mathcal{TO}$, such that it is the source of τ' and a heart of a fragment, the link of head T and queue τ' is called an IS_A link and denoted $L_{IS\_A\ (\tau' \rightarrow T)}$. T is called the source of the IS_A link and τ' the target.

The figure 7 illustrates the specialization link between 'Vehicle' and 'Vehicle_Used'.

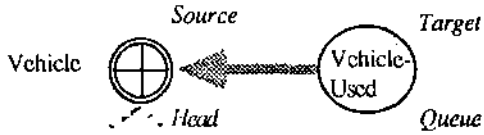

Fig. 7. Notation for Specialization Link

2.4.2 IFO_2 Schema

An IFO_2 schema is defined as a direct acyclic graph $G_S = (S_S, L_S)$ with S_S the set of type $T \in \mathcal{TO}$ 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 can be extended to a common vertex.

The structural part of IFO_2 model having been defined, we examine how it could be corrected or adapted by using update primitives.

3 Consistent Updates on IFO_2 Schema

3.1 Motivation

The problem with schema updates can be summarized as follows: how to modify a given schema whilst preserving a coherent representation? In other words, our aim is to ensure that updates retain the schema consistency. In object models, 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 [28]. In our context, we are only interested in the structural case.

An IFO_2 schema is a couple (S_S, L_S) where L_S is composed of both fragment and IS_A links but not every arbitrary (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*). Models such as Orion, O_2, Gemstone, Cocoon and Sherpa give a set of schema invariants which are conditions that have been satisfied by a valid schema.

Some schema changes are quite simple, whereas others need a complete reorganization of the database. The latters can often be expressed in terms of more elementary changes.
The following taxonomy, figure 8, presents the primitive schema updates in IFO_2. This set is minimal and complete in the sense that all possible schema transformations can be built up by a combination of these elementary operations (*completeness*). A similar 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 variables 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 types, functions and classes, the schema changes are respectively: updating types, updating functions and updating classes.

(1) Updating types	(2) Updating fragment links	(3) Updating IS_A links
(1.1) Add a new object type	(2.1) Add a new fragment link	(3.1) Add a new IS_A link
(1.2) Delete an object type	(2.2) Delete a fragment link	(3.2) Delete an IS_A link
(1.3) Change an object type (1.3.1) its name (1.3.2) its domain (for a printable type)	(2.3) Change the sort of a fragment link	
(1.4) Substitute a type		

Fig. 8. Taxonomy of Possible Updates in IFO_2

All schema structure changes such as fragment insertion into the direct acyclic graph, can 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 lattice)>.

All these updates are formally specified through the update functions (insertion and modification functions on schema, on fragment and on type). Intuitively, in IFO_2, 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 update 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. As we have seen, consistent sets have to be defined to perform only valid updates (for example, a type can be inserted as a sub-type if and only if the father has already more than one descendant, i.e. is an aggregation/composition/alternative type).

In the following sections, we present the insertion and modification of types both in an informal and formal way.

3.2 Informal Presentation

A schema being composed of IS_A and fragment links, the type insertion consists of relating a type to a schema. For example, consider the figure 9 which is a part of the figure 5.

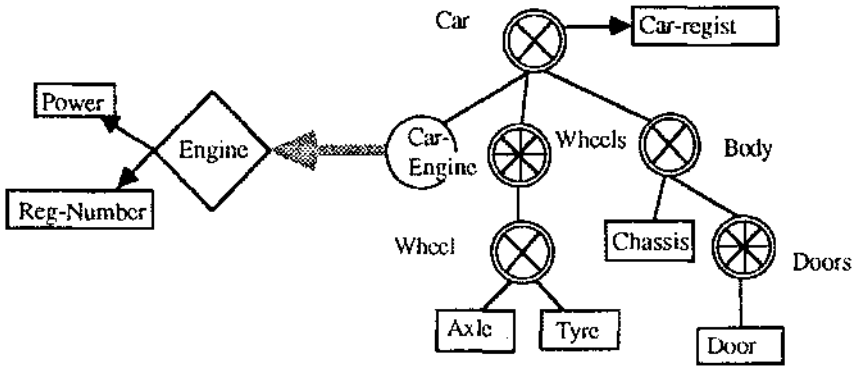

Fig. 9. An IFO_2 Fragment Creation Example

A type insertion is defined using a syntax which requires the added type and associated links relating it to the schema. The creation of the fragment 'Car' is done by insertion of the type 'Car' and is denoted by (Car, I). The link set 'I' describes both fragment and IS_A links which indicate how the type could be inserted into a schema. If the added type is a represented one, the reverse link between the represented type and the heart has to belong to the fragment link set. In our example, 'I' is composed of only one IS_A link (i.e. there is no fragment link) between the source 'Engine' and the target 'Car-Engine'. For the schema, the sets of vertices, fragment links and IS_A links are thus increased by the new components defined in the couple (Car, I). In our example, we obtain: S_S = {Engine, Power, Reg-Number) $\cup$ {Car, Car-Engine, Wheels, Wheel, Body} where S_S is the vertex set of the schema G_S; $L_{S\text{-}A}$ = {Engine $\rightarrow$ Power, Engine $\rightarrow$ Reg-Number}* where $L_{S\text{-}A}$ is the fragment link set of the schema G_S and $L_{S\text{-}IS\_A}$ = {Car-Engine $\rightarrow$ Engine}.

* For the purposes of simplification, a link (type edge, fragment or IS_A link) between two type T and T' is denoted by T $\rightarrow$ T'.

Let us now consider fragment modifications which are in fact type modifications. Intuitively, three cases have to be taken into consideration: a sub-type insertion, a type deletion and a type substitution. Such modifications are defined using the following syntax (p", p, p'). A partial insertion is thus denoted by (p", □, p') where p" is the vertex and p' is the inserted type; a deletion is denoted by (p", p, □) where p is the dropped type and a substitution is expressed by (p", p, p') where p is to be replaced by p'. A type being composed by edges and vertices, a type change modifies these components. We thus examine the necessary modification propagations for type deletion, insertion and substitution.

For example, if the type 'Wheels', in figure 9, is dropped by (Car, Wheels, □) then type edges and vertices are updated by deleting the 'Wheels' subtype and modifying recursively edges and vertices from the father of 'Wheels' to the root of type 'Car'. We thus obtain: V_{Car} = {Car, Car-Engine, Wheels, Wheel, Body} - {Wheels, Wheel} where V_{Car} is the 'Car' vertex set and E_{Car} = {Car → Car-Engine, Car → Wheels, Car → Body, Wheels → Wheel) - {Car → Wheels, Wheels → Wheel} where E_{Car} is the type edge set of the type 'Car'. The fragment is changed with an update function modifying vertices and fragment links. A type deletion has to satisfy the following property: all IS_A links whose target is in (or is) the deleted type have to be dropped.

The following sub-type insertion (Car, □, Roof-rack) adds the type 'Roof-rack' in the hierarchy. The sets of type vertices and type edges are thus increased respectively with 'Roof-rack' and the link from 'Car' to 'Roof-rack'. In our example, the sets are $V_{Car} = V_{Car} \cup$ {Roof-rack} and $E_{Car} = E_{Car} \cup$ { Car → Roof-rack}. In the same way, if the sub-type to be inserted is a represented type, then the set of schema IS_A links is increased with all the links whose represented type is the target. The fragment and schema are updated in the same way as in the deletion case.

Now consider the type substitution which replaces a type t by another type t'. For example, a consequence of (Car, Roof-rack, Wheels) is to replace, in the hierarchy, the type 'Roof-rack' by the type 'Wheels'. If the substitute is composed of represented type, the IS_A link set has to be updated. Finally, the vertex set, the fragment and IS_A links of the schema are updated.

These operations are carried out by using functions recursively applied **Modif_S**, **Modif_F** and **Modif_T** which update respectively schemas, fragments and types.

3.2 Formal Presentation

To facilitate the reading, the definitions are illustrated through simple examples. We may note that: an IFO_2 schema is **structurally consistent** if and only if it satisfies the two following properties:

(1) there is no IS_A cycle in the graph;

(2) two directed paths of IS_A links sharing the same origin can be extended to a common vertex.

3.2.1 Type Insertions

We define the insertion function as a type insertion.

An **insertion** is a pair Ins = (p, l) such that:
(1) $p \in \mathcal{TO}$;
(2) l is a set of fragments and IS_A links, denoted respectively by l_A and l_{IS_A}.

3.2.1.1 Consistent Type Insertions

An insertion of a type p into the schema has to be consistent. Informally, the insertion has to follow the model properties (each source of IS_A link is heart of fragment, there is a reverse link relating fragment heart to its represented property, ...) for preserving the schema in a valid state.

An added type in a schema is either an insertion of:
(1) a fragment heart, therefore the set of fragment link is empty or,
(2) a property p of a heart type h, then $L_{\perp}(h \rightarrow p)^*$ belongs to the fragment link set or,
(3) a property p of a nested fragment whose heart is T, therefore $L_{\perp}(T \rightarrow p)$ is an added fragment link.

Let $G_S = (S_S, L_S)$ be an IFO_2 schema, $Ins = \{(p_i, l_{A_i} \cup l_{IS\_A_i}), i \in [1..n]\}$ be a set of type insertions of G_S. Ins is said to be **consistent** if and only if it satisfies the following properties:

(1) $\forall (p, l) \in Ins$,
if $p \in \mathcal{TOR}$ then

$$l_{IS\_A} = \bigcup_{j=1}^{m} L_{\perp}\_IS\_A(p \rightarrow sj)^* \text{ where } sj \in S_S, \text{ fragment heart, is the source of p and m the number of sources of p;}$$

$$+ \bigcup_{k=0}^{n} L_{\perp}\_IS\_A(tk \rightarrow p) \text{ where } t_k \text{ is the target of p through an IS\_A link and n the number of p targets;}$$

$$\bigcup_{k=0}^{n} \left(\bigcup_{j=1}^{m} L_{\perp}\_IS\_A(tk \rightarrow sj) \right) \text{ to delete old IS\_A links of the hierarchy replaced by the previous adding;}$$

and
either $l_A = \{L_{\perp}(h \rightarrow p), L_{\perp}(p \rightarrow h)\}$ where $h \in S_S$ is heart of fragment;
or $l_A = L_{\perp}(T \rightarrow p)$ where $T \in S_S$ is a type such that $L_h(h \rightarrow T)$ is a complex fragment edge of L_S with h heart of fragment;
or $l_A = \emptyset$; /* *fragment creation* */

else

$$l_{IS\_A} = \bigcup_{k=1}^{m'} \left(\bigcup_{j=1}^{m} L_{\perp}\_IS\_A(tj \rightarrow skj) \right) \text{where m' is the number of represented types in p, m the number of sources of p and } s_{kj} \text{ an associated source, heart of fragment;}$$

and

* $\perp$ is used to indicate that these links are not still associated to a schema.

either $l_A = L_{\perp(T \rightarrow p)}$ where $T \in S_S$ is such that T is either heart of fragment or $L_{h(h \rightarrow T)}$ is acomplex edge in L_S;
or $l_A = \emptyset$. /* *fragment creation* */

(2) $(S_S' = S_S \cup \bigcup_{i=1}^{n} p_i, l_A \cup L_{IS\_A})$ follows the two schema rules.

Example: the fragment creation, through the type insertion of 'Car', is obtained with the 'l' link set composed by:

$l_A = \emptyset$. /* *There are no properties associated to the fragment heart* */
$l_{IS\_A} = L_{\perp\_IS\_A}$(Car-Engine→ Engine). /* *The inserted fragment is linked through the IS_A link between 'Car-Engine' (a represented type) and 'Engine' (heart of fragment)* */

3.2.1.2 Type Insertion Function

Let G_S be the set of IFO_2 schemas and let Ins be the set of consistent insertions into these schemas. The result of applying consistent insertions into an IFO_2 schema is a new structural consistent schema obtained as follows:

Let $G_S = (S_S, L_S)$ be an IFO_2 schema, let $Ins = \{(p_i, l_{A_i} \cup l_{IS\_A_i}), i \in [1..n]\}$ be a set of consistent insertions of G_S. The new structural consistent IFO_2 schema, noted $G_S' = (S_S', L_S')$, updated from G_S by Ins, is achieved by applying the insertion function on each element of Ins. Let Insert be the schema insertion function:

Insert:	$\mathfrak{G}_S$ X $\mathfrak{I}ns$	→	G_S
Insert	$(G_S = (S_S, L_S), Ins)$	=	G_S'

where $G_S' = (S_S', L_S'\_A \cup L_S'\_IS\_A)$ is such that:

$$S_S' = S_S \cup \bigcup_{i=1}^{n} p_i \ , \quad L_S'\_A = L_{S\_A} \cup \bigcup_{i=1}^{n} l_{A_i} \text{ and } \quad L_S'\_IS\_A = L_{S\_IS\_A} \cup \bigcup_{i=1}^{n} l_{IS\_A_i}.$$

Example: The type insertion of 'Car' updates the schema components as follows:
S_S' ={Engine, Power, Reg-Number} ∪ {Car, Car_Engine, Wheels, Wheel, Axle, Tyre, Body, Chassis, Doors, Door},
$L_S'\_A$ = {Engine → Power, Engine → Reg-Number},
$L_S'\_IS\_A$ = {Car_Engine → Engine}.

3.2.2 Schema Modifications

We define a modification which carries out a valid IFO_2 schema from an IFO_2 schema and a modification set. We differentiate several possible updates and present their result through a modification function.

Let **Father** be a function with domain and co-domain $\mathcal{TO}$ such that:
$\forall \tau \in \mathcal{TO}$, if τ is a root of type then **Father**(τ) = $\square$
else **Father**(τ) = τ' where τ' is the ascendant of τ.
Let **Outdegree** be a function with domain $\mathcal{TO}$ and co-domain $\mathbb{N}$ such that:
$\forall \tau \in \mathcal{TO}$, **Outdegree** ($\tau$) is the number of descendants of τ.

Let F be a fragment, let T be a type of F, a modification of F on T is a triple (p", p, p') such that:
if p = $\square$ then p" is a vertex of T and p' $\in \mathcal{TO}$
else /* *p is a vertex of T* */ p" = Father(p) and p' = $\square$ or p' $\in \mathcal{TO}$.

An update is a partial insertion of a type if p = $\square$, a deletion if p' = $\square$ and it is a modification otherwise (p' is a type). There is an update propagation only if p" is specified.

Example: The following triples provide modification examples:
(1) (Car, $\square$, Roof-rack) inserts the 'Roof-rack' type as subtype of the 'Car' type;
(2) (Car, Wheels, $\square$) drops the 'Wheels' sub-type from the 'Car' type;
(3) (Car, Wheels, Roof-rack) substitutes the 'Wheels' sub-type by 'Roof-rack' in the 'Car' type.

3.2.2.1 Consistent Modifications

Let $G_S = (S_S, L_S)$ be an IFO_2 schema, and M a set of modifications of G_S. M is said **consistent** if and only if it satisfies the following properties:

(1) $M = \bigcup_{i=1}^{n} M_i$

where n is the fragment number of G_S and M_i an update set of F_i such that:
$\forall T \in F_i$, $\forall$ (p", p, p') $\in M_i$ a modification of F_i on T,
if (q", q, q') $\in M_i$ is a modification of F_i on T then (p", p, p') = (q", q, q');
/* *There is a unique modification per type* */

(2) if ($\square$, p, p') is in M, a modification of F, whose heart is h, and p' is a represented type then $L_{(h \rightarrow p)} \in L_F$; /* *Case of reverse links for a represented type* */

(3) if (p", p, $\square$) is in M, a modification of F, whose heart is h,
and Outdegree(p") = 2 and Father(p") = $\square$ then if the other descendant of p" is a represented type then $L_{(h \rightarrow p")} \in L_F$;
/**A represented type is obtained by modification on other types* */

(4) the updated type must verify model definitions.

Example: The type insertion of a Wheel's "brother" is not possible because the constraint 4 is not satisfied (a grouping has only one "child").

3.2.2.2 Modification Functions

The result of applying consistent modifications on an IFO_2 schema is a new structural consistent schema obtained as follows:

Let $G_S = (S_S, L_S)$ be an IFO_2 schema and let M be a set of consistent modifications of G_S. The new structural consistent IFO_2 schema, noted $G_S' = (S_S', L_S')$, updated from G_S by M, is achieved by applying the update function on each element of M. Let G_S be the set of IFO_2 schemas, let M be the set of consistent modifications on these schemas.

Let **Modif** be the schema modification function:

$$\text{Modif: } G_S \times \mathcal{M} \rightarrow G_S$$
$$\text{Modif } (G_S = (S_S, L_S), M) = G_S'$$

where $G_S' = (S_S', L_S'\_A \cup L_S'\_IS\_A)$ is such that:

$$(S_S', L_S'\_A) = \bigcup_{i=1}^{n} \text{Modif\_F } (F_i, M_i)$$ where n is the fragment number of the schema;

and

$$L_S'\_IS\_A = L_S\_IS\_A - \bigcup_{j=1}^{m} L_S\_IS\_A\ (p \rightarrow Tj)\ /(p'', p, p') \in M$$ where p is a represented type;

/ deletion of links associated to deleted represented types */*

$$+ \bigcup_{k=0}^{n} \left(\bigcup_{j=1}^{m} L_S\_IS\_A\ (tk \rightarrow Tj) \right)$$ where k is the number of represented types whose source is p;

/ addition of links to the source of the deleted represented type */*

$$- \bigcup_{k=0}^{n} L_S\_IS\_A\ (tk \rightarrow p)$$ where k is the number of ex-targets of p;

/ Deletion of the links whose source is the deleted represented type, these types have been related by the previous set */*

$$- \bigcup_{j=1}^{m} L_S\_IS\_A\ (t \rightarrow Tj)\ /\ (p'', p, p') \in M$$ where t is a represented leaf of p;

/ deletion of links associated to represented types which belong to deleted types */*

$$+ \bigcup_{j=1}^{m} L_S\_IS\_A\ (p' \rightarrow Tj)\ /(p'', p, p') \in M$$ where p' is a represented type;

/ addition of links associated to added represented types */*

$$+ \bigcup_{j=1}^{m} L_S\_IS\_A\ (t' \rightarrow Tj)\ /(p'', p, p') \in M$$ where t' is a represented leaf of p';

/ addition of links associated to represented types which belong to added types */*

$$- \bigcup_{j=1}^{m} L\_IS\_A\ (T \rightarrow Tj)$$ with T a represented type of F_k ($k \in \{1..n\}$) such that: $\text{Modif\_F } (F_k, M_k) = \square$.

/ fragment deletion */*

Example: The IS_A link between 'Car_Engine' and 'Car' has to be deleted if the 'Car' type is dropped.

The fragments are updated by modifying their vertex and link sets through the following function. We may note that the heart dropping implies the associated fragment deletion.

Let F be the set of IFO_2 fragments.
Let **Modif_F** be the update function on fragments:

$$\begin{array}{lll} \textbf{Modif\_F}: \mathcal{F} \times \mathcal{M} & \rightarrow & \mathcal{F} \\ \text{Modif\_F}\ (F = (V_F, L_F), M_F) & = & F' \end{array}$$

where $F' = (V_{F'}, L_{F'})$ is such that:
if $(\square, \text{heart\_of\_F}, \square) \in M_F$ then $F' = \square$

else $V_{F'} = \bigcup_{j=1}^{m} \textbf{Modif\_T}\ (T_j, M_j)$ where m is the type number of the fragment and M_j is the set of modifications of F on the type T_j;

and $L_{F'} = L_F - \{ L_F(T_k \rightarrow T), L_F(T' \rightarrow T_k)$ with $k \in [1..n]$ / **Modif_T** $(T_k, M_k) = \square \}$.

Example: The consequence of 'Car-regist' type dropping is the deletion of the fragment link between 'Car' and 'Car-Regist'.

The following function realizes the type updates. It may be applied recursively for aggregation/composition types with two components and collection/grouping types. The resulting type is an IFO_2 one.

Let **Modif_T** be the update function on types:

$$\begin{array}{lll} \textbf{Modif\_T}: \mathcal{TO} \times \mathcal{M} & \rightarrow & \mathcal{TO} \\ \text{Modif\_T}\ (T = (S_T, E_T), (p'', p, p')) & = & T' \end{array}$$

where $T' = (S_{T'}, E_{T'})$ is such that:
if p' = □ then /* *Deletion case* */
  if p" = □ then T' = □ /* *Type root deletion* */
  else
    if Outdegree(p") = 2
    then T' = Modif_T (T, (Father(p"), p", s)) where s is the other descendant of p"
    else
      if Outdegree(p") > 2
      then $E_{T'} = E_T - E_T(p'' \rightarrow p) - E_p$ and $S_{T'} = S_T - S_p$*
      else
        if Outdegree(p") = 1
        then T' = Modif_T (T, (Father(p"), p", □));

* (S_p, E_p) defines the subtype whose root is p.

else /* *Substitution or insertion case* */
if p = □ then $E_{T'} = E_T + E_{T(p''\rightarrow p')} + E_{p'}$ and $S_{T'} = S_T + S_{p'}$
else $E_{T'} = E_T - E_{T(p''\rightarrow p)} + E_{T(p''\rightarrow p')} - E_p + E_{p'}$ and $S_{T'} = S_T + S_{p'} - S_p$.

Example: The deletion of the type 'Tyre' is done with (Wheel, Tyre, □). As 'Wheel' is a composition of two elements, the deletion of 'Wheel' would provide an inconsistent type (a composition of a unique element). The updates have thus to be sent back in the 'Wheel' father level: (Wheels, Wheel, Axle).
Then, we obtain the following sets: $E_{Car'} = E_{Car} - E_{Car(Wheels\rightarrow Wheel)} + E_{Car(Wheels\rightarrow Axle)}$ and $S_{Car'} = S_{Car} + S_{Axle} - S_{Wheel}$ where $E_{Car(Wheels\rightarrow Wheel)}$ is the type edge between 'Wheels' and 'Wheel' and $E_{Car(Wheels\rightarrow Axle)}$ is the new link between 'Wheels' and 'Axle'. The following figure shows the updated schema:

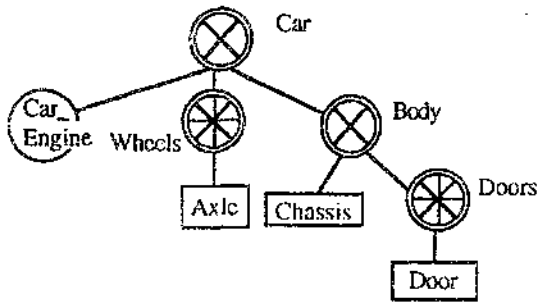

Fig. 10. The Resulting Type After Type Deletion

4 Conclusion

In this paper, we have formally defined an object model IFO_2 as well as the functions which update schemas whilst respecting their integrities. 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. IFO_2 integrates constructors, indispensable to the development of advanced applications, such as composition and grouping which enable the constituant sets to be "physically" taken into account. Its second strength is the update functions which are defined in the same way as the IFO_2 model concepts. 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 information is crucial.

A first version of the IFO_2 editor has been currently developed under Unix/XWindow (X11R5), with the help of the Aida/Masai (Version 1.5) programming environment, developed in object-oriented Le-Lisp (Version 15.24). This editor, as illustred in figure 11, is made up of three tools:

- a graphical view consisting of an editing panel, a tool panel and a workplace;
- a selection panel of object types and existing link types;

- an object editor enabling the textual representation of textual objects as well as information which does not appear in the schema.

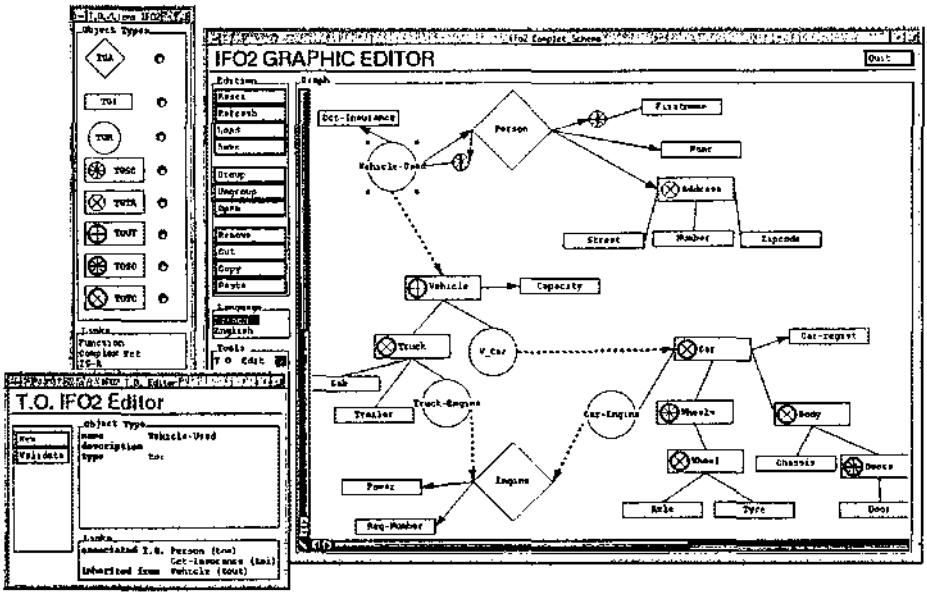

Fig. 11. The IFO_2 Editor

The type definition achieved with the editor are converted by a translator in O_2 descriptions (the algorithms can be found in [21]). The descriptions can thus be used in the target system. They have been implemented in the O_2 Database Management System (Version 3.3).

To carry out schemas without ambiguities and lacks, and which may be modified and easily reusable, it is essential to have a formal model. Furthermore, with the development of modeling and design tools, the users could work only with an intuitive perception of the formalized concepts. A formal framework thus provides the designers with real help without constraining them.

The prospects of the presented work here 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. The transfer from a conceptual schema to an optimized one is then easy. We believe that, dynamic and behavior will be integrated in the model using a formal approach based on the temporal logic [23, 10]. Such an approach automatically validates the specified constraints whilst being easily understood by the users.