Towards a Formal Approach for Object Database Design

P. Poncelet M. Teisseire R. Cicchetti L. Lakhal
University of Nice-Sophia Antipolis
Digital Equipment IUT Aix-en-Provence
Ferney Voltaire University of Aix-Marseille II
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 IFO2 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 IFO2 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 IFO2 schemas are specified through change rules. Finally, in order to design a database schema, an IFO2 schema is translated, in an automatical way, into an existing target (implementable) model. As an illustration, we present a translation from the IFO2 model into the O2 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.

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 IFO2 [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. IFO2 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 IFO2 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 IFO2 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 IFO2 model.
2. The associated structural update facilities through change rules.
3. The formalization of the translation rules from an IFO2 schema to an O2 one (according to the
established O2 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 modeled.

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 Mosaic0 system where algorithms are defined for type insertions into a lattice [19]; the Ese project where algorithms ensure consistent updates of an O2 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 IFO2 which is both type and attribute oriented. The IFO2 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 IFO2 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 IFO2. The object paradigm allows and encourages a modular modeling of the real world. So, object modeling cannot always 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 facilitating not only the comparison of different designs but also the verification of updates on specifications without further validations.

First of all, we present the IFO2 model. Update facilities are then explained and defined through change rules.

3 The IFO2 Model

IFO2 adopts the philosophy of the semantic model IFO. Two main extensions are realized. Firstly, an explicit 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 existence constraints are explicitly specified.

In the next sections, we propose a part of formal definitions of the IFO2 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 IFO2 schema are then detailed.

3.1 Object and Type

In the IFO2 model, an object has a unique identifier which is independent 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'.

279
Object Type  

Type Domain

Name  

String [20]

0

Object of Type

Figure 1 - Object Type Example

Definition 1: $T_0$ is an infinite set of object types such that:

$\forall \tau \in T_0, \text{Dom}(\tau)$ is an infinite set of symbols, including the empty set, called the value domain of $\tau$, $\text{Did}(\tau)$ is an infinite set of symbols called the identifier domain of $\tau$. Objects of type $\tau$ are defined by a pair (id, value) such that:

$\forall o, o' \in T_0, \exists (id, id') \in \text{Did}(\tau), \exists (value, value') \in \text{Dom}(\tau)$ such that: if $o=(id, value)$, $o'=(id', value')$ and $id \neq id'$ then $o \neq o'$.

The infinite set of objects of type $\tau$ is called Obj$(\tau)$.

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.

TOP  

TOA  

TOR  

Car Engine  

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 $T_0$, such that:

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

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 IFO2 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.

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 $T_{OA}$ be an infinite set of aggregation types, let $T_{TC}$ be an infinite set of composition types, two disjoint subsets of $T_0$, such that:

$\forall \tau \in T_{OA} \cup T_{TC}, \exists \tau_1, \tau_2, ..., \tau_n \in T_0, n > 1$, such that:

$\text{Dom}(\tau) \subseteq \text{Obj}(\tau_1) \times \text{Obj}(\tau_2) \times ... \times \text{Obj}(\tau_n)$.

$\tau$ is structurally defined as:

$\forall o \in \text{Obj}(\tau), \exists o_1 \in \text{Obj}(\tau_1), o_2 \in \text{Obj}(\tau_2), ..., o_n \in \text{Obj}(\tau_n)$ such that:

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

if $\tau \in T_{TC}$ then $\forall o' \in \text{Obj}(\tau)$ with $o \neq o'$

$\exists o'_1 \in \text{Obj}(\tau_1), o'_2 \in \text{Obj}(\tau_2), ..., o'_n \in \text{Obj}(\tau_n)$ such that $o' = (id', [o'_1, o'_2, ..., o'_n])$ with $\forall i \in [1..n], o'_i \neq o_i$.

Collection and Grouping Types

They represent the set-of constructor of object models with an exclusivity constraint for the grouping.
Definition 4: Let $T_{SC}$ be an infinite set of collection types, let $T_{GA}$ be an infinite set of grouping types, two disjoint subsets of $T_0$, such that:

$$\forall \tau \in T_{SC} \cup T_{GA}, \exists \not= \tau' \in T_0 \text{ such that:}$$
$$\text{Dom}(\tau) \subset P(\text{Obj}(\tau)) \text{ is the powerset of Obj(}\tau),$$

$\tau$ is structurally defined as:

$$\forall o \in \text{Obj}(\tau), \exists o_1, o_2, \ldots, o_n \in \text{Obj}(\tau') \text{ such that:}$$
$$o = (id, (o_1, o_2, \ldots, o_n))$$

if $\tau \in T_{GA}$ then $\forall o' \in \text{Obj}(\tau)$ with $o \neq o'$

$$\exists o'_1, o'_2, \ldots, o'_n \in \text{Obj}(\tau') \text{ such that:}$$
$$o' = (id', (o'_1, o'_2, \ldots, o'_n)) \text{ with } \forall i \in [1..n],$$
$$o_i \in \{o'_1, o'_2, \ldots, o'_n\}.$$  

Alternative Types (Union Types)

Structurally different types can be handled in a uniform 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 $T_{UT}$ be an infinite set of union type types, a subset of $T_0$, such that:

$$\forall \tau \in T_{UT}, \exists t_1, t_2, \ldots, t_n \in T_0, n > 0 \text{ such that:}$$

$$\text{Dom}(\tau) \subset \text{Dom}(t_1) \cup \text{Dom}(t_2) \cup \ldots \cup \text{Dom}(t_n),$$

$\tau$ is structurally defined as:

$$\forall i, j \in [1..n] \text{ if } i \neq j \text{ then}$$

$$\text{Obj}(t_i) \cap \text{Obj}(t_j) = \emptyset, \text{Obj}(\tau) = \text{Obj}(t_1) \cup \text{Obj}(t_2) \cup \ldots \cup \text{Obj}(t_n),$$

with $\forall o \in \text{Obj}(\tau), \exists ! k \in [1..n] \text{ such that:}$$
$$o = o_k, o_k \in \text{Obj}(t_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 $T_{R}$ be an infinite set of represented types, a subset of $T_0$, such that:

$$\forall \tau \in T_{R}, \exists t_1, t_2, \ldots, t_n \in T_0, n > 0 \text{ called}$$

source(s) of $\tau$ such that, $\text{Obj}(\tau) \subset \text{Obj}(t_1) \cup \text{Obj}(t_2) \cup \ldots \cup \text{Obj}(t_n)$ with $\forall o \in \text{Obj}(\tau), \exists o_i \in \text{Obj}(t_i)$ 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 $T \in T_0$ is a directed tree $T = (ST, ET)$, where $ET$ is a set of type edges. $T$ is such that:

1. The set of vertices $ST$ is the disjoint union of eight sets $T_{OP}, T_{OA}, T_{OR}, T_{OT}, T_{OC}, T_{OSC}, T_{OSG}, T_{OUT}.$
2. If $T \in T_{OA}$ 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 IF02 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.
Definition 8: An IFO2 fragment is a graph $F = (V_F, L_F)$, with $V_F$ the set of types $T \in \mathcal{T}_0$ 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 IFO2 fragment is called by its heart.

3.3 IFO2 Schema

An IFO2 schema is composed of $n$ IFO2 fragments: $F_1, F_2, ..., F_n$, $n > 0$, related by IS-A links according to two rules. The figure 11 illustrates one IFO2 schema made up with five fragments 'Person', 'Employee', 'Vehicle', 'Car' and 'Engine'. They are linked with IS_A links through the represented types ('Vehicle_Use', 'Employee', 'V_Car', 'Truck_Engine' and 'Car_Engine').

3.3.1 Specialization Link

The IS-A link in the IFO2 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 [IS].

Definition 9: Let $T'$ be a type of $\mathcal{T}_0$ and let $T$ be a type of $\mathcal{T}_0$, such that it is the (or one) source of $T'$ and a heart of a fragment, the link of head $T$ and queue $T'$ is called an IS_A link. $T$ is called the source of the IS_A link and $T'$ the target.

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

![Figure 5 - Notation for Specialization Link](image)

3.3.2 IFO2 Schema

Definition 10: An IFO2 schema is defined as a graph $G_S = (S_S, L_S)$ with $S_S$ the set of types $T \in \mathcal{T}_0$ 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 IFO2 fragments, called the IFO2 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 IFO2 model having now been formalized, we examine the supplied update facilities.

4 Updates on IFO2 Schema

Due to space limitation, we just present updates on IFO2 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 IFO2 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 IFO2 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 IFO2 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, OZ, Genesis, 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 IFO2, 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).

Figure 6 - Taxonomy of possible updates in IFO2

<table>
<thead>
<tr>
<th>Types updates</th>
<th>Fragment link updates</th>
<th>IS-A link updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1.1) Add a new type</td>
<td>(2.1) Add a new fragment link</td>
<td>(3.1) Add a new IS-A link</td>
</tr>
<tr>
<td>(1.2) Delete a type</td>
<td>(2.2) Delete a fragment link</td>
<td>(3.2) Delete an IS-A link</td>
</tr>
<tr>
<td>(1.3) Change a type</td>
<td>(2.3) Change the sort of fragment link</td>
<td></td>
</tr>
<tr>
<td>(1.3.1) its name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1.3.2) its domain for printable type only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1.4) Substitute a type</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7 - An IFO2 fragment creation example

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.

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 R1.1 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 1.5 is controlled.

Rule 3: IS_A link deletion condition
R3.1: an IS_A link can be deleted if and only if 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 and 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 1.1). 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](image)

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 (rule 3).
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.

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 \( T \) be the transformation function from the IFO2 model to the O2 one:

\[
T: IFO2 \rightarrow O2
\]

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

where \( m \) is the fragment number of the IFO2 schema.
This transformation uses a $F$ 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 O2 model. We adopt the following notation: for each $t$ type of $T_0$, $C_t$ is a class name and $t$ is an attribute name in the O2 model.

**Definition 12:** Let $n$ be the edge number whose source is the fragment heart, $T$ is defined as:

$$T\text{ (Fragment) } =$$

1. if heart $\in T_0 \cup T_R$:
   
   class $C_{\text{heart}}$
   
   public type tuple ( heart: $F$(heart, $\Box$, $\Box$) )

2. if heart $\in T_0$:
   
   class $C_{\text{heart}}$ public type tuple ( )

3. if heart $\in T_R$:
   
   $^*$ $t$ is the number of IS-A links
   
   class $C_{\text{heart}}$ inherit $C_{t_1}, ..., C_{t_k}$
   
   public type tuple ( )

   $^*$ $t$ has m sources:

   class $C_{\text{heart}}$ 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

   $^*$ $t$ has only one source:

   $F$(i, $\Box$, $\Box$) = $C_{t'}$  /* $t'$ is the source of the IS-A link */

   $^*$ $t$ has m sources:

   $F$(i, $\Box$, $\Box$) = $C_{t_1}, ..., C_{t_m}$
   
   $^*$ there are m sources of IS-A links

   $^*$ $n$ is the number of $T_0$ types

   $F$(i, $\Box$, $\Box$) = tuple ( $t_i$: $F$(ti, $\Box$, $\Box$) )

   $^*$ $k$ is the number of related types

   $^*$ $k$ is the number of aggregated types and $n$ the number of $T_0$ types

   $^*$ $t'$ is the collected type

   $F$(i, $\Box$, $\Box$) = set ( $F$(ti, $\Box$, $\Box$) )

   $^*$ $t'$ is the collected type

   $F$(i, $\Box$, $\Box$) = set ( )

   $^*$ $t'$ is the collected type

5.2.2 Types and Edges Transformation Function

**Definition 13:** The $F$ function translates fragment types either in $T_0$ type or in $O_2$ class. It is defined as:

$^*$ For the general case, partial and total terms will be omitted.
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 \( t \in \text{TOUT} \) is:

\[
\text{method body Check\_Compo}\_t \text{ boolean in class C}_t \{
\text{/* \( k \) is the number of component types */}
\text{if (!}(\text{self} -> \text{Test\_compo}_{t}(\text{self} -> t\text{.compo}_{t})) \text{return false;})
\text{return true;}
\}
\]

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

\[
\text{method body Test\_compo}_{t} \text{ (object: C\_compo}_{t}) \text{ boolean in class C\_Cla}ss \{
\text{/* \( t_1 \) is a component of type \( t \) */}
\text{if (object -> C\_Cla}ss\_compo}_{t} \text{ boolean result=true;}
\text{for (obj in C\_Cla}ss\_s where obj -> C\_Cla}ss\_compo}_{t} \text{ = object) {result=false;} return result;}
\}
\]

In the same way, the following method verifies the grouping of a type \( t \) whose collected type is \( \tau \):

\[
\text{method body Check\_Set}\_t \text{ boolean in class C}_t \{
\text{/* \( \tau \) is a component of type \( t \) */}
\text{if (object -> C\_Cla}ss\_compo}_{t} \text{ boolean result=true;}
\text{for (obj in C\_Cla}ss\_s where obj -> C\_Cla}ss\_compo}_{t} \text{ = object) {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 IFO2 system. A first version of the IFO2 editor is currently developed under Unix/XWindow (X11R5), with the help of the Aida/Masai (Release 1.5) programming environment, developed in object-oriented Le-Lisp (Release 15.24).

This editor, as illustrated 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_Cla}ss, the C\_Cla}ss\_s value is the instances set of the C_Cla}ss class. For each new instance of C_Cla}ss, it is necessary to modify the set which is defined as: Name C_Cla}ss\_s = set(C_Cla}ss).
The type definitions achieved with the editor are converted by a translator into \( \text{O}_2 \) descriptions. The descriptions may thus be used in the \( \text{O}_2 \) system. For instance, the methods described in the previous section have been implemented in the \( \text{O}_2 \) Database Management System (Release 3.3). The following example shows some generated methods associated to the class \( \text{C-Truck} \):

```plaintext
method body Check_Compo_Truck: boolean in class \( \text{C-Truck} \)
    ( if (!(self->Test_Cab(self->Truck.Cab))) return false;
      if (!(self->Test_Trailer(self->Truck.Trailer))) return false;
      if (!(self->Test_Truck_Engine(self->Truck.Truck_Engine)))
          return false;
      return true; );

method body Test_Cab (object: \( \text{C-Cab} \)): boolean in class \( \text{C-Truck} \)
    ( \( \text{O}_2 \) \( \text{C-Truck} \) \( \text{obj} \);
      \( \text{o2 boolean result=true;} \);
      for (\( \text{obj} \) in \( \text{C-Truck's} \) where \( \text{obj->Truck.Cab} \) == \( \text{object} \))
          \( \text{result=false;} \); return \( \text{result} \); );

/* named values declaration */
name \( \text{C-Truck's} \): set (\( \text{C-Truck} \));
```

7 Conclusion

In this paper, we have formally defined an object model \( \text{IFO}_2 \) as well as its transformation into the \( \text{O}_2 \) 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 \( \text{IFO}_2 \) 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 constituent sets to be "physically" taken into account.

The most original aspect of \( \text{IFO}_2 \) 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 \( \text{IFO}_2 \) 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 \( \text{IFO}_2 \) 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 \( \text{IFO}_2 \); 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, \( \text{IFO}_2 \) 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 \( \text{IFO}_2 \) schema into an \( \text{O}_2 \) one is a prime example of this. The formal rule definitions reduce data-loss and misinterpretation.
The prospects of the presented work begin with the approach automatically validating the specified constraints to be determined. According to us, dynamic and behavior principally based on the object behavior. Moreover, integration of modeling abilities for the application based on the temporal logic [101 and [24]. Such an approach will be integrated in the model using a formal approach fragment, the most suitable optimized representation can be set through "process" specification associated with the model advocate an attribute-oriented modeling and are consistent handled informations is crucial. They ensure the integrity of the updated information. The ambiguities and contradictions are then detected and handled.

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.

8 References