Dynamic Modelling with Events

Maguelonne Teisseire[1], Pascal Poncelet[2] and Rosine Cicchetti[2]

[1] Digital Equipment
[2] IUT Aix-en-Provence
LIM - URA CNRS 1787 - Université d'Aix-Marseille II,
Faculté des Sciences de Luminy, Case 901
163 Avenue de Luminy, 13288 Marseille Cedex 9 FRANCE.
E-mail: teisseir@gia.univ-mrs.fr

Abstract. This paper focusses on the behavioural aspects of the IFO_2[3] model, an extension of the semantic model IFO defined by S. Abiteboul and R. Hull. Its originality is in the formalization of complex events and their specification, which adopts semantics and syntax indentical to those of the structural part. In addition, it offers concepts - particularly modularity and re-usability - that are unanimously recognized as useful for structural specification of applications.

1 Motivation

Recent development tools for advanced applications - mainly Extended Relational or Object-Oriented Database Management Systems [1] - introduce new concepts relevant for complex object management. In parallel, conceptual approaches [3, 5, 9, 10, 13, 14, 15, 19] strive to meet the needs of both traditional and advanced applications. Some of them give pride of place to behaviour representation and resort to OODB models for the structural modelling, while forgetting their typically implementable feature. Thus the models defined can be used for manipulation of complex objects, but they lose some of the benefits of semantic approaches [8]. Dependent upon target systems, they have shortcomings as regards the proposed constructors and only express semantic constraints (such as cardinalities) with the aid of methods (i.e. coding, which is paradoxical for conceptual approaches).
From a dynamic viewpoint, conceptual approaches are based on new concepts and mechanisms or make use of temporal logic [5, 6]. Since they aim to specify application behaviour, the problems which are studied are nearly similar to those of concurrent system design and software engineering.
In proposing the IFO_2 conceptual model [11, 12], we intend to integrate both structural and behavioural representation of applications in a consistent and uniform manner with respect to both the formalization introduced and the associated graphical representation. It is based on the IFO model of S. Abiteboul and R. Hull [2] and its aim, for the structural part, is to combine the advantages of both semantic and object-oriented approaches[4].

[3] This work, supported by an External European Research Project in collaboration with Digital Equipment, comes within the scope of a larger project whose aim is to provide an aided modelling and design system for advanced applications.

This paper is devoted to the dynamic aspect of the model. Its originality is in the formalization of complex events and their specification, which adopts semantics and syntax identical to those of the structural part. In addition, it offers concepts - particularly modularity and re-usability - that are unanimously recognized as useful for structural specification of applications. With IFO_2, it is possible to fully comprehend the overall behaviour of a system, by specifying it in a manner that is both "fragmented" and "optimized". These strong points are stressed in Section 2, which summarizes our contribution by drawing a parallel with the mentioned conceptual work. The various concepts introduced to represent the application behaviour are defined in Section 3. Then, to conclude, we have a brief look at the IFO_2 system that is currently being developed.

2 Related Work and Proposal

An event is the representation of a fact that participates in reactions of the modelized system. It occurs in a spontaneous random manner (in the case of external event) or is generated by the application. In both cases, it occurs instantaneously, i.e. it is of zero duration. Like in [7], we make the following assumption: no more than one event can occur at any given instant.
The structural part of IFO_2 is defined with respect to the "whole-object" philosophy. We extend its scope to the behavioural part and refer to a "**whole-event**" representation. In fact, event modelling in IFO_2 complies with a dual precept: typing and identification. As regards the latter point, we use the instant of occurrence of an event as its identifier.
The IFO_2 model proposes two basic types:

- the simple event type (TES) represents the events that trigger an operation (method) included in the IFO_2 structural description;
- the abstract event type (TEA) is used to specify external or temporal events or events that generate other events.

To modelize a system behaviour, it is necessary to express different variants of event conjunction and disjunction. To answer this need, we have chosen to represent complex events by using **constructors**: composition, sequence, grouping and union. With this approach, we provide not only the required expressive power but also the uniformity with respect to the IFO_2 structural modelling. When representing both static and dynamic parts of an application, the designer handles concepts having the same philosophy.
The types of events are interconnected by functions through the event fragment concept. Its role is to describe a subset of the modelized behaviour that can then be used as a whole by means of the represented type concept. Consequently, it is possible to manipulate another type - without knowing its description - via an IS_A event link. These concepts offer a real modularity and re-usability of specifications:

[4] Here we do not aim to present the structural part of IFO_2 (the interested reader may refer to [11, 12]).

the designer may defer a type description or entrust it to somebody else, while using a represented type which symbolizes it.
The fragment functions express various constraints on the event chain (obligation of occurrence, multiplicity and any possible wait). In addition, we make a distinction between triggering functions and precedence functions which loosely express the fact that an event triggers the occurrence of other ones or that it is preceded by the origination of other ones. In order to underline this, let us consider an external or temporal event. By its very nature, it cannot be triggered by another modelized event, therefore it is sometimes necessary to express that its occurrence is mandatorily preceded by other events. This makes it possible, by adopting a specific observation point (called the fragment heart) to have an overview of the behaviour in question, i.e. including not only generating events (preceding events) but also generated events (triggered events). The fragment can thus be considered as a unit of description of the system behaviour.
In order to modelize the general behaviour of the application, the partial views provided by the fragments are combined (via IS_A links) within an event schema.

With the proposed approach, an application is described by a structural specification and a behavioural specification, which are closely related but clearly distinct. Our philosophy is therefore different to that of models such as in [10, 13, 14] that choose to combine these two aspects in a single schema. Therefore the behaviour is described for each class of objects and an additional mechanism must be used to specify the interactions between classes.
Our philosophy has two advantages in relation to these approaches: a single uniform description of events, including events shared by objects of different types, but, above all, an overview of the system dynamics, which we believe to be essential for a conceptual model. However, it should be noted that, in common with the approach proposed in [10], we are mindful of the modularity of specifications: the represented type concept in IFO_2 has the same purpose as that of a role.
An overall dynamic vision is also proposed in [6, 17], which describes the behaviour of an object database by using temporal logic. In this model, the constraints applied to the occurrence of events are expressed with the aid of the trace concept [16] (event history) and operators applied to the traces. We make use of such a concept for specification of precedence and triggering functions and manipulate traces to translate particular conditions of the event chain.

3 IFO_2 Behavioural Model: Formal Presentation

3.1 Time and Event Type

The behaviour of any real system is within the "time" dimension, therefore it is firstly necessary to specify this concept. We do so, in a similar manner to [4], by defining time as a set of equidistant instants with an origin and where each point in time can be represented by a non-negative integer. The spacing of intervals corresponds to the system granularity (i.e. the smallest representable unit of time). In our approach, the events that take part in the system dynamics occur in an ordered manner in this temporal dimension.

Definition 1 *Time* is an infinite set of symbols and $<_{Time}$ the total order relation on this set.

Apart from the identifier and event domain concepts (determining the events and operations that take part in instances), the definition of an event type entails structural elements through the concept of parameter domain.

Definition 2 $\mathcal{TE}$ is an infinite set of event types such that: $\forall te \in \mathcal{TE}$, $Did(te)$ is an infinite set of symbols called the **identifier domain** of te with $Did(te) \subset time$, $Dom(te)$ is an infinite set of symbols, including the empty set, called the **event domain** of te and $Dpara(te)$, the **parameter domain** of te, is included in $\mathcal{P}(S_S)$ (where $\mathcal{P}(S_S)^5$ is the powerset of the object types of the structural schema).

Definition 3 An **event** of type Te is a triplet $(id, occ, para)$ such that: $\forall e, e'$ of type Te, $\exists (id, id') \in Did(Te)^2$, $\exists (occ, occ') \in Dom(Te)^2$, $\exists (para, para') \in Dpara(Te)^2$ such that: if $e = (id, occ, para)$, $e' = (id', occ', para')$ and $id = id'$ then $e = e'$.
The infinite event set of type Te is called $Evt(Te)$.

To illustrate this paper, the modelized system is a lift. An event type involved in the description of the lift behaviour is "Up", which describes the ascending motion of the lift cage. Let us consider an event, e_{Up_1}, of this type. It could be specified as follows: e_{Up_1}=(id_{Up_1}, *up*, $@_1\_cage$). This means that the event e_{Up_1} occurred at the instant id_{Up_1}, with the $@_1\_cage$ as parameter. The *up* component maps with an operation of the structural schema.

For each event type te of $\mathcal{TE}$, there are two functions: a bijective function Id with domain $Evt(te)$ and codomain $Did(te)$ which associates with each event of type te its identifier and an injective function $Para$ with domain $Evt(te)$ and codomain $Dpara(te)$ which associates with each event of type te its parameters.

3.2 Basic Event Types

A simple event type, TES, describes the triggering of an operation which is specified in the structural schema. Other atomic types are modelized through the abstract event type concept, TEA. It describes events which are external or temporal events or generators of other events. Figure 1 shows the graphical formalism for basic event types.

Definition 4 Let $\mathcal{TES}$ be an infinite set of **simple** event types and let $\mathcal{TEA}$ be an infinite set of **abstract** event types, two disjoint subsets of $\mathcal{TE}$, such that:

1. $\forall te \in \mathcal{TES}$:
   (a) $\exists op \in OP(F_{Struct}) \mid Dom(te) = op$ where $OP(F_{Struct})$ is the operation set of the structural fragment $F_{Struct} \in G_s$;
   (b) $Dpara(te) \subseteq \mathcal{P}(V_S)$ where $\mathcal{P}(V_S)$ is the powerset of V_S and V_S is the object type set of the fragment F_{Struct}.
2. $\forall te \in \mathcal{TEA}$, $Dom(te) = \emptyset$.

Fig. 1. Example of Basic Event Types

Figure 1 presents the TES "Up", evoked in the previous example, and two TEAs: "Satis-Request" and "Floor-Request". The former describes events generated when the lift reaches the required floor. The latter represents external events occurring when users request a floor.

3.3 Complex Event Types

To describe the system behaviour, complex event combinations have to be expressed. They provide constraints on event occurrences. The four constructors (shown in Figure 2) proposed in the IFO_2 model specify the logical conditions on events: conjunction with different constraints and disjunction.
Each event may only take part in a single construction since it occurs only once. Accordingly, in the constructor definitions, this is formally expressed through an exclusivity constraint. Furthermore, a composite event stems from the occurrence of its components.

Composition and Sequence Event Types: the event composition and sequence constructors reflect the conjunction of events belonging to different types. The sequence includes a chronological order on the occurrences of the component events.

Definition 5 Let $\mathcal{TETC}$ be an infinite set of **composition** event types, and let $\mathcal{TETS}$ be an infinite set of **sequence** event types. $\mathcal{TETC}$ and $\mathcal{TETS}$ are two subsets of $\mathcal{TE}$, such that: $\forall te \in \mathcal{TETC} \cup \mathcal{TETS}$, $\exists te_1, te_2, ..., te_n \in \mathcal{TE}$, $n > 1$, such that:

1. $Dom(te) \subseteq Evt(te_1) \times Evt(te_2) \times ... \times Evt(te_n)$.
2. $Dpara(te) \subseteq Dpara(te_1) \cup Dpara(te_2) \cup ... \cup Dpara(te_n)$.
3. te is structurally defined as:
   $\forall e \in Evt(te)$, $\exists e_1 \in Evt(te_1)$, $e_2 \in Evt(te_2)$, ..., $e_n \in Evt(te_n)$ such that:

$$e = (id, [e_1, e_2, ..., e_n], \bigcup_{i=1}^{n} Para(e_i))$$

   and $\forall e' \in Evt(te)$ with $e \neq e'$, $\exists e'_1 \in Evt(te_1)$, $e'_2 \in Evt(te_2)$, ..., $e'_n \in Evt(te_n)$ such that $e' = (id', [e'_1, e'_2, ..., e'_n], para')$ with $\forall i \in [1..n]$, $e_i \notin \{e'_1, e'_2, ..., e'_n\}$.

[5] A structural schema is defined as a directed acyclic graph $G_S = (S_S, L_S)$ where S_S is the set of object types and L_S the link set of the schema.

Furthermore, if $te \in \mathcal{TETS}$, we have: $Id(e_1) <_{Time} Id(e_2) <_{Time} \ldots <_{Time} Id(e_n) <_{Time} Id(e)$.

The occurrence of a composition or sequence event type is defined by the Cartesian product of the aggregated events. Its parameters are the union of its component parameters. The exclusivity constraint imposes that a composite event type cannot occur from events which are already used.

Grouping Event Types: the grouping represents an event collection, i.e. a conjunction of events belonging to the same type.

Definition 6 Let $\mathcal{TESG}$ be an infinite set of **grouping** event types, subset of $\mathcal{TE}$, such that: $\forall te \in \mathcal{TESG}$, $\exists! te' \in \mathcal{TE}$, such that:

1. $Dom(te) \subseteq \mathcal{P}(Evt(te'))$ where $\mathcal{P}(Evt(te'))$ is the powerset of $Evt(te')$.
2. $Dpara(te) \subseteq \mathcal{P}(Dpara(te'))$.
3. te is structurally defined as:
   $\forall e \in Evt(te)$, $\exists e_1, e_2, ..., e_n \in Evt(te')$ such that:

$$e = (id, \{e_1, e_2, ..., e_n\}, \bigcup_{i=1}^{n} Para(e_i))$$

with $\forall i \in [1..n], Id(e_i) <_{Time} Id(e)$ and $\forall e' = (id', [e'_1, e'_2, ..., e'_n], para') \in Evt(te)$ with $e \neq e'$ then $\forall i \in [1..n], e_i \notin \{e'_1, e'_2, ..., e'_n\}$.

Union Event Types: a disjunction of different event types is described by the union constructor.

Definition 7 Let $\mathcal{TEUT}$ be an infinite set of **union** event types, subset of $\mathcal{TE}$, such that: $\forall te \in \mathcal{TEUT}$, $\exists te_1, te_2, ..., te_n \in \mathcal{TE}$, $n > 1$, such that:

1. $Dom(te) \subseteq Dom(te_1) \cup Dom(te_2) \cup ... \cup Dom(te_n)$.
2. $Dpara(te) \subseteq Dpara(te_1) \cup Dpara(te_2) \cup ... \cup Dpara(te_n)$.
3. te is structurally defined as:
   $\forall i, j \in [1..n]$ if $i \neq j$ then $Evt(te_i) \cap Evt(te_j) = \emptyset$ and
   $Evt(te) = Evt(te_1) \cup Evt(te_2) \cup ... \cup Evt(te_n)$
   with $\forall e \in Evt(te)$, $\exists!\ k \in [1..n]$ such that $e = e_k$ where $e_k \in Evt(te_k)$.

In figure 2, the union type "Up-Down" is an alternative between the two simple types "Up" and "Down". It triggers the descending or ascending lift motion. Thus an event of the union type "Up-Down" may be an event of either type "Down" or type "Up".

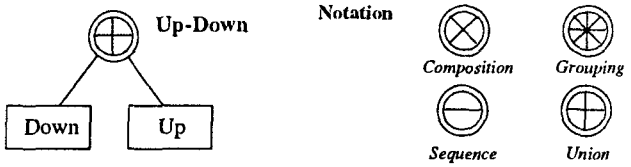

Fig. 2. The Up-Down Event Type

3.4 Represented Event Types

This type, symbolized by a circle, handles another event type through the IS_A specialization link (Cf. Definition 16). Consequently, the designer may use an event type without knowing its complete description. This concept is particularly interesting when considering modularity and re-usability goals.

Definition 8 Let $\mathcal{TER}$ be an infinite set of **represented** event types, subset of $\mathcal{TE}$, such that:
$\forall te \in \mathcal{TER}$, $\exists te_1, te_2, ..., te_n \in \mathcal{TE}$, $n > 0$, called the sources of te, such that:

1. $Dom(te) \subseteq Dom(te_1) \cup Dom(te_2) \cup ... \cup Dom(te_n)$.
2. $Dpara(te) \subseteq Dpara(te_1) \cup Dpara(te_2) \cup ... \cup Dpara(te_n)$.
3. te is structurally defined as: $Evt(te) = Evt(te_1) \cup Evt(te_2) \cup ... \cup Evt(te_n)$ with $\forall e \in Evt(te)$, $\exists e_i \in Evt(te_i)$, $i \in [1..n]$, such that $e = e_i$.

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

3.5 Event Types

From basic and represented types and constructors, event type may be defined. Event and parameter domains are explained as well as instance (at a given instant).

Event Type Specification: an event type is built up from simple, abstract and represented types and constructors which may be recursively applied.

Definition 9 An **event type** $Te \in \mathcal{TE}$ is a directed tree (S_{Te}, E_{Te}) such that:

1. S_{Te}, the set of vertices, is included in the disjoint union of seven sets $\mathcal{TES}$, $\mathcal{TEA}$, $\mathcal{TER}$, $\mathcal{TETC}$, $\mathcal{TETS}$, $\mathcal{TESG}$ and $\mathcal{TEUT}$.
2. E_{Te} is the set of edges called type links.

Event Set and Type Domains

Definition 10 Let Te be an event type, the infinite set of **events** of the type Te, $Evt(Te)$, the **event domain**, $Dom(Te)$, and the **parameter domain**, $Dpara(Te)$ are respectively equal to the set of events and the domains of events and parameters of its root type.

A **type instance** includes all the events of this type which already occurred.

Definition 11 Let Te be an event type, an **instance** J of Te, denoted by J_{Te}, is a finite set of events of type Te, i.e. $J_{Te} \subseteq Evt(Te)$.

The **attached events**, denoted by *Evt_att*, describe, for each vertex of the type, which events occurred. This concept is particularly useful to specify certain complex constraints (for instance, if an event triggering depends on some other specific events).

3.6 Event Fragment

The fragment goal is to describe a part of the system behaviour. One of its advantages is that it can be re-used and manipulated as a whole through the represented type concept. The fragment description focusses on a particular event type, called the heart, which is related to other types. These links represent the event chaining, i.e. a part of the specified behaviour. In the real world, events occur according to particular rules (temporal or not). In IFO_2, these rules are specified by using functions which combine the following features. They can be simple or complex (multivalued); partial or total (mandatory) and immediate or deferred. Furthermore, we have a distinction between triggering functions (whose source is the heart) and precedence functions (whose target is the heart).
There is at the most one precedence edge in a fragment.

Fragment Specification

Definition 12 An **event fragment** is a directed acyclic graph $F_e = (V_{Fe}, L_{Fe})$ with V_{Fe}, subset of $\mathcal{TE}$, the event type set of the fragment and L_{Fe}, the set of fragment links, defined such that:

1. there is only one directed tree $H_e = (V'_{Fe}, L'_{Fe})$ whose root is called fragment heart such that:
   (a) $V'_{Fe} \subseteq V_{Fe}, L'_{Fe} \subseteq L_{Fe}$.
   (b) The source of a triggering edge is either the root of the heart or the root of a target type of a complex edge whose source is the heart root (case of subfragment).
2. $(V_{Fe} - V'_{Fe})$ is either equal to the empty set - and $(L_{Fe} - L'_{Fe})$ too - or it is reduced to a singleton, source of the precedence edge belonging to $(L_{Fe} - L'_{Fe})$ whose target is the fragment heart.

The event fragment is called by its heart.

Figure 3 illustrates a fragment whose heart is an external event type "Floor-Request". In this fragment, there is no precedence function. This fragment describes the lift reactions when a user wishes to go to a floor, i.e. either he calls the lift from a floor or he pushes a button in the cage. The fragment heart is linked with a partial and deferred function to the simple type "Closure". The associated method in the structural fragment "Lift-Cage" closes the lift doors. The function[6] is partial because,

[6] Due to lack of space, the function specification language is not described in this paper.

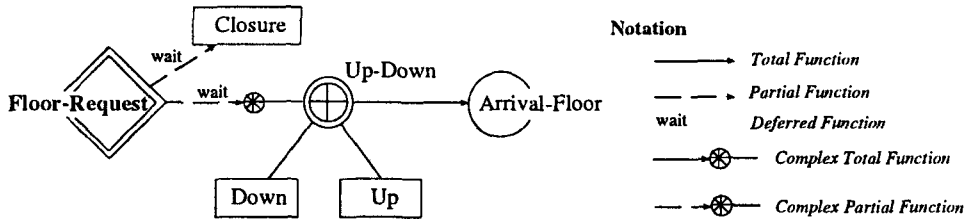

Fig. 3. The "Floor-Request" Event Fragment

in some cases, an event of "Floor-Request" would not trigger a door closure. These cases are the following: (i) the user wishes to go to the floor where he is currently located; (ii) or the door closure stems from another event, i.e. a previous request from the same floor or a previous button activation if the user is already in the lift cage. The function is deferred to take into account the case where the user requests the lift while the cage is moving up or down.
The TEA is also related to the composite type "Up-Down" through a partial, deferred and complex function. It is partial to take into account three cases: cases (i) and (ii) of the previous function and the case where the requested floor is served when satisfying previous current requests. The deferred feature of the function takes into consideration the possible delay between the user request and the resulting lift motion. In fact the methods corresponding to the TESs "Up" and "Down" perform a single floor ascent or descent for the cage. This is why the triggering function is complex. The union type "Up-Down" is heart of a subfragment. The triggering function which relates it to the represented type "Arrival-Floor" (whose consequences are described in another fragment) is total and immediate. This means that any event of the types "Up" and "Down" generates an event of "Arrival-Floor".

A fragment instance is a triplet: the generators of heart events, the heart events themselves and those triggered by heart events. It gives an historical view of the fragment behaviour with causality links between events.

The fragment generated events are those triggered from heart events, i.e. those obtained by applying the triggering functions to heart type instance.

Definition 13 Let F_e be an event fragment whose heart is T_{e0} with root r_{e0}, let $a_1 = (r_{e0}, r_{e1})$, $a_2 = (r_{e0}, r_{e2})$, ... $a_n = (r_{e0}, r_{en})$ be edges whose source is r_{e0}. For each $i \in [1..n]$, let f_{a_i} be the function associated to the a_i edge and let Z_{ei} be the subfragment obtained from the maximal subtree with root r_{ei}.
The **set of events generated** from an event e of the heart type of F_e, is achieved by the function Ψ_{F_e} which is such that:

$\Psi_{F_e}(e) = \emptyset$ if the F_e fragment is reduced to one type
else,

$$\Psi_{F_e}(e) = \bigcup_{i=1}^{n}(\bigcup_{k=1}^{q_i}(ei_k, \Psi_{Z_{ei}}(ei_k)))$$

where $\{ei_k; k \in [1..q_i]\}$ is the event set obtained by applying the f_{a_i} function to the event e (q_i is equal to 1 when f_{a_i} is a simple function) and $\Psi_{Z_{ei}}(ei_k)$ is the set of events generated from the event ei_k in the subfragment Z_{ei}.

The set of events triggered from the $J_{T_{e0}}$ instance with m elements, denoted by $\Psi_{F_e}(J_{T_{e0}})$, is then defined by:

$$\Psi_{F_e}(J_{T_{e0}}) = \bigcup_{j=1}^{m}(\Psi_{F_e}(e_j)).$$

The generator events are those having one image by the fragment precedence function.

Definition 14 Let F_e be an event fragment whose heart is T_{e0} with root r_{e0} and let $a_b = (r_b, r_{e0})$ be the possible edge whose target is r_{e0}. The **set of generator events** of heart events is obtained with the function Υ_{F_e} whose domain is $J_{T_{e0}}$ - a T_{e0} instance with m elements - and codomain is either the empty set if a_b does not exist or I_{T_b} an instance of type T_b with root r_b. Υ_{F_e} is defined by:

$\forall e \in J_{T_{e0}}$, $\Upsilon_{F_e}(e) = e_b$ if $e_b \in I_{T_b}$ exists and is such that $f_{a_b}(e_b) = e$ else $\Upsilon_{F_e}(e) = \emptyset$ where f_{a_b} is the function represented by the a_b edge.

The set of generator events through the a_b edge of the $J_{T_{e0}}$ instance, denoted by $\Upsilon_{F_e}(J_{T_{e0}})$, is defined by:

$$\Upsilon_{F_e}(J_{T_{e0}}) = \bigcup_{j=1}^{m}\Upsilon_{F_e}(e_j).$$

Fragment Instance

Definition 15 Let F_e be an event fragment whose heart is a type T_{e0} and let $J_{T_{e0}}$ be an instance of T_{e0} with m elements.
An **instance** of F_e, denoted by I_{F_e}, is defined by:

$$I_{F_e} = (\Upsilon_{F_e}(J_{T_{e0}}),\ J_{T_{e0}},\ \Psi_{F_e}(J_{T_{e0}})).$$

In the "Floor-Request" fragment, there is no precedence function. Consequently, the fragment instance is equal to the following triplet: $I_{Fr} = (\emptyset,\ J_{Fr},\ \Psi_{Fr}(J_{Fr}))$.
Let us suppose that J_{Fr} is reduced to the event e_{Fr1}, we have: $\Psi_{Fr}(J_{Fr}) = \Psi_{Fr}(e_{Fr1})$ $= (e_{C1}, \emptyset)$ where e_{C1} is of type "Closure". This instance expresses that there is not yet a going up or down order associated to the event e_{Fr1}.

3.7 Event Schema

The overall behaviour of a system is modelized by grouping the partial views described through fragments. The resulting event schema is thus composed by fragments related by IS_A links according to two rules.

Specialization Link: the specialization link represents either the role of an event type in another fragment or the event subtyping.

Definition 16 Let Te be a type of $\mathcal{TER}$ and let $Ts \in \mathcal{TE}$ be a source of Te and heart of fragment, the link of source Ts and target Te is called an **IS_A** link and is denoted by $L_{IS\_A(Te \rightarrow Ts)}$.

Schema Specification

Definition 17 An **event schema** is a directed acyclic graph $G_{Se} = (S_{Se}, L_{Se})$ such that:

1. S_{Se}, the set of schema types, is a subset of $\mathcal{TE}$.
2. L_{Se} is the disjoint union of two sets: L_{Se_A} the fragment link set and L_{Se-IS_A} the IS_A link set.
3. (S_{Se}, L_{Se-A}) is a forest of event fragments.
4. (S_{Se}, L_{Se-IS_A}) follows the two rules: there is no IS_A cycle in the graph and two directed paths of IS_A links sharing the same origin must be extended to a common vertex.

Figure 4 partly shows the IFO_2 event schema "Lift", involving three fragments, each one dedicated to a particular aspect of the lift reactions. "Floor-Request" describes the system behaviour when a user request occurs. "Cage-Arrival" is a particular fragment since it is reduced to its heart which is a TES re-used in other fragments. The corresponding method in the structural fragment "Lift" returns the floor reached by the cage. Finally "Satis-Request" is dedicated to the lift behaviour when the cage arrives at the requested floor. These fragments are related by IS_A links through the represented types "Go-Floor", "Arrival-Floor" and "Arrival".

Schema Instance

Definition 18 Let G_{Se} be an event schema composed by p event fragments F_{e1}, F_{e2},...,F_{ep} with $p > 0$. An **instance** of G_{Se}, denoted by $I_{G_{Se}}$, is such that:

1.
$$I_{G_{Se}} = \bigcup_{i=1}^{p} (I_{F_{ei}})$$
where $I_{F_{ei}}$ is the instance of the F_{ei} fragment.
2. If $L_{IS\_A_{(T'_e \rightarrow T_e)}} \in L_{Se-IS\_A}$, $T_e \in F_{ei}$ and $T'_e \in F_{ej}$ then:
$Evt\_att_{T_e}(I_{F_{ei}}) \subseteq Evt\_att_{T'_e}(I_{F_{ej}})$.

3.8 Satisfaction Fragment

The application behaviour is represented by the event schema. It may be simulated by navigation through the graph, from the root to the leaves, from left to right. An outline of this behaviour consists in a propagation of event triggering. It stops when all the actions reflecting the goal sought by the system, are achieved. These actions

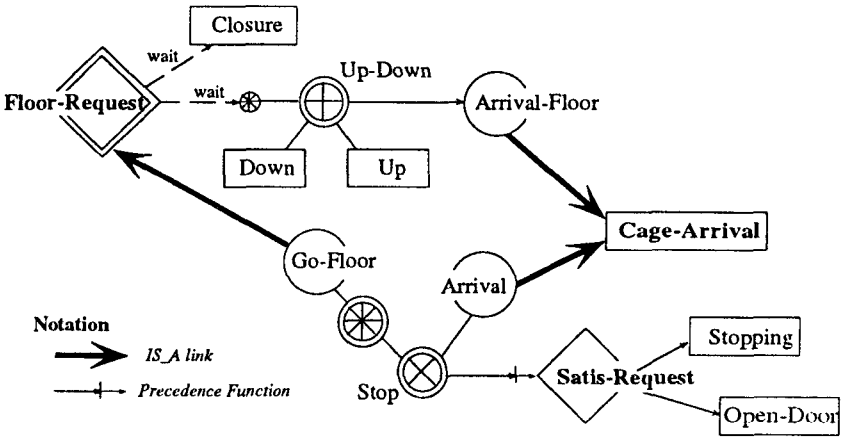

Fig. 4. Part of the IFO_2 event schema "Lift"

are described in the schema, within one or more fragments called satisfaction fragments. The latter have to include a TER. All the events belonging to the IS_A link origin would be satisfied when generated events in the satisfaction fragment occur. This vision has to be refined by taking into account iterations that would possibly be performed during the graph navigation. Iterations aroused by the satisfaction fragment are performed by considering triggering functions which are complex or deferred. The chosen iteration is the first one found along the reverse path.

Definition 19 Let G_{Se} be an event schema composed by p fragments $F_{e1}, F_{e2},...,F_{ep}$ with $p > 0$, let $I_{G_{Se}}$ be the instance of G_{Se} and let $I_{F_{ei}}$ be the F_{ei} instance. A **fragment of satisfaction** F_{ej}_sat for a represented type T_{er} is a G_{Se} fragment such that:

> $T_{er} \in V_{F_{ej}\_sat}$ and $\forall\ k \in [1..p]$, $k \neq j$, such that the source of T_{er}, T_s, belongs to the fragment F_{ek} with: $\forall\ e \in Evt\_att_{T_{er}}(I_{F_{ej}\_sat})$, $Cause(e)$[7] $\in Tr_{F_{ek}}$.
> The fragment of satisfaction F_{ej}_sat is such that:
>
> $$Evt\_att_{T_s}(J_{F_{ek}}) \subseteq Evt\_att_{T_{er}}(I_{F_{ej}\_sat}).$$

When the represented type is the target of several IS_A links, all the sources have to be taken into account. This is performed by the inclusion of attached events belonging to the types which are IS_A link origins.
In our example, the satisfaction fragment is "Satis-Request", which specifies that each user who requests a floor has to reach it, in the end. Iterations are made to trigger the complex function between "Floor-Request" and "Up-Down" until the requested floor is reached.

[7] For each event, the *Cause* function determines its generator event

4 Conclusion

In this paper, we have presented the behavioural part of the IFO_2 conceptual model. Its originalities are a "whole-event" approach, the use of constructors to express complex combination of events and the re-usability and modularity of specifications in order to optimize the designer's work. The IFO_2 model proposes a twofold specification, structural and behavioural, for application modelling. The advantage of this choice is the uniformity of the resulting approach. We think that such a feature is particularly important on a conceptual level. In the two frameworks, static and dynamic, the designer uses the same fundamental concepts, such as re-usability, modularity, identification, etc.
Links between the two specifications are stated as follows. First of all, basic operations are included in the associated structural schema and are used as simple types in the behavioural description. Object types on which event types operate are specified through the parameter domain concept. Finally, conditions on data may be expressed in the triggering functions.
According to us, a formal model is strongly required to avoid ambiguities particularly in a conceptual context. A rigourous approach must provide real assistance to the designer without constraining him to a tedious learning process. Consequently, it is important to offer him an aid that supports the IFO_2 model and guides its intuitive perception. The IFO_2 system is currently developped under Unix/X11R5 with the Interviews programming environment developed in C++. It is an extension of the tool presented in [18].

References

1. Directions for future database research and development. *Special Issue of Sigmod Record*, 19(4), December 1990.
2. S. Abiteboul and R. Hull. IFO: A formal semantic database model. *ACM Transactions on Database Systems*, 12(4):525–565, December 1987.
3. M. Bouzeghoub and E. Métais. Semantic modelling of object-oriented databases. In *Proceedings of the 17th International Conference on Very Large Data Bases (VLDB'91)*, pages 3–14, Barcelona, Spain, September 1991.
4. S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. Technical report, University of Florida, March 1993.
5. E. Dubois, P. Du Bois, and M. Petit. Elicitating and formalizing requirements for C. I. M. information systems. In *Proceedings of the 5th Internation Conference on Advanced Information Systems Engineering (CAiSE'93)*, Lecture Notes in Computer Science, pages 252–274, June 1993.
6. J. Fiadeiro and A. Sernadas. Specification and verification of database dynamics. *Acta Informatica*, 25:625–661, 1988.
7. N. H. Gehani, H. Jagadish, and O. Shmueli. Event specification in an active object-oriented database. In *Proceedings of the ACM Sigmod Conference*, pages 81–90, San Diego, California, June 1992.
8. R. Hull and R. King. Semantic database modelling: Survey, applications and research issues. *ACM Computer Surveys*, 19(3):201–260, September 1987.
9. P. Loucopoulos and R. Zicari. *Conceptual Modeling, Databases and CASE: An Integrated View of Information Systems Development.* Wiley Professional Computing, 1992.

10. B. Pernici. Objects with roles. In *Proceedings of the Conference on Office Information Systems*, pages 205–215, Cambridge, April 1990.
11. P. Poncelet and L. Lakhal. Consistent structural updates for object-oriented design. In *Proceedings of the 5th International Conference on Advanced Information Systems Engineering (CAiSE'93)*, volume 685 of *Lecture Notes in Computer Science*, pages 1–21, Paris, France, June 1993.
12. P. Poncelet, M. Teisseire, R. Cicchetti, and L. Lakhal. Towards a formal approach for object-oriented database design. In *Proceedings of the 19th International Conference on Very Large Data Bases (VLDB'93)*, pages 278–289, Dublin, Ireland, August 1993.
13. C. Quer and A. Olivé. Object interaction in object-oriented deductive conceptual models. In *Proceedings of the 5th International Conference on Advanced Information Systems Engineering (CAiSE'93)*, volume 685 of *Lecture Notes in Computer Science*, pages 374–396, June 1993.
14. C. Rolland and C. Cauvet. Modélisation conceptuelle orientée objet. In *Actes des 7ièmes Journées Bases de Données Avancées*, pages 299–325, Lyon, France, September 1991.
15. G. Saake. Descriptive specification of database object behaviour. *Data & Knowledge Engineering*, 6:47–73, 1991.
16. A. Sernadas, C. Sernadas, and H. D. Ehrich. Object-oriented specification of databases: An algebraic approach. In *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB'87)*, pages 107–116, Brighton,UK, August 1987.
17. C. Sernadas and J. Fiadeiro. Towards object-oriented conceptual modeling. *Data & Knowledge Engineering*, 6:479–508, 1991.
18. M. Teisseire, P. Poncelet, and R. Cicchetti. A tool based on a formal approach for object-oriented database modeling and design. In *Proceedings of the 6th International Workshop on Computer-Aided (CASE'93)*, IEEE, Singapore, July 1993.
19. R. J. Wieringa. A formalization of objects using equational dynamic logic. In *Proceedings of the 2nd International Conference on Deductive and Object-Oriented Databases (DOOD'91)*, volume 566 of *Lecture Notes in Computer Science*, pages 431–452, Munich, Germany, December 1991.