IFO_2: A Uniform Approach for Information System Modelling

Maguelonne Teisseire[1] Pascal Poncelet[1,2] Rosine Cicchetti[1,2]
[1] LIM - Université Aix-Marseille II [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@lim.univ-mrs.fr

Abstract

This paper is devoted to the IFO_2 conceptual model, an extension of the semantic IFO model defined by S. Abiteboul and R. Hull. Its originalities are a uniform approach for both structural and behavioural application specifications, a "whole-object" and "whole-event" approach, the use of constructors to express combinations of objects or events, the modularity and re-usability of specifications in order to optimize the designer's work. Furthermore, it offers an overview of the modelled system. To complement the modelling part, IFO_2 includes a derivation component to perform the implementation of specifications by using an object-oriented or an active DBMS.

1 Introduction

One important aspect in database modelling is the dual representation of applications which need to be specified through both their structure and their behaviour. The problems related to modelling are so complex that conceptual models have often dealt with the static part or the dynamic part of applications but not both of these aspects. Numerous approaches deal with the structural representation of applications. Some of them extend classical conceptual models for example by integrating complex object specification [HK87]. Others are based on Object Oriented Data Base (OODB) models [RC92]. These last approaches enrich OODB models by including, for instance, the explicit expression of semantic constraints. Apart from these approaches, few models give priority to behavioural representation [BM91, LZ92, Per90, PS92, QO93, RC91, Saa91, SF91, SSE87]. They avoid the problem of structural modelling by simply adopting an OODB model. Consequently from a static viewpoint, they inherit the typical drawbacks of implementation models when they are used on a conceptual level: some constructors, proposed by semantic models, are not always provided; semantic constraints such as cardinalities are expressed through methods, i.e. coding which is paradoxical for conceptual approaches; an overview, closely resembling the modelled real world, is not provided for several reasons. The application schema is only seen as the inheritance hierarchy and does not reflect the complex constructions. Futhermore, this schema includes implementation classes (such as the root). From a strictly dynamic viewpoint, the objective

of these models is to represent the system's reactions to events which occur throughout its life. More precisely, the behaviour of the system is seen as the set of reactions of all the modelled objects. In order to represent these dynamic aspects, new concepts are introduced. They capture the semantics of events, of synchronization conditions and of triggering links between events.
In proposing the IFO_2 conceptual model [PTCL93, TPC94], an extension of the IFO model defined by S. Abiteboul and R. Hull [AH87], our aim is to reconcile static and dynamic modelling. IFO_2 is based on two main ideas. The first idea is that the conceptual qualities which are provided are the same for the structural representation as for the behavioural modelling. These are, through some key-words: overview of the real world, faithfulness to this world, expressive capabilities, specification modularity and re-usability, independence from implementation models. Secondly we believe that, despite different contexts, the behavioural modelling process is rather close to the structural representation process. In other words, the designer has to identify the description units, which can be objects or events according to the context. Furthermore, he has to specify the links between these description units. These relations are varied. For instance, in the static framework, they can reflect specializations, generalizations or also compositions of objects. In the dynamic framework, these links are translated as either synchronization conditions or as triggering chaining between events. Following from these analogies, we introduce behavioural concepts which mirror the structural concepts of IFO_2. They not only provide the required expressive power but they also provide the behavioural part of the model with, what we believe, are the qualities necessary for a conceptual approach.
In this paper, our aim is to present an overview of the IFO_2 model, by describing the structural and behavioural modelling components and presenting the twofold derivation process.

2 IFO_2 modelling components

In order to model DB applications, we propose a twofold approach which adopts a "whole-object" philosophy for the structural part and a "whole-event" one for the behavioural modelling. The result is a structural schema which specifies the static aspects and a behavioural schema which describes the application behaviour. These two schemas are closely related but clearly distinct. They are based on concepts similar but adapted to each context (static or dynamic). We now propose an interleaved presentation of these concepts and we illustrate them with the application example of a set of lifts in a building.

2.1 Objects and events

In the static part, an object has a unique identifier which is independent from its value. Furthermore, the domain of a type describes the possible value of its objects.

Example 1 The structural specification of our application example includes the object "Max-User" which gives the maximum number of users allowed for a lift cage. This object can be specified as follows: $o_{MU_1} = (id_{MU_1}, 10)$.

The behavioural description of a system is based on event specification. An event is the representation of a fact that participates in reactions of the modelled system. It occurs in a spontaneous manner (in the case of external or temporal events) or is generated by the application. Events occur instantaneously and as in [Cha89, GJS92], we make the following assumption: no more than one event can occur at any given instant. This allows us to choose the instant of occurrence as the identifier of events. This choice simplifies the definitions of the dynamic concepts. In order to reduce the underlying constraint, we make another assumption: the time scale is infinitely dense. The event specification also includes its value and the objects which react to this event and that are called the event parameters. The values of events can be either methods of their parameters or other events (for complex events).

Example 2 An event involved in the lift reactions is "Up". It accounts for an ascending motion of a lift cage. Let us consider the following specification: $e_{UP_1} = (id_{UP_1}, up, \{cage_1\})$. This means that the event e_{UP_1} occurred at the instant id_{UP_1} and concerned the cage of a particular lift: $cage_1$. The "up" element in this definition corresponds to a method of the object parameter, which is described in the structural schema.

2.2 Basic types

According to our "whole-object" philosophy, objects are not only identified but they also have a type. IFO_2 proposes three basic types (Figure 1):

- The **Printable Object Type** (TOP), which can be materialized. It is used for application I/O (Input/Output are therefore environment-dependent: String, Integer, Picture, Sound, ...). This type is comparable to attribute type of the Entity-Relationship model [TYF86].
- The **Abstract Object Type** (TOA) reflecting an entity of the real world which is not described by its own structure but rather through its properties (attributes). TOA is close to the entity type concept of the Entity-Relationship model.
- The **Represented Object Type** (TOR) which stands for another type. By using a TOR, the designer can handle (and thus re-use) any different type without knowing its precise description.

Example 3 Figure 1 illustrates the IFO_2 basic object types. "Max-Weight" is a printable type which gives the maximum weight allowed for a lift cage. "Cage" is an abstract type corresponding to a real entity and the TOR "Lift-Cage" is defined to re-use the previous type.

Figure 1: Basic Object Type Examples

The three basic event types proposed in the behavioural part of the model respect the philosophy of their structural equivalent types and the associated graphic formalism (Figure 2) mirrors the structural representation. These event types are the following:

- The **Simple Event Type** (TES) which represents the only events that can be materialized through their direct effects on objects. In fact a TES describe the events that trigger a method included in the IFO_2 structural description. This means that we do not consider operations in the behavioural part of the model but only the events triggering these operations.
- The **Abstract Event Type** (TEA) is used to specify external and temporal events or events that generate other ones. Such internal types are interesting only through their consequences.
- The **Represented Event Type** (TER) symbolizes any other type which may then be re-used without knowing its precise description.

Example 4 A simple event type involved in the description of the lift behaviour is "Up", which stands for the ascending motion of the lift cage and maps with a method of the structural schema. The TEA "Floor-Request" represents external events which occur when users request a floor (inside or outside the cage). "Satis-Request" corresponds to the internal events produced when users reach the requested floor and "Arrival-Floor" stands for another type.

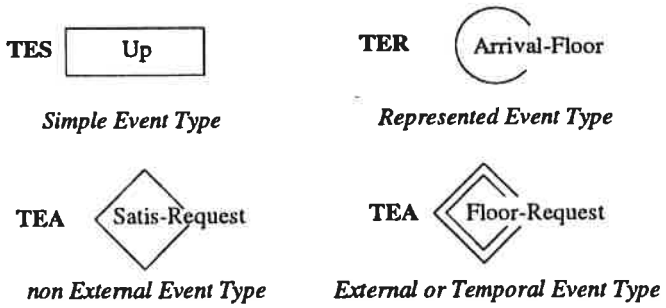

Figure 2: Basic Event Type Examples

2.3 Complex types

In order to represent complex objects, the structural part of IFO_2 takes into account five 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.

- The **aggregation** and **composition** types represent the aggregation abstraction of semantic models [HK87] 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.

- The **collection** and **grouping** types represent the set-of constructor of object models with an exclusive constraint for the grouping.
- The **union** type is used for similar handling of stucturally different types. This constructor represents the IS_A generalization link enhanced with a disjunction constraint between the generalized types.

Example 5 A property involved in the description of a lift is its "Engine". It is specified as the composition of a printable type "Axle" and a grouping type "Generators" which is built up from "Generator". This type is a composition of two printables types "Reel" and "Magnet".

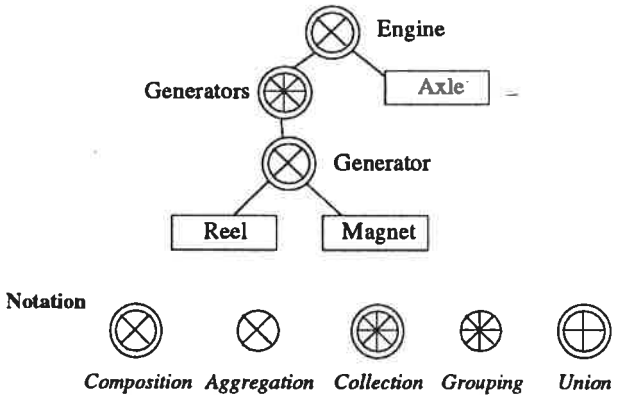

Figure 3: Complex Object Type Example

To model the behaviour of a system, it is necessary to express event synchronization conditions, i.e. different variants of event conjunction and disjunction. To answer this need, we choose to represent complex events by using **constructors**. With this approach, we provide not only the required expressive power but also the uniformity with respect to the IFO_2 structural modelling. The associated graphical formalism is illustrated in Figure 4.
The event constructors, which may be recursively applied, are the following:

- The event **composition** type reflects the conjunction of events belonging to different types.
- The event **sequence** type is defined as the previous one but with a chronological constraint on the occurrences of the component events.
- The event **grouping** type expresses conjunctions of events belonging to the same type. It is comparable to the HiPAC closure constructor [CBB+90].
- The event **union** type expresses a disjunction of events of different types.

Example 6 Let us imagine that the designer would like to specify the descending or ascending lift motion. He may use the union type "Up-Down" which is an alternative

between the two simple types "Up" and "Down". Each one of these types triggers a method which performs a single floor motion for the cage.

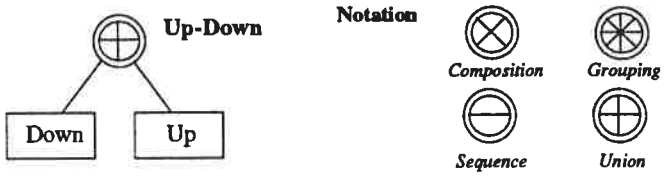

Figure 4: Complex Event Type Example

2.4 Structural and behavioural fragments

The concept of fragment, inherited from the IFO model, is very important for the modularity of specifications. In fact, the fragment can be really considered as a **unit of description** of either the structure or the dynamics of the application since it describes a complete sub-set of the modelled system: either an entity of the real world in the static part of the model or a "subbehaviour" in the dynamic part.
Structural and behavioural fragments play a similar role by providing the modularity of specifications, but their semantics is different in the two contexts, as it is explained below.

Building relations are not the only links to be modelled between objects. It is also necessary to specify certain objects as properties of other ones. IFO_2 answers this need through the concept of **structural fragment**. Its aim is to describe properties (attributes) of the principal type called **heart**, by linking the corresponding object types with functions. These functions express particular semantic constraints since they can combine the following features:

- simple or complex, i.e. mono or multivalued;
- partial or total. They express either 0:N or 1:N links between objects.

Example 7 The figure 5 describes the fragment of heart "Cage" having "Position", and "Floor" as properties. The printable object type "Position" corresponds to the current floor where the lift cage is located. "Floor" accounts for all the floors which can be desserved by the cage. This is why the function between "Cage" and "Floor" is complex. An object of the type "Button" is associated to each floor. This is expressed through the simple and total function between "Floor" (heart of a subfragment) and "Button".

Furthermore, a set of methods (only their signature) is associated to each fragment. They describe the basic operations on the object types involved in this fragment.

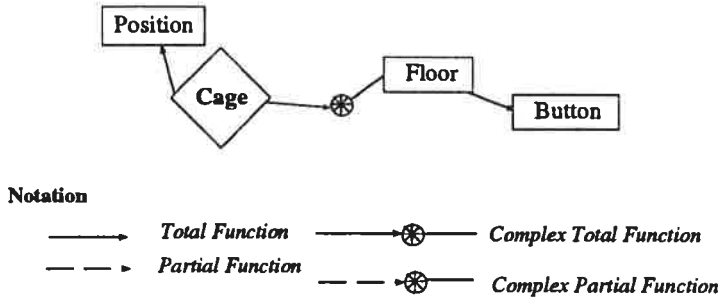

Figure 5: The "Cage" Structural Fragment

Example 8 Among the methods of the "Cage" fragment illustrated Figure 5, "Open" and "Close" perform the opening or the closure of the cage doors.

From a behavioural view point, the relation between events are not only descibed through synchronization conditions but also through the chaining of events. These causality links are expressed with functions. In fact, the event types are interconnected by functions through the **event fragment** concept, focused on a principal type called **heart**. Functions of event fragments express general conditions on the event chaining since they can combine the following features:

- simple or complex (mono or multivalued), i.e. an event of their type origin triggers one or several events of their target;
- partial or total , i.e. an event of their type origin may or must trigger an event of their target;
- and deferred or immediate, if there is a delay or not between the occurrences of the origin and target events.

In addition, we make a distinction between **triggering** and **precedence functions**, which roughly express the fact that an event of the fragment heart triggers the occurrence of other events or that it is preceded by the occurrence of other events. In order to emphasize this, let us consider an external or temporal event. By its very nature, it cannot be triggered by another modelled event, therefore it is sometimes necessary to express that its occurrence is necessarily preceded by other events. Contrary to triggering functions, a precedence function, if it exists, is unique in a fragment since triggered events may be constrained by different specific conditions while preceding events must only be synchronized by using the mentioned constructors.

Example 9 Figure 6 illustrates a fragment whose heart is the external event type "Floor-Request". In this fragment, there is no precedence function. This fragment describes the lift reactions when a user requests a floor inside or ouside the cage. The fragment heart is linked with a partial and deferred function to the simple type "Closure". The associated method in the structural schema closes the lift doors. The function is partial because, 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. The function is deferred to take into account the case where the user requests the lift while the latter is moving up or down.
The TEA is also related to the composite type "Up-Down" which specifies an alternative between the two TESs "Down" and "Up". The triggering function between the heart and the union type is partial, deferred and complex. 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" (standing for a type described in another fragment) is total and immediate. This means that any motion of the lift cage is concluded by the arrival of the cage to a particular floor.

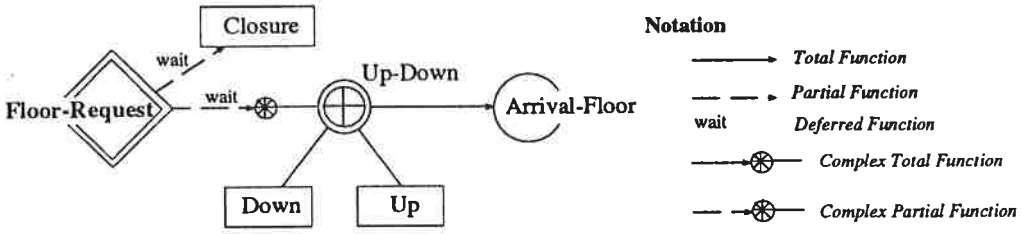

Figure 6: The "Floor-Request" Behavioural Fragment

2.5 IFO_2 structural and behavioural schemas

The role of fragments (either structural or behavioural) is to describe a subset of the modelled application that can then be used as a whole by means of the **represented type** concept (either TOR or TER). More precisely, represented types are related to fragment hearts via IS_A links according to building rules. Consequently, it is possible to manipulate an object or event type without knowing its description which is inherited from another type. Thus, the designer may defer a type description or entrust it to somebody else, while using a represented type which symbolizes it. Through the concept of represented type, the re-usability of specifications is actual for the structural description as well as for the behavioural description. Furthermore, IFO_2 takes into account the multiple inheritance since represented types may have several sources.

In order to model the structure of the application, the partial views provided by the structural fragments are combined (via IS_A links) within a **structural schema.**

Example 10 Figure 7 proposes the IFO_2 structural schema for our application example. It is made up of two fragments "Lift" and "Cage". They are related by an IS_A link through the represented type "Lift-Cage". The fragment of heart "Lift" has "Id-Number", "Load" (built up as an aggregation of "Max-Weight" and "Max-User"

types), "Lift-Cage", a set of "Lift-Button" and the complex type "Engine" as properties. Among the methods of the fragment, "up" and "down" are triggered to perform a single floor motion for the cage.

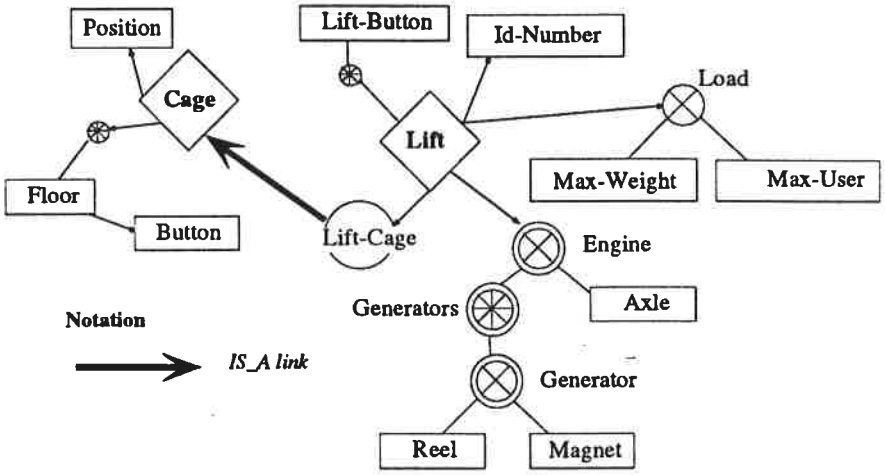

Figure 7: The "Lift" Structural Schema

In the same way as for the structural inheritance, the behavioural inheritance is represented through IS_A links, called event IS_A links. These links relate represented event types to event fragment hearts. An inherited behavioural aspect may be re-defined or refined by specifying the concerned represented type as the heart of a new fragment having other preceding or triggered types. From the choice of a twofold specification, behavioural and structural, stems the original aspect of IFO_2 behavioural inheritance: it is independent from the structural inheritance hierarchy. Thus, it is possible to re-use parts of the modelled behaviour even if they do not concern specialized objects. Consequently the re-usability of dynamic specifications is not limited by static considerations. In order to model the general behaviour of the application, the partial views provided by the fragments are combined (via IS_A links) within an **event schema**.

Example 11 Figure 8 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 simple event type re-used in other fragments. The corresponding method in the structural fragment "Lift" is an alerter which returns the floor reached by the cage. Finally "Satis-Request" is dedicated to the lift behaviour when the cage arrives at the requested floor. The origin of the precedence function is a composite type "Stop", which combines several events of the TER "Go-Floor", in fact several floor requests, and a cage arrival. These fragments are related by IS_A links through the represented types "Go-Floor", "Arrival-Floor" and "Arrival".

The behavioural schema mirrors the structural schema, but with a specific characteristic: it captures the semantics of application dynamic working that we call activity.

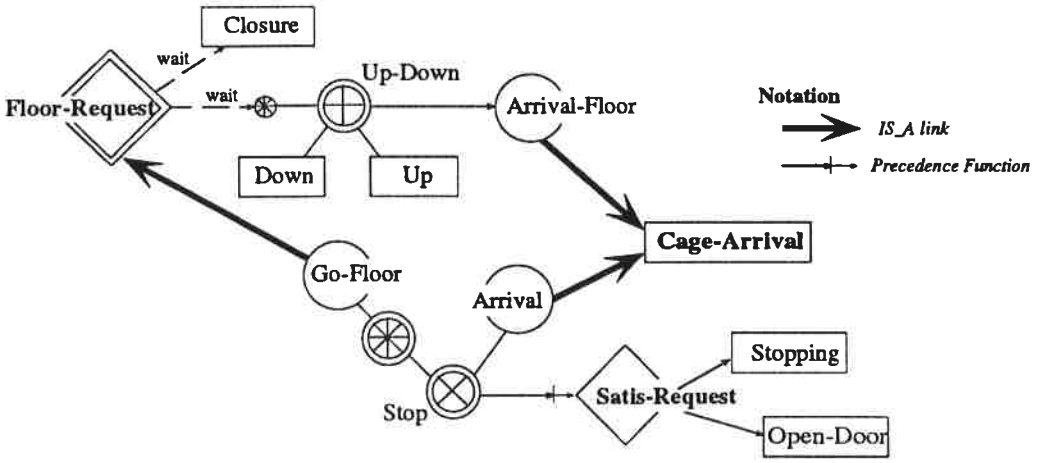

Figure 8: The "Lift" Behavioural Schema

2.6 Activity of an event IFO_2 schema

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 are described in the schema, within one or more fragments called satisfaction fragments. In our example, there is one satisfaction fragment: "Satis-Request", which specifies that each user who requests a floor has to reach it, in the end.

This section provides further details of this general principle. Within each fragment, the triggering propagation is of course oriented by the precedence and triggering functions. This propagation stems from an external or temporal event or a combination of such events. In this case, the underlying TEA (or the type built up from these TEAs) is either the heart of a fragment without a precedence function or it is the origin of the precedence function in a fragment. It is considered as an **entry** of the IFO_2 event graph.

Example 12 In our example, the only fragment illustrating this case is "Floor-Request" which is consequently the only entry of the graph. In fact, the whole triggering cascade stems from the external event: a user requests a floor. Let us suppose that the user is the only one at this instant and that he wishes to go to a floor above the one where he is. Under this assumption, the external event triggers the door closure and then the occurrence of an "Up" event. This event causes the generation of an "Up-Down" event. In a general way, events of composite type stem from the occurrence of the component types. The propagation continues on the level of the subfragment by the triggering of an "Arrival-Floor" event.

Therefore, in the general case, the event propagation is triggered, within a fragment, by the occurrence of an event in another fragment. The behaviour is then simulated by the navigation along the IS_A link associating these two fragments, i.e. the heart of one to a

TER of the other. Along an IS_A event link, the navigation may follow one direction or the other depending on the "position" of the TER within its own fragment. If it is the target of a triggering function, directly or by construction, then the event occurrence of the TER systematically generates an event occurrence of the heart type of the related fragment. Therefore, the navigation takes place from the IS_A link target to its source and there is equality between the sets of occurred events of both the fragment heart and the TER, if the latter has only one source[1].

Example 13 In our example, one TER is the target of a triggering function: "Arrival-Floor". According to our previous assumption, an event of this type has just happened. This triggers the occurrence of a fragment heart "Cage-Arrival" event by navigating along the IS_A link between these two types.

Let us now consider a TER that is the origin of the precedence function, or involved in the construction of this origin. This means that one of its events can only be triggered by the occurrence of an event belonging to the fragment heart to which it is related. The navigation along the IS_A link takes place from its source to its target and there is inclusion or equality between the sets of events occurred for the TER and the heart, if the former has only one source.

Example 14 The two TERs, illustrating this situation in our event schema, are "Go-floor" and "Arrival". They are both used to build up the "Stop" type origin of the precedence function in the fragment "Satis-Request". The occurrence of an event of these types is necessarily caused by an event of "Floor-Request" or "Cage-Arrival".

From this first outline of the system activity, it results that "everything begins" with the occurrence of an external or temporal event.
When a triggering cascade is started, it must stop in the end. "Everything ends" on the level of satisfaction fragments since they model, as previously mentioned, the ultimate goal of the system. In fact, such fragments not only describe the manner in which the propagation stops. They also specify this obligation to stop, by integrating the following constraint: a satisfaction fragment has to include a TER which is origin of the precedence function or is involved in the construction of this origin. The obligation to stop is then partially taken into account by the fact that any event of the related fragment hearts must also be an event of the TER. The heart events are considered as being satisfied when the TER corresponding events actually trigger the set of generated events in the satisfaction fragment. 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.

Example 15 Let us resume the activity of our schema example where we left it, i.e. after the occurrence of a "Cage-Arrival" event. From the latter stems an event of the

[1] We do not explain the case where a TER has several sources since its description requires the concept of attached events which is not presented in this paper.

"Arrival" represented type. Similarly, from the initial floor request stemmed a corresponding event for the TER "Go-Floor" in the satisfaction fragment. During the cage motion, let us suppose that another person calls the lift from the floor that is requested by the first user. This call does not yet generate any event in the "Floor-Request" fragment because the lift is engaged, but it triggers a corresponding event for the TER "Go-Floor". At this stage, none of the two floor requests may be satisfied because the cage is not yet at the desired floor. This condition is expressed in the precedence function of the satisfaction fragment by comparing the structural parameters standing for the floor of the event "Arrival" and of the events "Go-Floor". A first iteration is then performed, by following the IS_A links, in order to trigger again the complex function between "Floor-Request" and "Up-Down". Such a process is performed as many times as necessary to reach the required floor. When this happens, the condition of the precedence function in the fragment "Satis-Request" is true and the propagation carries on generating a fragment heart event. Let us examine its preceding event, which is of the "Stop" type. This type is defined as a composition of the "Arrival" type and a "Go-Floor" grouping. Therefore, the preceding event in question is built up from our two user requests combined with the last "Arrival" event. The event of the type "Satis-Request" immediately generates a "Stopping" event and then triggers the opening of the lift doors.

3 Derivation of IFO_2 schemas

In order to perform the implementation of applications described with IFO_2, we define a twofold transformation process. Structural IFO_2 schemas can be translated into relational [Pon93] or OODB schemas while event schemas are transformed in E-C-A rules. Since our aim is to show the feasibility of translation, we opt for the O_2 model as the target of the structural derivation and for the behavioural derivation we choose E-C-A rules similar to HiPAC ones [CBB+90] (i.e. adopting the philosophy but without strictly following its syntax) because it is a reference in the active DBMS researchs [Cha89, CBB+90, DBM88, DHL91, DPG91, GJS92].

3.1 Structural derivation

In this section, we describe the transformation process into the O_2 model [Deu90]. The structural derivation of IFO_2 follows the same principle as the transformation performed from MORSE to O_2 [BMHL90]. The formalization of the mapping between IFO_2 and O_2 is given in [PTCL93]. In this paper, we just illustrate this derivation process with our application example.

3.1.1 Presentation

Generally speaking, an IFO_2 structural schema is mapped into an O_2 schema generating for each IFO_2 fragment at least one O_2 class. For each class, we have to specify: its name, its type, its location into the hierarchy and its methods. Furthermore, checking methods and additional classes have to be defined for composition, grouping and union type.
The additional classes are generated since the O_2 model is value and object-oriented.

To account for exclusivity constraints, methods use the identifier of objects. Thus, it is necessary to define classes where object instances can be created. Now, let us consider the mapping of these constructors. A union type is mapped into a particular class which includes boolean attributes adopting a similar principle as in [CB91]. These attributes indicate the object type, i.e. they describe the component of the union type. A method checking the exclusive constrainst is also included. The grouping and composition types generate classes and methods checking that an object can only take part in a unique construction.
The class location in the hierarchy is provided by the represented types. If a represented type is not a fragment heart, it is mapped as attribute using the O_2 composition link. Otherwise, the associated class has to inherit the generic (i.e. the source of the IS_A link) class. The multiple inheritance is considered in the same way.

3.1.2 Illustration

The described transformation is applied on the IFO_2 structural schema example (Cf. Figure 7). The derivation of the "Cage" fragment generates the following class:

Class C_Cage
Public type tuple(Position: integer,
Door-Status[2]*: string,*
Floor: set(tuple(Floor: integer,
Button: integer))
Method public Verif-Simple-Floor(object: C_Floor): boolean,
Method public Stop: boolean,
Method public Open,
Method public Close
end;

By considering the "Lift" fragment, the translation of the composition generates the following class.

Class C_Lift
Public type tuple(Id-Number: integer,
Status: string,
Lift-Button: set(tuple(LiftButton: integer)),
Load: tuple(Max-Weight: integer, Max-User: integer),
Engine: C_Engine,
Lift-Cage: C_Cage)
Method public up (Lift-Cage),
Method public down (Lift-Cage),
Method public arrival (Lift-Cage): Lift-Cage.Position
end;

"Engine" being a composition type, we have two classes which describe the components

[2]This attribute is added when considering the behavioural derivation (see next section).

of the engine. Furthermore, the method Verif-Compo-Engine checks that an object of "Generators" and an object of "Axle" take part in a single engine.

Class C_Engine
Public type tuple(Engine: tuple(Generators: C_Generators,
Axle: C_Axle))
Method public Verif-Compo-Engine: boolean,
Method public Stop-Engine,
Method public Start-Engine
end;

The C_Axle class is generated since each axle must be identified.

Class C_Axle
Public type tuple(Axle: string)
end;

The grouping type "Generators" is translated into a class having a checking method.

Class C_Generators
Public type tuple(Generators: set(Generators: C_Generator))
Method public Verif-Set-Generators: boolean
end;

Finally, the following classes are generated:

Class C_Generator
Public type tuple(Generator: tuple(Reel: C_Reel))
Magnet: C_Magnet))
Method public Verif-Compo-Generator: boolean
end;

Class C_Reel
Public type tuple(Reel: string)
end;

Class C_Magnet
Public type tuple(Magnet: string)
end;

3.1.3 Presentation

On the basis of an IFO_2 event schema, a set of E-C-A rules can be produced. Generally speaking, this process starts with the entries of the IFO_2 graph, examining the corresponding fragments. It continues by transforming the fragments linked to the entry fragments, following the IS_A links from target to their source. All fragments that

are not yet derived are then examined. More precisely, each fragment gives rise to the creation of at least one E-C-A rule, except in the case where it is reduced to a TES. In general, however, a fragment generates several E-C-A rules. Without presenting the algorithm in detail (it is given in [TPC94]), we give its general principles. In the IFO_2 model, every fact involved in the behavioural specification is modelled as an event. During the derivation process, some IFO_2 events are translated in events while other are translated in actions. The criterium used to perform one translation or the other is the location of the IFO_2 event type in the fragment. If the considered type is a triggering type (source of a precedence function or heart of fragment), then it corresponds to the event part of an active rule. In this case, the possible used constructors give the type of combinations of events in the E-C-A rules (conjunction, disjunction, sequence or grouping). If the considered type is a triggered type (target of a triggering function), then it corresponds to the action part of an active rule. This action part can be a method triggering (for TES), an activation or triggering of an active rule. The iteration of a behavioural aspect, expressed by the grouping constructor, is translated by including an activation order of the rule in its own actions. Such a rule is called recursive. In IFO_2 the conditions of event triggering are expressed when specifying the functions, in a language that we do not present in this paper [TC94]. The conditions must be highlighted in a preliminary stage of application of the algorithm. Lastly, the characteristics of the functions have an influence on the derivation that is performed. The deferred triggering functions of a fragment can introduce a deferred coupling mode between the Event component and the Condition component. They can also be translated in the case of generation of a recursive rule, by activation (instead of triggering) of a new rule. The complex functions are translated according to the same principles as the grouping types, since they represent an iteration on the level of the target event types, i.e. through generation of recursive rules triggered by an action of the rule corresponding to the fragment. Let us note that the rules which we call recursive have the peculiarity of not having triggering events; this is because they have to be repeatedly executed several times. In the ODE model [GJS92], they could correspond to "perpetual" rules. In the ATM model [DHL91], such rules would be described in an even more natural manner because the activity concept allows a loop to be introduced.

3.1.4 Illustration

We apply the previous algorithm to the IFO_2 behavioural schema. Firstly, we describe, in an intuitive way, the preliminary step for the state specifications.
In fact, conditions over event occurrences are expressed through functions by using manipulation operators on events [TC94]. The problem is then to translate constraints over events into constraints over object states. The function between "Floor-Request" and "Closure" includes two conditions. Firstly, a floor request would trigger the door closure only if the lift door is opened. In our specification language the expression of this condition looks like: "was there a door opening since the last closure ?". An attribute, "Door-Status" whose values would be "closed" or "opened" appears suitable for capturing the required semantics. The second condition to be taken into account is the following: the door closure is performed only if there are still requests to be satisfied. These requests are identified, in the function specification language, by comparing the events of the "Floor-Request" type and the events of "Go-Floor" which actually triggered

a "Satis-Request" event. To translate this constraint on events, it is relevant to introduce in the structural fragment "Lift" an attribute called "Status" which indicates whether the cage is engaged or not ("engaged" or "waiting"). These attributes, shaded in figure 9, are called "artificial", since they do not correspond to properties identified during the structural analysis.

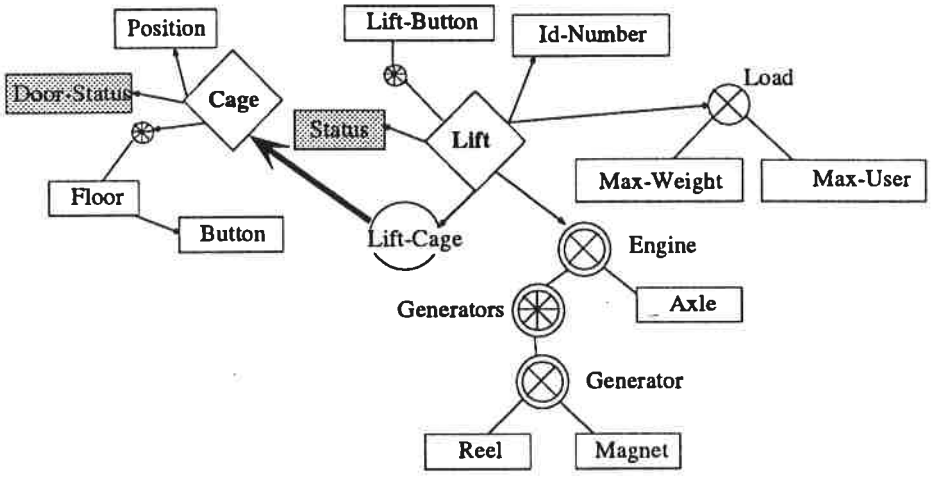

Figure 9: The "Lift" example with artificial types

Now, we examine the application of the algorithm to the "Lift" behavioural schema. In our example, the only entry of the graph is "Floor-Request". So, the derivation starts by the translation of the associated fragment by evaluating the following function:
$RULE(Floor\text{-}Request,$
$((EVENT(Floor\text{-}Request), immediate), (true, immediate), \emptyset)).$
The generated E-C-A rule includes the activation of two recursive functions expressing the triggering functions of the fragment.

r Floor-Request

E:	Floor-Request
Immediate	
C:	True
Immediate	
A:	enable r $_{Closure}$
	enable r $_{Up\text{-}Down}$

The recursive rule $r_{Closure}$ is done by:
$RULE_{Recursive}(Fermeture, (\emptyset, (C_{Fermeture}, immediate), \emptyset))$
which reflects the condition produced from the function between "Floor-Request" and "Closure" and includes in its actions the closure method call and its own re-activation.

$r_{Closure}$
E:
C: Lift.Door-Status='open' ∧ ((Lift-Cage.Status='waiting'
∧ Para(Floor-Request, Floor) <> Lift-Cage.Position)
∨ Lift-Cage.Status='engaged')
Immediate
A: Closure
enable $r_{Closure}$

In the same way, the translation of the composition "Up-Down" generates a recursive rule. Let us note that the union constructor derivation triggers two rules with conditions r_{Up} and r_{Down} which are mutually exclusive.
The TER "Arrival" is translated into the method call of its source since the concerned fragment is reduced to its heart.

$r_{Up\text{-}Down}$
E:
C: Lift.Door-Status='closed' ∧ Lift-Cage.Status='engaged'
Immediate
A: fire r_{Down}
fire r_{Up}
Cage-Arrival
enable $r_{Up\text{-}Down}$

The derivation ends with the translation of the satisfaction fragment by applying:
$RULE(Satis\text{-}Request, ((EVENT(Stop), immediate), (true, immediate), \emptyset))$.

The generated rule includes, in the event part, the composition which triggers the fragment and specifies which occurrences of "Floor-Request" may actually cause the triggering of the "Stop" and "Open" methods.

4 Related work

This section briefly presents a survey of conceptual models in order to give some elements of comparison between IFO_2 and the related work. This presentation firstly focuses on structural modelling approaches and then describes the behavioural models. Among structural approaches, 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 [LV87, Teo90, Twi89]. However, the classical models in this group generally suffer from the lack of concepts (object-identity, re-usability,...) which are efficient for advanced application modelling.
The second class encompasses object-oriented currents. 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 modelling. Generally, additional methods are used to express semantic structural constraints. These approaches 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 [HK87]. The implication for the database designer is the necessity of specifying preliminary representation choices. These choices sometimes cut off parts of the real world being modelled.
The objective of the IFO_2 structural model is actually to reconcile apparently opposed ideas: an optimal data representation and a complete real world modelling. IFO_2 attempts to preserve the acquired strengths of semantic approaches, while integrating concepts of the object paradigm [ABD+89].
Conceptual models which give priority to behavioural specification [FS88, Per90, QO93, RC91, SF91] adopt an OODB model to represent the structural part of applications [RC92]. They deal with problems which are nearly similar to those of concurrent system design and software engineering [LZ92].
In these approaches, the behaviour of applications is perceived as the set of reactions of the modelled objects when certain events occur. In fact, these models make a distinction between local and global behaviours. Local behaviours focus on the object dynamics within classes while global behaviours represent the interactions between classes. Consequently two kinds of events are to be considered: local and shared events.
The object dynamics may be perceived through states and transitions between states. The local behaviour consists in valid transitions and an additional mechanism is required to coordinate the state changes of objects belonging to different classes when shared events occur.
These models make use of statecharts or temporal logic to represent the system's behaviour.
This perception of behaviour is rather simple, clear and matches well with object-oriented models. For local behaviours a behavioural facet, integrated in the class description, specifies events, occurrence conditions and triggered actions (methods or other events which are produced). The description of global behaviours is either encapsulated in the classes (through interaction equations in [SSE87]) or given outside the classes [RC91]. The latter case avoids to choose one of the class involved in the described interaction but the behaviour in question is also excluded from the inheritance hierarchy.

Nevertheless these models have some drawbacks. First of all, they do not provide an overview of the system's behaviour since it is split into classes: priority is given to a complete vision of objects (structural and behavioural) rather than an overview of the system. Events are not represented in a uniform way: two abstraction mechanisms are necessary for local and shared events. Futhermore, behavioural inheritance depends on the structural representation. In fact behavioural aspects may be re-used only for specialized objects according to the static inheritance hierarchy. In these models, the conditions on event occurrences make use of object states. These states and the underlying objects are not always easy to identify since they do not necessarily correspond to attributes existing in the real world. Consequently, it is necessary to introduce "artificial" objects in the structural representation in order to express behavioural constraints. This perception of object behaviour through states has two drawbacks: the problem of making a complete inventory of states and, above all, a structural representation which is no less faithful to the real world because of the "artificial" objects. Finally, the concepts defined for the static and dynamic representations are very different and the designer's

required skills have to be extended.
In proposing the IFO_2 behavioural model, our basic idea is that a conceptual model must offer the same qualities for the structural representation as for the behavioural modelling. More precisely, it must provide the designer with an overview of specifications not only for the structural part but also for the application behaviour. These specifications must be as faithful as possible to the real world. The modularity and re-usability mechanisms have to be as powerfull in the static as in the dynamic contexts.

5 Conclusion

In this paper, we have described the IFO_2 conceptual model. Its original aspects are a "whole-object" and "whole-event" approach, the use of constructors to express complex combinations of objects and events and the re-usability and modularity of specifications in order to optimize the designer's work. The IFO_2 model offers a uniform specification of both structural and behavioural parts of applications. We believe that such a uniformity is particularly important on a conceptual level. In the two frameworks, structural and behavioural, the designer uses the same fundamental concepts, such as re-usability, modularity, identification, etc. Types, constructors and fragments are defined by adopting an analogous formalism and they have the same semantics or at least the same philosophy in the static and dynamic parts of the model. A homogeneous graphical representation is presented and can facilitate the dialogue between designers in order to better take advantage of specification modularity.
Links between the structural and behavioural 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 concept. Finally, conditions on objects may be expressed in the specification of fragment functions.
The derivation component which generates E-C-A rules from the IFO_2 behavioural specifications can be associated to the transformation of IFO_2 structural schema into OODB models [PTCL93] in order to perform a complete implementation of applications.

References

[ABD+89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S.B. Zdonik. The Object-Oriented Database System Manifesto. In *Proceedings of the 1st Deductive and Object-Oriented Databases Conference (DOOD'89)*, pages 40–55, Kyoto, Japan, December 1989.

[AH87] S. Abiteboul and R. Hull. IFO: A Formal Semantic Database Model. *ACM Transactions on Database Systems*, 12(4):525–565, December 1987.

[BM91] 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.

[BMHL90] M. Bouzeghoub, E. Métais, F. Hazi, and L. Leborgne. A Design Tool for Object Databases. In *Proceedings of the 2nd International Conference on Advanced Information System Engineering (CAiSE'90)*, volume 436 of *Lecture Notes in Computer Science*, pages 365–392, June 1990.

[CB91] C. Collet and E. Brunel. Définition et manipulation de formulaires avec FO_2. *TSI - Technique et Science Informatique*, 10(2):97–124, 1991.

[CBB+90] S. Chakravarthy, B. Blaustein, A. P. Buchmann, M. Carey, U. Dayal, D. Goldhirsch, M. Hsu, R. Jauhari, and al. HiPAC: A Research Project in Active, Time-Constrained Database Management. Technical report, Xerox Advanced Information Technology, Cambrige, MA, August 1990.

[Cha89] S. Chakravarthy. Rule Management and Evaluation: An Active DBMS Perspective. *Sigmod Record*, 18(3):20–28, September 1989.

[DBM88] U. Dayal, A. P. Buchmann, and D. R. McCarthy. Rules Are Objects Too: A Knowledge Model for An Active, Object-Oriented Database System. In *Advances in Object-Oriented Database Systems*, volume 334 of *Lecture Notes in Computer Science*, pages 129–143, September 1988.

[Deu90] O. Deux. The Story of O_2. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):91–108, 1990.

[DHL91] U. Dayal, M. Hsu, and R. Ladin. A Transactional Model for Long-Running Activities. In *Proceeding of the 17th International Conference on Very Large Data Bases (VLDB'91)*, pages 113–122, Barcelona, Spain, September 1991.

[DPG91] O. Diaz, N. Paton, and P. Gray. Rule Management in Object-Oriented Databases: A Uniform Approach. In *Proceedings of the 17th International Conference on Very Large Data Bases (VLDB'91)*, pages 317–326, Barcelona, Spain, September 1991.

[FS88] J. Fiadeiro and A. Sernadas. Specification and Verification of Database Dynamics. *Acta Informatica*, 25:625–661, 1988.

[GJS92] N. H. Gehani, H.V. 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.

[HK87] R. Hull and R. King. Semantic Database Modelling: Survey, Applications and Research Issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

[LV87] P. Lyngbaek and V. Vianu. Mapping a Semantic Database Model to the Relational Model. *Sigmod Record*, 16(3):132–142, 1987.

[LZ92] P. Loucopoulos and R. Zicari. *Conceptual Modeling, Databases and CASE: An Integrated View of Information Systems Development.* Wiley Professional Computing, 1992.

[Per90] B. Pernici. Objects With Roles. In *Proceedings of the Conference on Office Information Systems*, pages 205–215, Cambridge, MA, April 1990.

[Pon93] P. Poncelet. *Contribution à la conception d'applications avancées : modèle, mécanismes d'évolution et dérivation.* PhD thesis, Université de Nice-Sophia Antipolis, Mai 1993.

[PS92] C. Parent and S. Spaccapietra. *ERC+: An Object-Based Entity Relationship Approach.* in [LZ92], 1992.

[PTCL93] 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.

[QO93] 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, Paris, France, June 1993.

[RC91] C. Rolland and C. Cauvet. Modélisation Conceptuelle Orientée Objet. In *Actes des 7ièmes Journées Bases de Données Avancées*, pages 299–325, Lyon, France, Septembre 1991.

[RC92] C. Rolland and C. Cauvet. *Trends and Perspectives in Conceptual Modeling.* in [LZ92], 1992.

[Saa91] G. Saake. Descriptive Specification of Database Object Behaviour. *Data & Knowledge Engineering*, 6:47–73, 1991.

[SF91] C. Sernadas and J. Fiadeiro. Towards Object-Oriented Conceptual Modeling. *Data & Knowledge Engineering*, 6:479–508, 1991.

[SSE87] 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.

[TC94] M. Teisseire and R. Cicchetti. An Algebraic Language for Event-Driven Modelling. In *Proceedings of the 5th International Conference on Database and Expert Systems Applications (DEXA'94)*, Lecture Notes in Computer Science, Athens, Greece, September 1994.

[Teo90] T. J. Teorey. *The Entity-Relationship Approach.* Morgan Kaufmann, 1990.

[TPC94] M. Teisseire, P. Poncelet, and R. Cicchetti. Towards Event-Driven Modelling for Database Design. In *Proceedings of the 20th International Conference on Very Large Databases (VLDB'94)*, Santiago, Chile, September 1994.

[Twi89] S. Twine. Mapping between a NIAM Conceptual Schema and KEE Frames. *Data & Knowledge Engineering*, 4(4):125–155, December 1989.

[TYF86] T.J. Teorey, D. Yang, and J.P. Fry. A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. *ACM Computing Surveys*, 18(2):197–222, June 1986.