Object-Oriented Analysis in Practice

J. Brunet, C. Cauvet, D. Meddahi, F. Semmak

Centre de Recherche en Informatique.
Université de Paris I - La Sorbonne
17, rue de Tolbiac 75013 PARIS, FRANCE

Abstract : The paper aims at proving that some object-oriented fundamental principles provide new suitable mechanisms for the analysis and the specification of complex systems. Three principles are presented and discussed in the paper through a case study. The locality principle allows to concentrate on one object, stressing its structure and behavior through the notion of life cycle, the refinement principle allows to refine objects by means of inheritance links, the globality principle allows to enlighten different kinds of dependencies amongst objects.
The case study is developped with the model of the Modway method. The Modway method is an object-oriented method which makes use of object concepts since the very beginning of the development process.

1 Introduction

During the last decade, the object-oriented paradigm has become progressively a main centre of interest in the field of information systems development. The basic concepts of object-oriented programming, object-oriented database systems and object-oriented design seem now to be relatively stable (e.g. [3], [15], [29]), although improvements should be done in order to take into account specific requirements of complex information systems. In contrast, up to now, there is no consensus on the very notions of object-orientation in information systems analysis and conceptual modeling.

Indeed, at least four families of methods are dealing with object-oriented analysis. The first one consists of methods originated from object-oriented design (e.g. [4], [8], [12]). Their purpose is to use software engineering concepts, such as the *use* relation, during the analysis process. These methods are often criticized because they don't take into account the behavior of the objects ([7], [18], [26]).
The purpose of the second family is to use functional decomposition, such as process and data flow, in order to provide a familiar way of working to the analysts already using a functional method (e.g. [1], [2], [20], [21], [24], [27]). Despite of the attempts to adjust data flows with the object-oriented paradigm, it is now recognized that these two approaches are inherently incompatible ([14], [25], [28]).
The third family of methods principally concentrates on data modeling (e.g. [13], [19], [23]). Objects are related with inheritance, association and aggregation constructors, and the objects behavior is specified by state transition diagrams. Unfortunately, these approaches have difficulties to handle the behavioral interactions between objects. In particular, the data flows diagrams technique used to represent the transformations of the system is not completely integrated; it does not allow to express all the dynamics of a system, and the possibility of visualizing together all the transformations applicable to an object is not supported.
Methods of the fourth family attempt to explicitly specify this dynamics by means of events or rules (e.g. [5], [9], [10], [11], [16], [17], [22]), following the assumption that the coherence of the future information system depends to a great extent on the explicit specification of the

interactions between objects. The expression of these interactions is made either by a procedural or a declarative approach.

The aim of the paper is to present some fundamental principles of object-orientation that we experimented on a case study. The three principles that we believe fundamental in an object-oriented analysis process are : locality, refinement and globality. The locality principle allows to concentrate on one object, stressing its structure and behavior through the notion of life cycle. The refinement principle allows to refine objects by means of inheritance links. The globality principle allows to enlighten different kinds of dependencies amongst objects.

The case study is developped with the object-oriented model of the Modway method[1] [6]. This method, which belongs to the fourth family of methods mentioned above, proposes to exploit the useful features of the object-oriented paradigm by defining different views with which the analyst can apprehend a real-world system and complete a specification.

The structure of the paper is the following. Section 2 gives a description of the principles and briefly presents the model supporting the Modway method. In section 3, we apply and discuss the three object-oriented principles on a case study.

2 Principles and Concepts for O.O. Analysis

This part presents three fundamental principles of O.O. analysis, and provides an overview of the Modway concepts supporting them.

2.1 Object-Oriented Analysis Principles

O.O. analysis widely makes use of the following principles, which purpose is to discipline the analysis process in order to achieve new qualities for conceptual schema such as clarity, integration, modularity, reusability, etc.

Locality

The locality principle consists of viewing each object in isolation from the other objects, as a unit being understandable as a whole. Locality is supported by two complementary concepts : abstraction and encapsulation. An object is an abstraction of a real world, which implies that it exists independently of other objects. Encapsulation ensures that an object description includes all structural and behavioral properties that caracterize it. These notions lead to produce object classes having a precise meaning, through the description of object local structure and behavior. Therefore, the analysis and modeling process will gain from elaborating, verifying and completing a specification in a local perspective. The locality principle originates from the abstract data types theory underlying to the object-oriented programming languages. However an important aspect of this principle, concerning local constraints on objects evolution, are often neglected by these languages.

Refinement

The refinement principle allows to consider an object in different roles, by defining inheritance hierarchies on classes. Many real world phenomena can play several roles at the same time. In

[1] This research is supported by the French Association for Research Development (ANVAR) under the MODWAY project. The MODWAY project is a collaborative project between University of Paris I and ALCATEL ISR compagny.

this way, we must be able to represent a single phenomenon of the real world through several classes, each class providing a particular view of the phenomenon. Unlike inheritance as defined in object-oriented programming languages, refinement allows an object to belong to more than one class.

Globality

The globality principle consists in viewing a system as a set of objects interacting together. It states that dependencies between objects must be explicitly specified in order to insure the system consistency in its totality. The application of this principle needs powerful mechanisms to conceptually express behavioral dependencies between objects. It is the author's belief that the message passing mechanism used in object-oriented environments leads to hide these dependencies in the method bodies, therefore it is not convenient to the analysis task which purpose is to explicitly represent all the complexity of a system.

2.2 The Concepts of the Modway Method

2.2.1 The Object Concept

The object concept is considered as a modeling concept which can be used to represent any kind of relevant element of the domain space.
Object description is based upon *attributes* and *events* which respectively characterize objects in space and time. The generic term property is used to refer to attributes and events.
Description of an object O consists of four parts:

$$O = (\mathrm{Id}, S, ST, \Gamma)$$

where Id, S, ST, Γ respectively refer to the object identifier, the object state, the object structure and the object life cycle.

The object state is the set of attribute values as well as the set of event occurrences. We consider that each object is able to store its state in its memory and is also able to react to events which occur during its life time.
The object structure is the set of relevant attributes which allow the object to have an existence in the domain space. Attribute values can be objects or data.
The object life cycle is the set of possible events which allow an object to evolve in time. An event occurs on an object when the object state satisfies a specific condition so-called *occurrence condition*. An event may be initiated in three different ways:

- by an actor of the environment (external event),
- by another event (internal event), in this case the event is inferred by another one,
- by Time (temporal event).

The life cycle of an object O is bounded by its birth event and its death event. Events occurrences in an object life cycle are totally ordered according to their occurrence time. The notation $ev(t_i)$ refers to the occurrence of the event ev at the time t_i.
The structure and the life cycle of an object makes its static and dynamic properties explicit and localized. These concepts fit well with the locality principle introduced above.

2.2.2 The Object Class Concept

An *object class* is a collection of objects having the same properties (attributes and events). Object classes are organized into a class hierarchy in which links between classes are "is_a" links. A class S "is_a" class G if and only if:

(i) Every object belonging to S also belongs to G,
(ii) properties defined in G are inherited in S.

It results from the definition that an object can belong to more than one class. An "is_a" link between two classes supports object refinement in the following way : an object o considered as an instance of a specialized class S is a refinement of o considered as an instance of a generalized class G.

Thus, an entity of the real-world may be modeled by an object which can belong to several classes. For example, a student might be represented by an object in the classes PERSON and STUDENT. The object viewed as an instance of STUDENT can be thought of as a role played by the PERSON.

An "is_a" link between two classes allows *refinement* of properties according to the augmentation mechanism.
Refinement of objects uses an augmentation mechanism (in opposition to the overriding mechanism) : properties defined in a specialized class are additional to the properties defined in the generalized class that is, they do not replace them.
In the following, the term of object will often be used instead of the term of class, because the first results of the analysis of the real-world is the identification of objects, not classes. However defining an object comes to define its class.

2.2.3 Object Dependencies

The structure and the life cycle of one object depend of other objects. An object may contain another one and an object may evolve by sharing events with other objects. There exists two types of dependencies between objects: structural and behavioral dependencies.

Structural dependencies

Two kinds of structural dependencies are defined : the *composition* link and the *refering* link [5].

(i) Formally, there is a composition link from an object O_1 to an object O_2 (we say that O_2 is a component of the composite object O_1) if and only if :

- O_1 has an attribute with O_2 as value,
- each event within the life cycle of O_2 is inferred by an event of O_1.

It results from this definition that a component object cannot be shared and its existence is entirely dependent of the composite one.

(ii) Formally, there is a refering link from an object O_1 to an object O_2 (we say that O_1 refers to O_2) if and only if:

- O_1 has an attribute with O_2 as value,
- birth $(O_1) \geq$ birth(O_2) and death$(O_1) \leq$ death (O_2)

As a consequence, a refered object can be shared and events may occur independently on both objects.

Behavioral dependencies

There exists two types of behavioral dependencies: the *synchronization* link and the *chronological* link.

(i) Events can be shared by two or several objects. A shared event may be viewed as several events which occur upon different objects with the same occurrence time. It defines a synchronization point in object life cycles. A shared event usually refers in its occurrence condition to the state of all the objects it is defined on, and it changes the state of all these objects.
Formally an event ev is shared by O_1 and O_2 if and only if:

- $ev \in \Gamma(O_1)$ and $ev \in \Gamma(O_2)$,
- For each $ev(t_i)$, $ev(t_i) \in S(O_1)$ and $ev(t_i) \in S(O_2)$

(ii) Events on objects may infer events on others objects. The inference mechanism allows to describe chronological dependencies between events defined in object life cycles.
Formally, an event ev on an object O_1 infers an event ev' on an object O_2 if and only if:

- $ev \in \Gamma(O_1)$ and $ev' \in \Gamma(O_2)$,
- For each $ev'(t_j)$, it exists $ev(t_i)$ / $ev(t_i) < ev'(t_j)$

Remarks :
- an event ev' may be infered by several events ev_0, ev_1, ... In this case, $ev'(t_j)$ can occur if and only if either $ev_0(t_j)$ or $ev_1(t_j)$ ... has occurred,
- ev and ev' may belong to the same object.

These four kinds of dependencies between objects support the globality principle according to which we can consider a system as a collection of objects which are involved in structural dependencies and which interact by event sharing and event infering.

2.2.4 Modway Graphical Notations

The Modway model provides diagrammatic tools to graphically represent the inner structure and behavior of an object, on the one hand, and its structural and behavioral dependencies with other objects, on the other hand. They are summarized in figure 1.

2.2.4.1 Local Representations of Objects

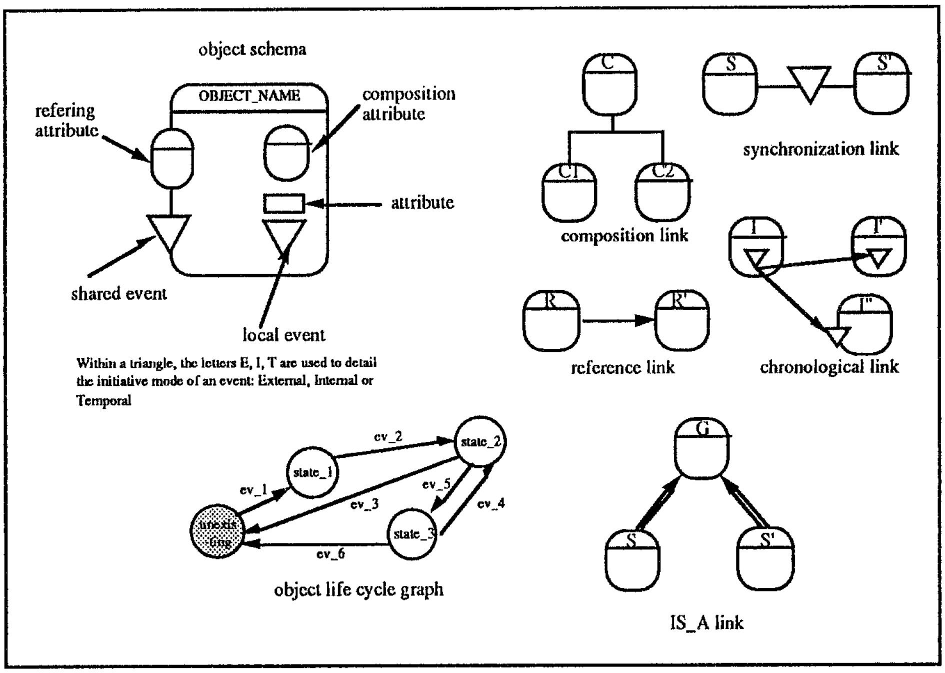
There are four local representations which encourage the description of objects in isolation from others:

- the *object class* enlightens object properties, that is attributes and events,
- the *composition graph* provides the components of the object (at all levels),
- the *life cycle graph* defines constraints on the object life cycle. A life cycle graph is a state diagram in which nodes correspond to object states and edges correspond to events. At a given time, an object can be in one and only one state of a life cycle graph.
- the *inference graph* vizualizes for each event of an object its inferences on other objects.

2.2.4.2 Global Representations of Objects

There is two global representations which aim at enlightening the designer on the dependencies between objects:

- the *dependency graph* consists of synchronization and reference links between objects,
- the *inheritance graph* consists of "is_a" hierarchies amongst object classes.



Within a triangle, the letters E, I, T are used to detail the initiative mode of an event: External, Internal or Temporal

Fig. 1 Graphical notations

3 Analysis of Case Study

At first, this part presents the functionalities of a system of Air Traffic Control. Then the three principles defined above are applied and discussed on the case study .

3.1 Description of the Case Study

The case study deals with an Air Traffic Control System (ATCS). The system offers the following functionalities :
- it allows to visualize a geographical control zone, where several aircraft traces evolve in time (a trace is the radar echo of an aircraft),
- it allows to correlate a trace with a flight plan,
- it allows to anticipate the trajectory of some aircraft traces :
  (i) check the correctness of the trace according to its flight plan,
  (ii) extrapolate the trace position according to its successive positions and its flight plan.

Traces for which the controller has requested an anticipation are called anticipated traces.

- it allows to evaluate possible conflicts for a given trace : a conflict occurs when there is a risk of collision for this trace.

Traces for which the controller has requested a risk evaluation are called protected traces.

3.2 Applying the Locality Principle on ATCS

In the Modway approach, the local description of objects is based on attributes and events. Life cycle graphs allow to state constraints on the possible occurrence of events, and composition links allow to define complex objects. These modeling concepts encourage the analyst in designing objects in isolation from others.

Events and attributes

Attributes allow to describe static properties of an object while events are concerned with dynamic properties. The set of events that may occur on an object provides a complete view of what can affect it, while the set of attributes defines its characteristics in the domain space.
Figure 2 shows the description of the object TRACE resulting from a local analysis.

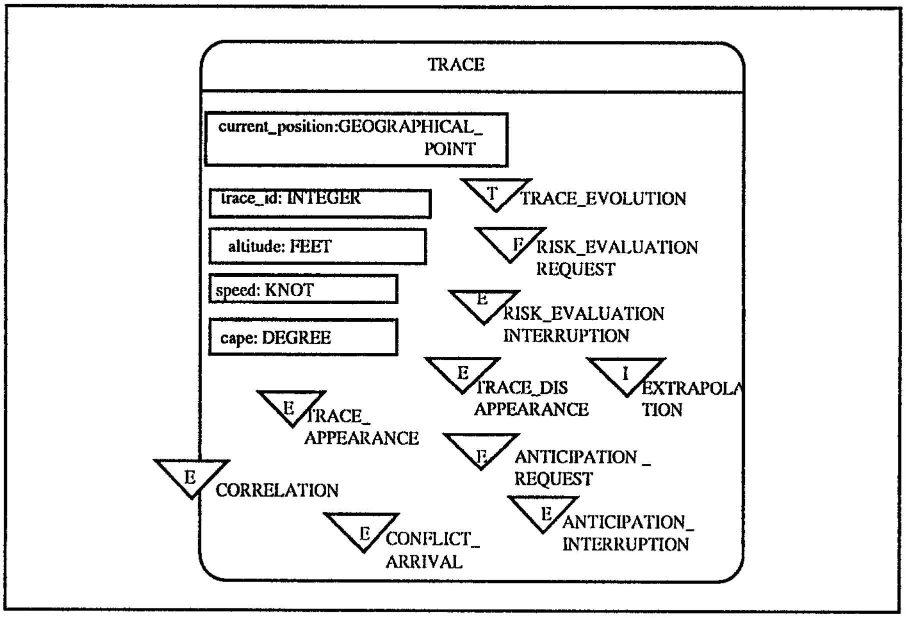


Fig. 2 Schema of a trace object

The local description consists of :

- several attributes characterizing a trace in the airspace,
- several events characterizing its life cycle :
  - a birth event TRACE_APPEARANCE corresponding to the appearance of a plane trace in the control zone,
  - an update event TRACE_EVOLUTION upon attributes values (new values are sent by the radar),

- some events representing controller requests for the anticipation of a trajectory, for the evaluation of conflict risks with other aircraft traces and for the interruption of anticipation and evaluation,
- an event CORRELATION which correlates a trace with its presumed flight plan,
- an event EXTRAPOLATION which extrapolates a trace trajectory,
- an event CONFLICT_ARRIVAL which occurs when a conflict has been detected for this trace,
- a death event TRACE_DISAPPEARANCE corresponding to the trace disappearance from the control zone.

The local analysis of an object includes the investigation of its events, in particular the reason of their occurrence (External, Internal or Temporal) and their occurrence condition. This investigation leads to locally detect infered and shared events.

For instance,
- the event EXTRAPOLATION is internal and it is infered by the event ANTICIPATION_REQUEST, because the extrapolation of a trace trajectory results from an anticipation request on this trace by the controller,
- the event CORRELATION is external and it is shared with an object FLIGHT_PLAN, because a correlation on a trace makes sense only with a flight plan.

In summary, this local description provides a general view of what is a trace and how it can behave. Merging together statics and dynamics in the same description leads to enhance intelligibility.

Life cycles

Having identified all the events applicable to an object, we may specify its life cycle. An object life cycle is a local view that sets some constraints on the possible occurrences of its events. For instance, the life cycle graph of an aircraft trace is presented figure 3.

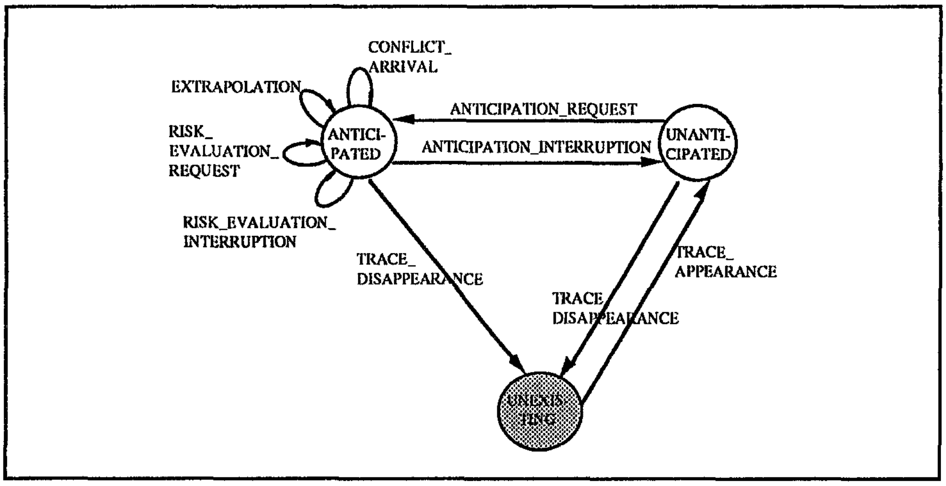


Fig.3 Life cycle graph of a trace

The states UNEXISTING, ANTICIPATED and UNANTICIPATED characterize different parts of the life of a trace, for which only a subset of the events defined on the class may occur. An edge relating states represents a transition from a state to another resulting from the arrival of the corresponding event.

For instance, the event TRACE_APPEARANCE can only appear if the trace does not yet exist, and sets the trace in the state UNANTICIPATED, on which the event ANTICIPATION_REQUEST can occur.
Some events, like the event TRACE_EVOLUTION, may occur for each state of the graph (unless the particular state UNEXISTING). For sake of clarity, this kind of event is not represented in the graph.

A life cycle graph states the possible sequences of events, considering a single object. The global study of relations between events of several objects will complete the behavioral description of the object.

Composition links

The composition link allows to capture a strong structural relationship between objects. Within a local perspective, a composite object encapsulates the component objects linked with it, i.e. the composite object and its component objects are seen as a whole.
In the ATCS case for instance, this link fits well to the relationship between a protected trace and its related conflict risks, because the existence of a conflict risk is totally dependent of a protected trace (let's remember that a protected trace is a trace for which the controller has requested a risk evaluation, and a conflict risk represents a risk of conflict between the current protected trace and another trace). Figure 4 describes the resulting composite and component objects according to the definition of the composition link :

- a protected trace object has a multi-valued attribute 'conflict_risks' which has conflict risks objects as values,

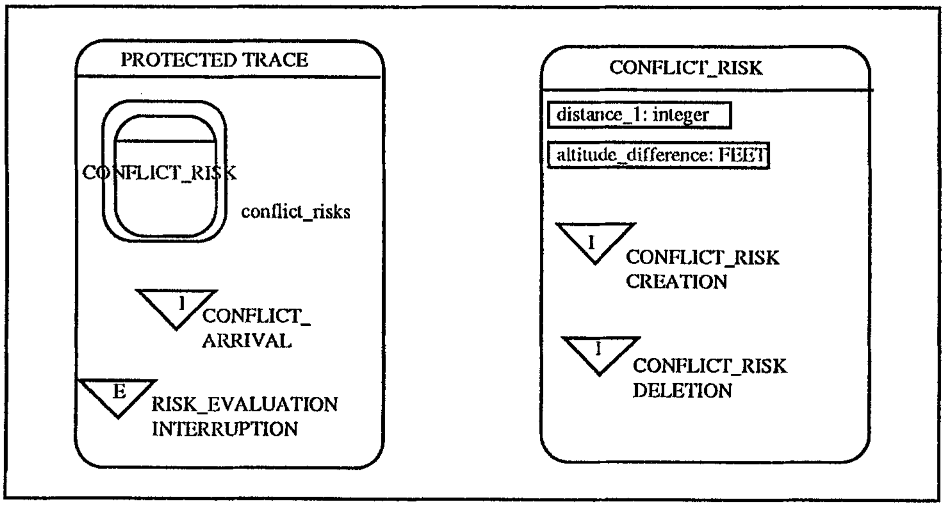


Fig.4 Composition link between a protected trace and a conflict risk

- each event of a conflict risk is infered by an event of its protected trace object; a CONFLICT_RISK_CREATION event on a conflict risk is infered by a CONFLICT_ARRIVAL event on a protected trace, and a CONFLICT_RISK_ DELETION event on a conflict risk is infered by a RISK_EVALUATION_ INTERRUPTION event on a protected trace.

In a local perspective, we concentrate on the object PROTECTED_TRACE, the object CONFLICT_RISK being visible through its attribute 'conflict_risks'.

Remark that the two events defined on the object CONFLICT_RISK are internal : they are infered and not accessible outside the object PROTECTED_ TRACE.

3.3. Applying the Refinement Principle on ATCS

The refinement principle allows to refine objects by means of the inheritance mechanism : an object of a class may be refined in a specialized class. Object refinement implies object properties refinement. We illustrate how object refinement implies event refinement and how object life cycles may be used in refining events.

For instance, in figure 5 are represented two specialized classes ANTICIPATED_TRACE and UNANTICIPATED_TRACE which inherit from TRACE. Each one corresponds respectively to the states ANTICIPATED and UNANTICIPATED of the trace life cycle graph (figure 3).

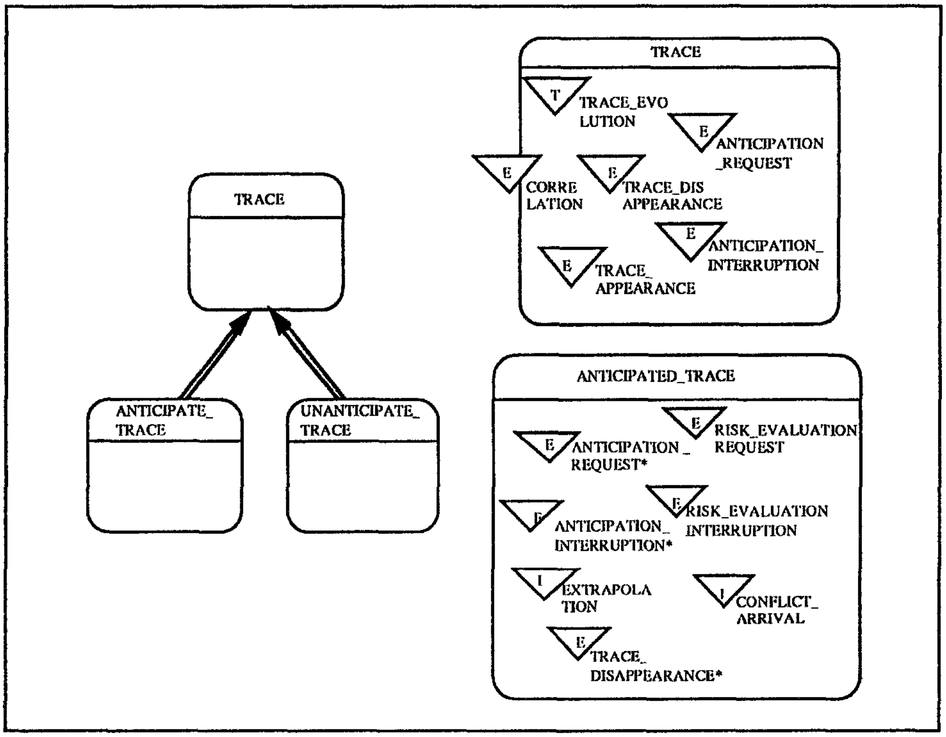


Fig. 5 Refinement of a trace object

The description of an anticipated trace will include specific attributes and specific events. These specific events are obtained from two complementary ways.
At first, all events of an object TRACE that apply only when the trace is in the state ANTICIPATED are carried down on the object ANTICIPATED_TRACE. This is the case of the events RISK_EVALUATION_REQUEST, RISK_ EVALUATION_INTERRUPTION, EXTRAPOLATION and CONFLICT_ ARRIVAL. This leads to increase the modularity because these events are only relevant for anticipated traces.
Secondly, some events of a trace are refined in an anticipated trace (and also in an unanticipated trace). Event refinement applies to each event in the life cycle graph of a trace having as origin or destination (but not the both) the state ANTICIPATED. This is the case of the events ANTICIPATION_REQUEST, ANTICIPATION_INTERRUPTION and TRACE_ DISAPPEARANCE, which appear in the description of an anticipated trace with a star attached to the event name (figure 5). For instance the event ANTICIPATION_REQUEST, considered on an object TRACE, changes the state of a trace from UNANTICIPATED to ANTICIPATED, while its refinement allows to adds the role "anticipated" to this trace.

In order to complete the specification of an anticipated trace, its life cycle graph can be defined (fig.6). This life cycle graph can be seen as a refinement of the state ANTICIPATED appearing in the trace life cycle graph.

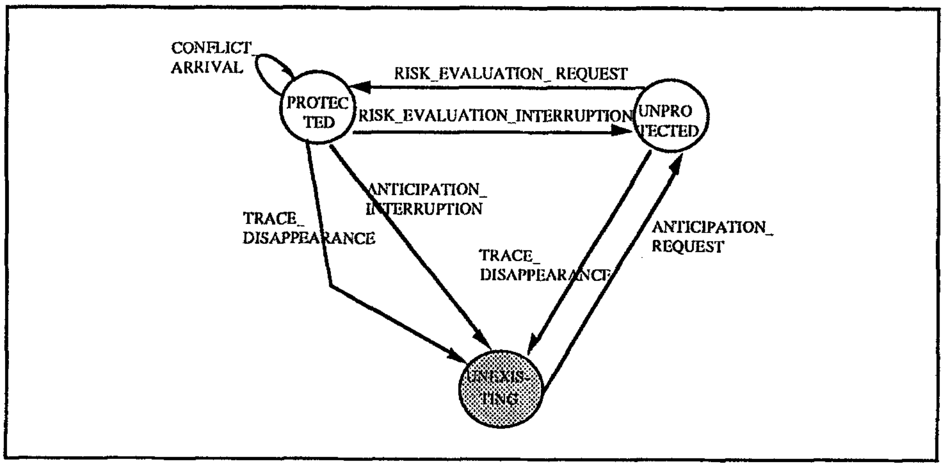


Fig. 6 Life cycle graph of an anticipated trace object

Like a trace, an anticipated trace can be refined, as a protected trace or an unprotected trace corresponding respectively to the states PROTECTED and UNPROTECTED (figure 7). It follows that the event CONFLICT_ARRIVAL is carried down from an anticipated trace to a protected trace, and the other events of the protected trace are refined.

In conclusion it seems that the refinement mechanism is suitable for the analysis of a phenomenon in different contexts. Each particular role of an object is modelized as a specialized object which can then be studied locally. Note that refinement is more general than inheritance as currently used in object-oriented design, because it allows to specialize an object in accordance with their states.

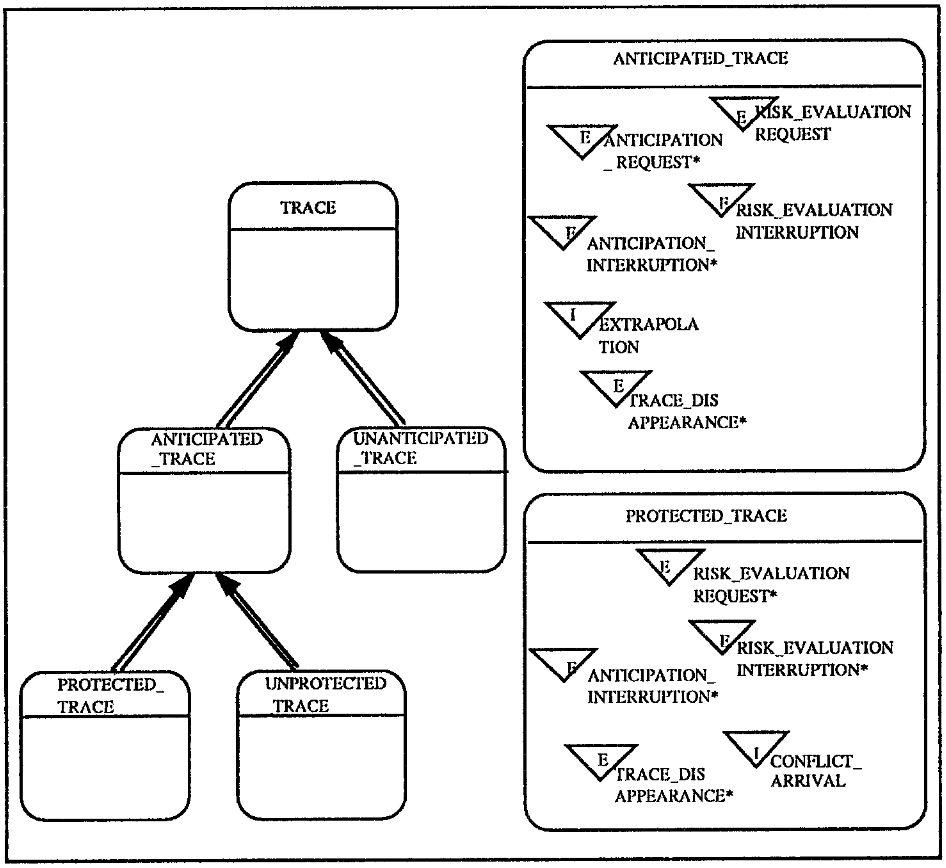


Fig. 7 Refinement of an anticiped trace object

3.4 Applying the Globality Principle on ATCS

The globality principle aims at emphasizing behavioral dependencies between objects. They are specified through synchronization and chronological links relating objects.

Synchronisation links

Contrarily to the local view in which objects evolve concurrently, in a global view, objects interact the ones with the others. Synchronization supported by shared events is a way to specify interactions between objects.
A shared event is a synchronisation point on the life cycles of two or several objects.

For instance in figure 8, the event CORRELATION is shared by the objects TRACE and FLIGHT_PLAN : a correlation affects simultaneously the state of the trace and the state of the related flight plan. By considering the two objects, it is possible to complete the specification of the event occurrence condition : the event CORRELATION occurs only if the flight plan is not already related to a trace and if the trace position is consistent with the flight plan.

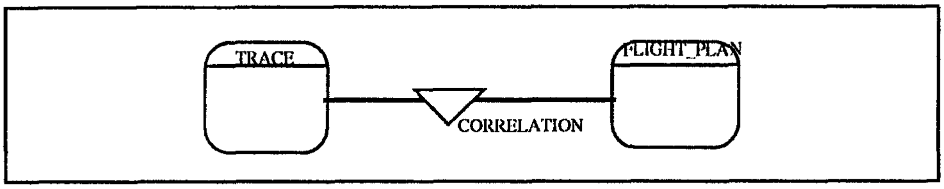


Fig. 8 The correlation shared event

In a local perspective, a shared event is placed on the border of the object class (see for instance figure 2). In a global perspective, it is represented by a link between the two objects.

Synchronizational aspects are commonly encountered during an object-oriented analysis process. They are important elements for the preservation of the system coherence. In the context of a banking application, the simultaneity of credits and debits is an essential point. The non-recognition of these aspects can lead to serious errors during the object-oriented design process.

Chronological links

Contrarily to the local view in which a composite object hides the detail of its components, in a global view the composite and the component objects are considered separatly. Their behavioral dependencies are expressed by an infering mechanism between events from the composite object to the component objects.
For instance in figure 9, the event CONFLICT_RISK_CREATION on the object CONFLICT_RISK is infered by the event CONFLICT_ARRIVAL on the object PROTECTED_TRACE. In other words, an event CONFLICT_ARRIVAL is followed by an event CONFLICT_RISK_CREATION, and an event CONFLICT_ RISK_CREATION can only occur after the occurrence of an event CONFLICT_ARRIVAL. In this case, event infering is a way of specifying propagation of events between a composite and component objects.

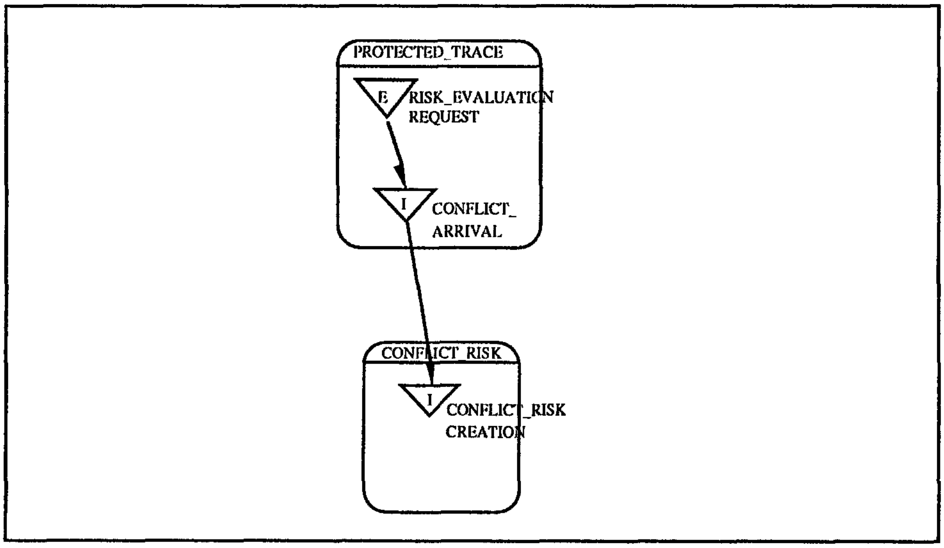


Fig.9 Two event inferences

Chronological links can relate events of the same object, in order to express other local chronological dependencies between events. For instance, in the same figure, a chronological link relates two events of the object PROTECTED_TRACE.

Chronological links allow to represent inferences between the evolution of several objects. This notion is different from those of message passing and client-server relation from object-oriented design, because it states the constraint that an infered event can occur only if the infering event has previously occurred.

In summary, the globality principle has to be applied for describing behavioral dependencies between objects, because of the crucial impact they have on the coherence of an objet-oriented specification.

4 Conclusion

At the present time, the principles of locality, refinement and globality are influencing more and more the task of system analysis. In this paper, we tried to demonstrate that these principles are necessary to discipline the analysis process in order to increase the quality of an object-oriented specification.

A first advantage is that the use of different points of view allows to enrich the knowledge an analyst may have upon a system. Some properties of the system may be apprehended locally in the context of an object, or class. Refinement allows to study in an isolated manner each facet of an object, even if several facets can exist at the same time. Finally, the global perspective allows to take into consideration the emergent properties of the system, resulting from the incrrelations and the interactions between the objects.

A second advantage is that the interdependence between these points of view allows to increase the flexibility of the analysis process : a given characteristics may be identified and partially described in a particular view, afterwards its description may be completed or verified by changing of perspective. For instance, an event identified locally on a single class may then be considered in the global perspective.

Using these principles in practice makes necessary the extension of the usual object-oriented concepts : structural and behavioral links between objects, roles of objects, events, etc. Moreover, an analysis process should integrate these three principles in order to provide new perspectives in the analysis of a system : local, contextual and global perspectives, instead of classical static and dynamic perspectives.
The avantages of these principles will be substantial when they will be supported both by a model and by the afferent methodological process.

REFERENCES

[1] B. Alabiso, "Transformation of Data Flow Analysis Models to Object Oriented Design", OOPSLA, San Diego, California, Sept. 1988

[2] S.C. Bailin, "An Object-Oriented Requirements Specification Method", Communications of the ACM, May 1989

[3] F. Bancilhon, "Object-Oriented Database Systems", 7th Symposium on Principles of Database Systems, Austin, March 1988

[4] G. Booch, *Object Oriented Design With Applications*, Benjamin Cumming Ed., 1991

[5] J. Brunet, "Modeling the World with Semantic Objects", in the Proceedings of the IFIP WG8.1 Working Conference on the Object-Oriented Approach in Information Systems, Quebec, Canada, octobre 1991

[6] C. Cauvet, C. Rolland, "An Event-Driven Approach to the Dynamic Modelling of Objects", 3rd International Working Conference on Dynamic Modelling of Information Systems, Delft, 1992

[7] C. Chee, C. Ng, M. Sim, "TOAD : Towards an Object-Oriented Analysis & Design Methodology, Experiences & Preliminary Observations", 3rd Int. Workshop on Software Engineering & its Applications, Toulouse, France, Dec. 1990

[8] P. Coad, E. Yourdon, *Object-Oriented Analysis*, Second Edition, Yourdon Press, 1990

[9] U. Dayal, A.P. Buchmann, D.R. McCarthy, "Rules Are Objects Too: A Knowledge Model For An Active, Object-Oriented Database System", 2nd Int. Workshop on Object-Oriented Database Systems, Springer-Verlag, Sept. 1988

[10] L.J.B. Essink, W.J. Erhart, "Object Modelling and System Dynamics in the Conceptualization Stages of Information Systems Development", IFIP TC8/WG8.1 Working Conference on the Object-oriented Approach in Information Systems, Quebec, Canada, Oct. 1991

[11] M. Fowler, "The Use of Object-Oriented Analysis in Medical Informatics for Large Integrated Systems", TOOLS 4, Prentice Hall, Paris, 1991

[12] J.G.M. van den Goor, "A Practical Approach to Object-Oriented Software Engineering", TOOLS, Paris, 1990

[13] B. Henderson-Sellers, "Analysis and Design, Methodologies and Notation", Tutorial, TOOLS, Paris, 1991

[14] P.H. Loy, "A Comparison of Object-Oriented and Structured Development Methods", Pacific Northwest Software Quality Conference, 1989 / reprinted in System and Software Requirements Engineering, IEEE Computer Society Press, Washington, DC, 1990

[15] B. Meyer, *Object-Oriented Software Construction*, Prentice Hall, Hemel Hemstead, 1988 / *Conception et programmation par objets*, InterEditions, 1990

[16] J. Morejon, R. Oudrhiri, "Le modèle EA2 : Entité - Association / Evénement - Action", Congrès Autour et à l'entour de Merise, Sophia Antipolis, April 1991

[17] B. Pernici, "Objects with Roles", ACM/IEEE Conference on Office Information Systems, Boston, MA, April 1990

[18] B. Pernici, "Requirements Specifications for Object-Oriented Systems", Nouvelles perspectives des Systèmes d'Information, INFORSID, Biarritz, France, May 1990

[19] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, 1991

[20] U. Schiel, "OKAY- Object-Oriented Knowledge Analysis and design", COMAD, Inde, Dec. 1989

[21] E. Seidewitz, M. Stark, "Towards a General Object-Oriented Software Development Methodology", SIGAda Ada Letters, July 1987

[22] A. Sernadas, J. Fiadero, C. Sernadas, H.D. Ehrich, "The Basic Building Block of Information Systems, Information Systems Concept", North Holland, Namur, 1989

[23] S. Shlaer, S. J. Mellor, *Object Lifecycles : Modeling the World in States*, Prentice Hall, 1991

[24] P.D. Sully, "Essentially Objects", TOOLS'90, Paris

[25] A.G. Sutcliffe, "Object-oriented systems development : survey of structured methods", Journal of Information and Software Technology, vol. 33, n°7, July 1991

[26] M. Teisseire, P. Poncelet, A. Cavarero, S. Miranda, "A-HOOK, The object-oriented analysis of the HOOK system", report of External European Research Project, 1991

[27] P.T. Ward, "How to Integrate Object Orientation with Structured Analysis and Design", IEEE Software, March 1989

[28] R.J. Wieringa, "Object-Oriented Analysis, Structured Analysis, and Jackson System Development", Int. Conference on the Object-Oriented Approach in Information Systems, Quebec, Canada, Oct. 1991

[29] R. Wirfs-Brock, B. Wilkerson, L. Wiener, *Designing Object-Oriented Software*, Prentice-Hall, Englewood Cliffs, N.J., 1990