Object Oriented Specification based on Restrictions: Participation and Interaction

J. TORRES, J.A. TROYANO, M. TORO, R. CORCHUELO, A. DURÁN
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Sevilla
Avd. Reina Mercedes s/n 41012 Sevilla
SPAIN

Abstract: - We show in this paper an object oriented model (and an associate language called TESORO) based on several kinds of restrictions. We describe the template of an object class using four types of restrictions: a) static restrictions that allow us to express a state invariant, b) transition restrictions that describe the relationship between the previous and following states of an event occurrence, c) participation restrictions that specify the possible ways of evolution, and d) interaction restrictions that describe the way in which objects synchronise and communicate among them. Another important characteristic of our model is the differentiation between individual and collective features of objects. In this sense when we specify the template of a class, we can describe individual restrictions that have an effect only on an object, or we can describe collective restrictions that have an effect on a set of objects or even on a whole class of objects.

*Key-Words:*Restriction,participation,interaction,communication of objects. *CSCC'99 Proceedings,* Pages:6441-6445

1 Introduction

Object oriented development is a popular approach in Software Engineering, it has been successfully used in all the phases of software development from analysis to implementation.

Particularly in the analysis phase, in recent years have appeared several specification languages based on object oriented concepts, like OOZE[1], Z++[9], TROLL[6], LCM[4], OASIS [10], etc.

In this paper we do not aim to present another object oriented specification language. On the contrary we are going to present an object model that combines object oriented features and restrictions.

Objects are the principal pieces in our model. An object has a structure defined by means of attributes, and a behaviour described by means of events. Classes are sets of objects with the same structure and behaviour. Actually, we do not specify objects, we specify the features of classes of objects, describing templates of classes. Once we have the description of a class, we can define an object as an instance of this class.

A whole system is a set of objects that can be created and destroyed. We assume that all the objects that exist in every single moment evolve concurrently. However, the internal behaviour of an object is sequential [3].

Events are the points that determine the evolution of every object and of the whole system. Objects can participate in events, and this participation can provoke a change in their states. In the same event can participate one or more objects, this aspect provides synchronisation and communication among objects.

Our communication model among objects is based on *multi-part synchronous interactions*[5]. With this kind of interactions we can synchronise several objects with their participation in the same event. These objects will negotiate the values of the parameters of the event and this negotiation will imply the communication of values among objects. This approach is not new, and has been presented

in several proposals, for example: synchronisation between two processes with communication of value (CSP[7]), or rendezvous n-ary asymmetric (LOTOS[8]). These two proposals have in common that the number of parts in the communication is static, that is, we have to know how many objects (process in LOTOS and CSP) are going to participate in the same event. However, our communication model is more flexible, because we do not have to specify the number of objects that will participate in an event, this number is dynamic and it will be determined at the time of the event occurrence.

The organisation of this paper is that follows. This introduction is the first section. In the second section we are going to present a classification of the restrictions that we can use to specify the structure and behaviour of a class of objects. In third and fourth sections we present two of the most interesting characteristics of our language: participation restrictions and interaction restrictions. Finally, in the fifth section we extract the main conclusions of this work.

2 Types of restrictions

Our model allows us to specify a system using classes and objects. We can impose restrictions on the structure of the objects in a similar way that in Data Base specifications[2], and restrictions on the behaviour of the objects like in concurrent systems[7],[8].

Depending on the way in which restrictions limit the structure and behaviour of objects, we can establish the next classification:

a) Static restrictions: With this kind of restrictions we can limit the set of possible states in which objects of a class can be. According to the scope of application, we have three kinds of static restrictions:
   - *Individual static restrictions:* These restrictions are defined in the template of a class and establish relationships among the values of the attributes of each object of this class. These restrictions are evaluated individually.
   - *Collective static restrictions*: These restrictions limit the state of the whole class of objects (no only the state of an individual object). We define collective static restrictions by means of predicates evaluated over multi-sets of attributes (all values that takes an attribute for each object of the class). For example, we can express that the sum of the values that takes an attribute for each object of a class do not go beyond a limit.
   - *Inter-class static restrictions:* With these restrictions we can describe structural relationships among classes of objects. We specify these restrictions by means of predicates over multi-sets of values of different classes. With a inter-class static restriction, for example, we can specify that the number of objects of a class must be greater than the number of objects of another class.

b) Transition restrictions: These kind of restrictions relate two states, present state and next state. Like static restrictions, we can establish transition restrictions at the object level (individual), at the class level (collective) and at the system level (inter-class).

c) Participation restrictions: With these restrictions we can specify when and how an object is able to participate in events. We can express participation restrictions using three mechanisms:
   - *Permissions*: That are logical expressions associated to an event. These expressions are evaluated over the values of the attributes of an object and over the parameters of the event. If the permission is true, the object will be able to participate in the event.
   - *Dynamic restrictions*: With this kind of restrictions we are able to describe an order relationship in the event participation of an object. We use process algebra operator to specify *behaviour expressions*.
   - *Obligations:* With permissions and dynamic restrictions we only can express possibility of participation, but we can not express certainty of participation. Now, with obligations, we specify conditions that if

they are true, assure us the participation of an object in a certain event.

The behaviour of an object can not violate none of the participation restrictions imposed to its class. But these are not the only restrictions that we have to safeguard, we also have to guarantee that static restrictions are not violated. Due to the participation in an event can provoke a change of state, its participation only will be possible if the change of state that it causes do not violate any static restriction. In this sense, we can say that static restrictions are implicit participation restrictions.

d) Interaction restrictions: These kind of restrictions describe which objects will participate in the same event and which roles play each one of them.

3 Participation restrictions

Through participation restrictions we will be able to express when an object is disposed to participate in an event. We can use three different mechanisms, *permissions, dynamic restrictions*}, and *obligations*.

3.1 Permissions

Permissions are logical expressions associated to events (actually to channels that are abstracts views or *types* of events). These expressions are evaluated over the values of the attributes of an object and over the parameters of a channel. If a permission is true, it will make possible the participation of the object in an event through the channel. The way in which this participation will affect to the state will depend on the enabled transition.

Permissions are restrictive mechanisms, because they limit the participation of an object in events depending on its state. For example, we permit the participation of an object of a class *Server* in an event of the channel *service(req_res:nat)*, only if the number of requested resources (the parameter *req_res*) is smaller than the number of available resources (the variable attribute *avl_res*):

req_res<=avl_res => service(req_res)

3.2 Dynamic restrictions

With permissions and transitions we can specify state machines to describe the behaviour of a class of objects. However, there is a more compact way to do this using dynamic restrictions. These mechanisms allow us to describe an order relationship in the event participation of an object.

Dynamic restrictions are specified by means of processes. A process has a name and its behaviour is described by a *behaviour expression*. In this type of expressions, events are the operands and they are combined using a set of operators with a predefined semantic [7],[8].

A behaviour expression has the following abstract syntax:

```
behav_exp  : event
           | behav_expr[]behav_expr
           | behav_expr|||behav_expr
           | behav_expr;behav_expr
           | behav_expr *
```

The operator '[ ]' denotes choice, the operator '|||' denotes interleaving, the operator ';' represents sequential composition, and '*' means iteration.

For example, we can specify the behaviour of a vending machine with the following behaviour expression:

*(coin; (coffee [] chocolate)) **

3.3 Obligations

Obligations are (like permissions) logical expressions associated to channels. With this mechanism we are able to describe an obligatory participation in a event. If an obligation (evaluated for a concrete object *o* and a concrete event *cn(v)*) is true, the object *o* is obliged to participate in the event *cn(v)*.

For example, given a class *Client* of a library, we can oblige to its objects to participate in the event

return_book when he has more than three books (attribute variable *number_books*):

number_books > 3 => return_book

4 Interaction restrictions

In our model, it is possible that more than one object participate in the same event. Interaction restrictions let us to specify how many objects (and how) can participate in an event.

Through interaction restrictions, we can say how many classes are involved in the same channel *c*, and how the objects of these classes decide the values of parameters of the event (through the channel *c*) in which they are going to participate.

Given a channel *c*, each class cl_i of objects involved in it has a local view of *c*, denoted c_i. All local views of a channel *c* have not to be equal, because from the point of view of each class, all parameters of a channel are not interesting. So, these *local views* are a syntactic mechanism that let us specify which aspects (parameters) of a channel are interesting for a class.

In fact, with interaction restrictions, we establish relationships among the different local views of a channel, composing the global (or system) view of the channel. To do this we only have to *link* the parameters of a local view to the equivalent parameters of other local views.

For example, given a library system, with two important classes, the *Book* class and the *Client* class. There is an action (event) in which object of both classes are going to coincide, this action is the *loan* of a book to a client. From the point of view of the class book we can assume that it is only important to know that somebody want to get it. So the local view of the channel may be:

be_lent()

However, from the point of view of the client, it is also important to say the book that he wants. So the local view of the channel may be:

request(b:Book)

the global view of the channel (named *loan*) will be determined by the following interaction restriction:

Client(c).request(b)= loan:Book(b).be_lent

where we are saying that the object *b* of the class *Book* can participate in a *loan* event at the same time that the object *c* of the class *Client*. Besides, we are saying that the identification of *b* must coincide with the parameter *b* of the *Client*'s local view of the channel.

In this example, the global channel will have the following structure:

loan(c:Client, b:Book,)

Actually, what we manage with an interaction restriction is to merge local participation restrictions of several classes (imposed to local views of a channel) into a unique participation restriction (imposed to the global view of the channel).

To allow the occurrence of an event through a channel that involves several classes there must be at least one object of each class disposed to participate in that event.

If there are more than one objects of a class disposed to participate in that event they will be able to do it. So, although the number of classes involved in an interaction restriction is always static, the number of objects that can participate in the same event is dynamic, and it will depend on the local participation restrictions of each object.

5 Conclusions

In this paper we have shown the main characteristics of an object oriented model based on several kinds of restrictions. We have only presented two kinds of restrictions: *participation restrictions*, and *interaction restrictions*, but the whole model includes also static and transition restrictions.

By means of static restrictions we can limit the set of possible states of an object, specifying logical predicates that can not be violated.

Through transition restrictions we establish relationships between two states, the present state (the state before the occurrence of an event) and the next state (the state after the occurrence of the event).

With the help of participation restrictions we will be able to express when an object is disposed to participate in an event.

Finally, through interaction restrictions we can describe the way in which objects synchronise and communicate among them.

The other important conclusion is the difference between individual and collective characteristics of a template. In our model there are two levels of specification, the individual level and the collective level. We can express properties at the individual level, involving only one object of the class, and properties at the collective level, involving a set of objects of a class or even the whole class of objects.

A more extensive definition of TESORO can be found in the PhD. Thesis [11] which presents the formal semantic of it, and [12] which studies inheritance aspects of TESORO.

References:

[1] A. Alencar and J.A. Goguen. OOZE: An Object Oriented Z Environment. *ECOOP'91 Proceedings*, vol. 512, pp. 180-199, 1991.

[2] S. K. Das. Deductive Databases and Logic Programming. Addison-Wesley, 1992.

[3] H.-D. Ehrich and A. Sernadas. Local Specification of Distributed Families of Sequential Objects. In *Recent Trends in Data Type Specification*, Proc. 10th Workshop on Specification of Abstract Data Types. Pp. 219-235. Springer, LNCS 906, 1995.

[4] R.B. Feenstra and R.J. Wieringa. LCM 3.0: a Language for Describing Conceptual Models. Technical Report IR-344, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, December 1993.

[5] N. Francez and I.R. Forman. Interacting Processes. A Multiparty Approach to Coordinated Distributed Programming. Addison-Wesley, 1996.

[6] T. Hartmann, J. Kusch, G. Saake and P. Hartel. Revised Version of the Conceptual Modeling and Design Language TROLL. Pro. ISCORE Workshop WS'94}, pp. 89-104, 1994.

[7] C.A.R. Hoare. Communicating Sequential Processes. Prentice-Hall International Series in Computer Science, 1985.

[8] ISO - Information Processing Systems Open Systems Interconnection - LOTOS, A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. ISO 8807, 1988.

[9] K. Lano. Z++, an Object-Oriented Extension to Z. In J. Nicholls editor, *Z User Meeting*. Workshops in Computing. Springer-Verlag, 1991.

[10] O. Pastor, F. Hayes, S. Bear. OASIS: An Object-Oriented Specification Language. *Proceeding of the II Conference CAiSE*, pp. 348-363. Manchester, UK. May 1992.

[11] J. Torres. Especificaciones Orientadas a Objetos basadas en Restricciones: prototipado en un lenguaje orientado a procesos. PhD. Thesis. Universidad de Sevilla. Diciembre de 1997.

[12] J.A. Troyano. Herencia y Clasificación en un Lenguaje de Especificación Orientado a Objetos. PhD. Thesis. Universidad de Sevilla. Junio de 1998.