CONSTRUCTS FOR PROTOTYPING INFORMATION SYSTEMS WITH OBJECT PETRI NETS

P. Sánchez, P. Letelier, I. Ramos
Department of Information Systems and Computation
Valencia University of Technology
Camino de Vera, s/n, 46071 Valencia - Spain
☎ 3877350; Fax: 3877357
email {ppalma | letelier | iramos} @dsic.upv.es

ABSTRACT

OASIS is a Language for the Specification of Object Oriented Conceptual Models. Object Petri Nets (OPNs) support a full integration of object-oriented concepts into Petri Nets. We propose a way to represent object-oriented concepts used in the OASIS language with OPNs as a suitable semantic model for validating software specifications. We have developed a Basic Execution Model for OASIS Specifications including its main features. Communication aspects between objects are taken into account in our proposal (triggering mechanism and shared events). We will consider event preconditions reducing the worlds to be reached, attribute valuations changing the state of objects, creation and deletion of objects, and life cycles of objects. OPNs are an appropriate semantic foundation over build a concurrent software engineering environment for distributed computation because it allows a natural representation of concurrence. We show how the object-oriented concepts of an OASIS Specification are represented into OPNs.

1. INTRODUCTION

OASIS [4][5] is a Formal language for the Conceptual Specification of Information Systems. Nowadays there is a growing interest in formal approaches to model information systems. CMSL [9] and TROLL [1] are languages similar to OASIS that address the system specification in such way. OO-Method [6] is the companion methodology for the OASIS approach.

Is important to animate the system specification by means of a prototype automatically generated. Experiments have been carried out using Petri Nets [7] and Concurrent Logic Programming [2] as semantics domains for OASIS Specifications. These efforts have led to the establishment of a basic Execution Model [3]. This model can be used in order to animate OASIS Specifications implemented over concurrent programming environments. The aim of this paper is to present the mapping between OASIS concepts and Object Petri Nets. OPNs support a complete integration of object-oriented concepts. Inheritance, polymorphism, dynamic binding and multiple levels of activity can also be included.

2. BASIC CONCEPTS OF OASIS

OASIS is a language for Open and Active System Information Specification. In the object model of OASIS, an ***object*** is an observable process encapsulating structure and behaviour. An object is an operational unit that has static and dynamic features. Each object has an internal and unique identifier (*Oid*) that allows identifying the object during its whole existence. Object behaviour is characterised by the actions received, those which the object is able to carry out, and by the actions sent, those which it is able to request. Actions are received and sent by interaction mechanisms between objects.

The object properties are represented by ***attributes***. Thus, the ***state*** of the object in a given moment is determined by the value of its attributes. An ***event*** is the abstraction of a change of state. The state can change after the occurrence of an event that modifies the values of some attributes. An event does not have duration and occurs in an instant of time.

Each object possesses a fixed set of events that could affect it during its existence. The state of the object in a given instant will depend on the events occurred in its life until the moment during which is observed. In this way, an object is seen as an observable process.

The attributes of the object are classified as follows: ***constant*** if its value does not change during the life of the object, ***variable*** if its value can change as a consequence of the occurrence of events. Furthermore, if the value of an attribute is obtained from the value of other attributes, the attribute is said ***derived***.

Two kinds of events can be distinguished: ***private*** and ***shared***. Private events participate in the life of objects of only one class, they appear only in the signature of one class or in several classes belonging to the same hierarchy of specialisation. Shared events are part of the object life of more than one class. These events appear in

0-7803-4053-1/97/$10.00 © 1997 IEEE

the signatures of all the classes that share them. ***Agents*** (also called ***clients***) activate the events and they have that responsibility. The event requested by an agent object to some ***server*** object(s) is called an ***action*** An event will only be relevant in the life of an object if the state of the object is suitable for that. Event ***preconditions*** are formulas and they should be satisfied in order to allow the occurrence of an event.

The objects are not isolated. In OASIS, the interaction between objects is modelled in two forms: by ***events sharing*** and by ***triggers***. Triggers introduce activity in the society of objects, allowing an object to act as an agent whenever some conditions are satisfied in its state.

The values of attributes should follow some rules. These rules are named ***integrity constraints***. They are defined by formulas in some kind of logic. They should be satisfied in the states in the life of the object (static constraints) or among certain sequences of states (dynamic constraints). A group of events that operate as a unit of execution of greater granularity is called a ***transaction***. They follow the two basic principles for transactions: all or nothing, and not observation of intermediate states. It is possible to construct ***processes*** using events and transactions as atomic actions, They allow specifying the correct sequences of actions in the life of the objects.

A ***template*** is a set of common properties that are shared by of several objects. A ***class*** is defined as the group of objects that share the same template. An individual object is called a class instance. The Society of Objects of our object-oriented system is built from primitive, elementary and complex classes. Objects that have only one state constitute the Primitive Classes or domains and always exist. In general, they undertake the domain of the abstract data types. The ***Elementary Classes*** are built from the primitive classes. There are attributes, events, transactions, preconditions, integrity constraints, triggers and process description defined and they characterize the objects of the class.

Complex Classes can be defined from elementary ones by applying operators between classes. The operators that allow the construction of complex classes are ***Aggregation***, ***Association***, ***Specialisation*** and ***Parallel Composition***. These operators are orthogonal, that is, they are independent and they can be combined.

OASIS is a formal specification language that allows one definition of conceptual schemas according to the object model that has been presented. Here is an example representing an account in a bank system.

```
class Account
identification
   by_id:(id).
constant_attributes
   id:nat;
   name:string.
variable_attributes
   balance:int;
   number:nat (0).
derived_attributes
   good_balance:bool.
private_events var N:nat.
   open new;
   close destroy;
   deposit(N);
   withdraw(N);
   pay_commission.
valuation
   [deposit(N)] balance=balance+N and number=number+1;
   [withdraw(N)] balance=balance-N and number=number+1;
   [self:pay_commission] balance=balance-100 and number=0.
derivation
   good_balance={balance>100000}.
preconditions
   withdraw(N) if balance>=N;
   close if balance=0.
triggers
   self::pay_commission if (number>10) and
                           (good_balance=false).
processes
   ACCOUNT =   open.ACCOUNT0;
   ACCOUNT0    =  close +
               deposit(N).ACCOUNT0 +
               withdraw(N).ACCOUNT0 +
               pay_commission.ACCOUNT0.
end_class
```

This concrete syntax is formalized in dynamic logic [8] and it has been used in [5].

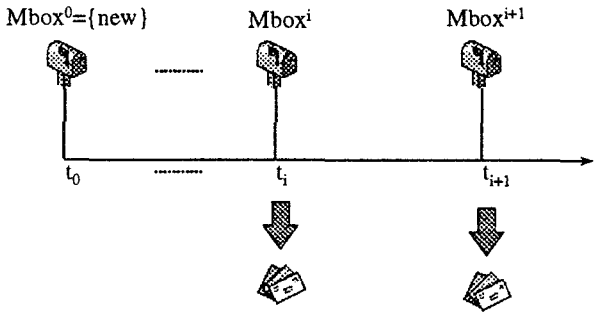

Figure 1: Object life cycle

Figure 1 shows life cycle of an object. It is shown according to the previous concepts. Each object has a mailbox. The change of state does not have duration and occurs at a precise instant of time, when the actions received in the mailbox are executed. To ***attend*** an action means to extract it from the *Mbox*. The attended actions can be ***admitted*** or ***rejected*** depending on they satisfy or not their preconditions. Hence, $Accept^i$ will be the set of admitted actions at the instant t_i , then $Accept^i \subseteq Mbox^i$.

Considering $Exec^i$ as the set of actions executed when the state is $State^i$, then $Exec^i \subseteq Accept^i$. Therefore, an

object will change from the state $State^i$ to the $State^{i+1}$ by the ***execution*** of the actions included in $Exec^i$. Intra-object concurrence is allowed, but the set of executed actions must not be in conflict [3]. Requested actions received by an object are called ***service actions***. For each action that the object must trigger, whenever it reaches the state $State^i$, it will exist another action in the mailbox in the instant t_i. Thus, the server perspective of an object is integrated with its client perspective in a homogeneous form of operation. These actions received by the object due to send triggers are called *trigger actions*. This means, the object must serve service actions as well as trigger actions.

Service actions in $Mbox^i$ are represented by $Mbox^i_{services}$.

Trigger actions in $Mbox^i$ are represented by $Mbox^i_{triggers}$.

Therefore, it is always true that:

$$Mbox^i = Mbox^i_{triggers} \cup Mbox^i_{services}$$

At each t_i, $Mbox^i_{triggers} \subseteq Exec^i$, because all trigger actions must be executed. Thus, $Exec^i$ can additionally contain subsets of accepted service actions. These actions are in conflict neither with one another nor with the trigger actions.

The contents of the mailbox must be updated in each t_i:

(a) Trigger actions or service actions at t_{i-1} have to be extracted.
(b) All the new actions received until t_i must be included.
(c) The actions accepted and not executed in previous states (because they were in conflict) must be stored again.

We have considered all these ideas in order to define a basic execution model including the main features of OASIS specifications. Hence, this model allows animating an OASIS specification accurately.

3. OBJECT PETRI NETS

The enhancements of OPNs [10], [13], [14], include allowing token values to be identifiers, inheritance, test and inhibitor arcs. Furthermore, OPNs provides functions for evaluating the state of the net without changing its state. The possible use of superplaces and supertransitions makes OPNs suitable to model both synchronous and asynchronous interaction between objects. Each instance of an OPN class could be an independent agent passing messages between OPN objects through tokens. The notion of OPN superplace permits synchronous interaction. One transition deposits in (or extracts from) a token of the superplace. This operation is synchronous with an internal transition of the superplace that accepts (or produces) the token. Both transitions can restrict the interaction by means of the adequate guard.

4. CONCEPTUAL SCHEMAS IN OPN

An OPN class at the highest level (meaning the *Conceptual Schema*) is designed to contain references to each OASIS class. Likewise, each OASIS class is represented by a OPN place. As shown in Figure 2, communication between classes is performed through a special *router*.

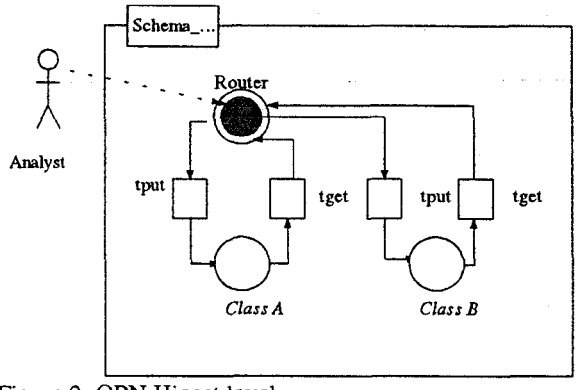

Figure 2: OPN Higest level

The analyst interacts with the running system. All incoming events are routed to the target class. Each event between classes is forwarded through the router, enhancing the facilities to broadcast operations and environment communication.

For example, *ClassA* (more accurately, some instance) sends an event to another class, say *ClassB*. The token extraction, by firing the *tget* transition, is synchronous with the token insertion in the router class. After that, the router class sends the same token (i.e. the event) to *ClassB*. Equally, the token admittance in *ClassB* will be synchronous with the token sent by the router.

5. OASIS CLASSES IN OPN

We will distinguish the OASIS class (which contains the template, the identification of instances, etc.) from the instances that belong to that class. Henceforth, *class* and *instances* will designate OASIS class and OASIS class instances, respectively. For each OASIS class (e.g. *ClassA*), an OPN class (e.g. *ClassA*) and another one to designate the instances (e.g. *InstancesA*) will be defined.

Example 1: The class *Accounts* in our bank system implies two OPN classes: (1) *Class_Account*, and (2) *Instances_Account*.

A general representation for each class is shown in Figure 3. Few details have been shown in order to have a more readable diagram and to emphasize the main interactions.

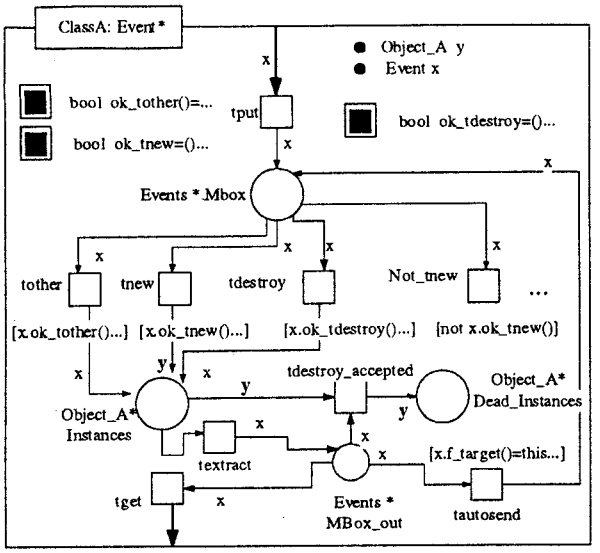

Figure 3: Internal representation of classes.

Notes:

(a) Following notation proposed by Lakos [13], a function is drawn with a double black square, a single transition with a square and a single place with a circle. Arrows reaching the class limits represent synchronous communication. The token type that can be received at the class level (or offered) is named after the class name (*Event* in our example).

(b) The transitions *tput* and *tget* (which redefine the original transitions *put* and *get* for capturing and offering tokens at the superplace, respectively) give the synchronous communication between the router and the classes [12]. Received events are put in the place *Mbox*. The related transition will be fired and questions like identification mechanism, event existence, etc., will be checked. Wherever required, functions (denoted with double black squares) will give information about the state of the object.

(c) The place *Instances* contains every class object. Each event received at the class level is routed to the instance level in a synchronous way.

(d) Each generated event from instances is put in the place *Mbox_out*. If the target is another class, then the event is routed through the *tget* transition. On the contrary, if the target is an object belonging to the same class, then it is forwarded once more to the place *Mbox* (see the transition *tautosend*).

Object Creation

If an object sends an event *new*, two possibilities arise: (1) the preconditions are satisfied and the event is admitted (see *ok_tnew()*), and (2) the event is rejected. The former provides the creation of objects at *Instances* place (note the token type of variable *y* added to the place *Instances* when transition *tnew* is fired). All active objects are tokens of the place *Instances*. Each object has an internal activity [13] that will depend on the OASIS specification. It is important to highlight that preconditions associated to the event *new* are checked by the class instead of the object instance, which is has not been created yet.

Example 2: The creation event *open* of the class *Account* will be received by the OPN class *Class_Account* (through *tput* firing). Then, the transition *tnew* is fired (if *ok_tnew()* is satisfied) and a new object *Account* is added.

Object Destruction

If the class receives an event to remove an object then this event is routed to the correspondent instance. Afterwards, if the event is accepted, the instance replies with another one, confirming the removing action. If the state of the object permits it, then (see *tdestroy_accepted*) it will be moved from the place *Instances* to the place *Dead_Instances*, otherwise it will remain alive.

Example 3: If the event *close* occurs the transition *tclose* will be fired (whether *ok_tclose()* is satisfied), and it will be sent to the corresponding instance of *Account*.

6. OASIS OBJECTS IN OPN

A superplace is also defined for class instances in the same way as it was previously done. The representation of instances is shown in Figure 4:

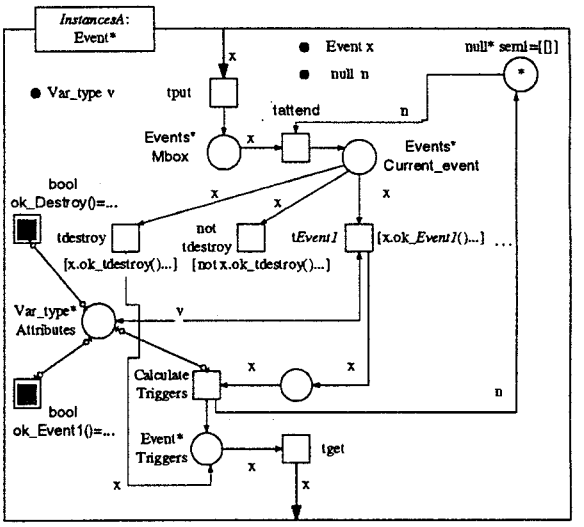

Figure 4: Representation of instances.

Notes:

(a) The events sent from the class level (of which the current object is an instance) are accepted through the transition *tput*. Similarly, events sent from the instance to the class are routed through the

transition *tget*.

(b) The place *Attributes* contains pairs (*attribute, value*) and represents (partially) the state of objects.
(c) The place *Current_Event* contains the selected current event to process next.
(d) The place *sem1* permits to catch only one event each time and to wait for the following state.

Event Preconditions

Event preconditions are always checked (whether they exist or not) by means of defined functions that test the state of the object with tests and inhibitors arcs [11] (see *ok_destroy(), ok_Event1()*, etc.). If an event is accepted then the variable attributes may change. Possible variations of preconditions will only affect these functions. This permits to change the system specification in an easy way. It is possible to break down any Boolean condition into OPN functions by including test and inhibitor arcs.

Example 4: *Balance=0* is the precondition associated to the event *close*. In *Instances_Account* there is a function named *ok_tclose()* that tests the attribute *balance*.

Change of Object State

When an object accepts an event, the associated valuations (coming from OASIS specification) determine the possibility of the modification of the attribute values. The transition *tevent1* achieves the actions that change the tokens representing the variables (in place *Attributes*). After that, the object will probably have a set of triggers to send. Some of those triggers will be forwarded to the same object; others will be sent to another instance (in the same class or another).

Example 5: The event *deposit* implies some valuations, which are performed through the firing of transition *tdeposit*.

Derived Attributes

We can express the relation between the derived and non-derived attributes by defining OPN functions. Each access to a derived variable needs to recalculate its value through the function.

Example 6: The derived attribute *good_balance=(balance>100000)* can be expressed with the following LOOPN++ [11] function that can be used in whatever transition:

```
Boolean good_balance() = exists
    Var_type  x -- Attributes | x.balance >100000;
End exists
```

Inter-Object Communication

We are interested in two kinds of communication: asynchronous and synchronous communication. The former gives the triggering mechanism; the second refers to the OASIS shared events.

Triggers: The triggering is performed by:

1. Whoever sends the trigger prepares an event token and offers it through the *tget* transition.
2. The related class gets the token and sends it to the router class.
3. The router class forwards the event to the target class where it is sent to the server.

The transition *Calculate_Triggers* gives the relevant triggers according to the reached state (after valuations). These triggers are offered (transition *tget*) to the related class. Afterwards, the class sends the trigger event to the router or to another instance in the same class.

Example 6: The condition *(Number>10) and (good_balance=false)* enables the trigger *self::pay_Commision*. The object will put the event *pay_Commision* into the place *Triggers*. Afterwards, it is routed to the class *Class_Account*. To make it simple, we have omitted some relevant matters, such as:

1. If the target is the same object, the trigger is get into the place *Current_Event* without outgoing the instance.
2. Transition *Calculate_Triggers* may need several transitions to express the OASIS specifications.
3. Some additional features are needed to allow calculating triggers although there were no event in the previous state.

Shared Events: We need an object (e.g. a coordinating object) that broadcasts toward all the objects that have defined the shared event. Furthermore, it must be executed following an *all-or-nothing* policy. A dialogue between the coordinator and the servers is established and the protocol that follows is similar to the Two-Phase Commit protocol used in Distributed Databases. We have established an algorithm to control efficiently these kinds of events [15].

Process Specification

Possible objects lives are defined by means of a process specification. These processes are constructed using events and transactions as terms.

OPNs gives a natural way to represent process specification expressed by process algebra. Each possible state determines an OPN place. A token in such place indicates that the object is in such state. Changes from one state to another may be restricted with boolean functions (restricted to current state).

A simplified OPN that implements the process section in the example *Account* is shown in Figure 5.

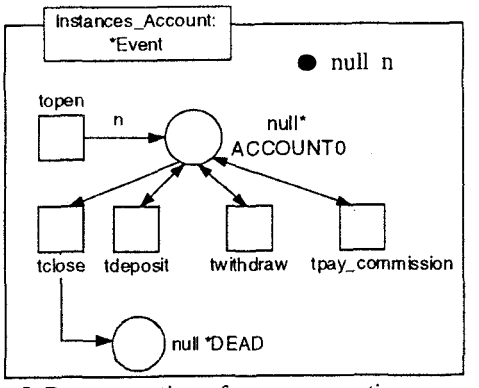

Figure 5: Representation of a process section.

Observe that the transitions included are defined in the same way as previously done in order to indicate acceptance of events. Thus, we can limit transitions firing depending on the current state and on the history of previous events. As already mentioned, condition over variables can be included in process section. As usually, those conditions will be expressed by OPN functions.

7. CONCLUSION

We have shown how the main features of our Object Oriented Conceptual Modelling Language OASIS can be naturally and directly represented in OPNs. Object Oriented concepts (such as classes, instances, interaction between objects, encapsulation, etc.) have been addressed by using the properties of the OPNs.

The support of multiple levels of activity in the OPNs makes the design of architectures easy enough to prototype a society of concurrent objects. Using the execution model of OASIS as a guide, we can guarantee an accurate animation of the OASIS specification. We are now working to obtain LOOPN++ code automatically from an OASIS specification. Our final aim is to integrate this work inside a CASE tool for system modelling supporting the OASIS model.

8. REFERENCES

[1] Hartmann T., Saake G., Jungclaus R., Hartel P., Kusch J., "Revised Version of Modelling Language TROLL", Informatik-Bericht 94-03, TU Braun-schweig, 1994.

[2] Letelier P., Sánchez P., Ramos I., "Conceptos Básicos de Especificaciones OASIS implementados utilizando Programación Lógica Concurrente", Technical Report DSIC-II/32/97, Universidad Politécnica de Valencia, 1996.

[3] Letelier P., Sánchez P., Ramos I., "Un Modelo Básico de Ejecución para Especificaciones OASIS en un Entorno Concurrente", Technical Report DSIC-II/10/97, Universidad Politécnica de Valencia, 1997.

[4] Pastor O., "Diseño y Desarrollo de un Entorno de Producción Automática de Software basado en el modelo orientado a Objetos", Tesis doctoral (dirigida por Dr. Isidro Ramos), DSIC-UPV, Valencia, 1992.

[5] Pastor O., Ramos I.,"OASIS versión 2 (2.2): A Class-Definition Language to Model Information Systems Using an Object-Oriented Approach", SPUPV-95.788, Universidad Politécnica de Valencia, 1995.

[6] Pelechano V., Pastor O., "Case OO-Method: Un entorno de producción automática de software orientado a objetos", Proc.of CIL-95, Barcelona, 1995.

[7] Sánchez P., Ramos I., "Object Petri Nets: Modelo de Implementación para OASIS", Technical Report DSIC-II/17/96, Universidad Politécnica de Valencia, 1996.

[8] Wieringa R.J. "A Conceptual Model Specification Language (CMSL Version 2)", Technical Report IR-248, Vrije Universiteit, Amsterdam, 1991.

[9] Wieringa R.J. "A Formalization of Objects using Equational Dynamic Logic", 2nd Conference on Deductive and OO Databases, pages 431-452, Springer-Verlag, Lecture Notes in Computer Science 566, 1991.

[10] Lakos C., "From Coloured Petri Nets to Object Petri Nets", Proceedings of 16th International Conference on the Application and Theory of Petri Nets, LNCS 935, Torino, Italy, Springer-Verlag, 1995.

[11] Lakos C., Keen C., "LOOPN++: A new Language for Object-Oriented Petri", Proceedings of Modelling and Simulation, Barcelona, Society for Computer Simulation, 1994.

[12] Keen C., Lakos C. "Information Modelling using LOOPN++, an Object Petri Net Scheme", Proceedings of 4th International Working Conference on Dynamic Modelling and Information Systems, 1994.

[13] Lakos C., Keen C., "An Open Software Engineering Environment Based on Object Petri Nets", Department of Computer Science, University of Tasmania, 1995.

[14] Lakos C., "The Consistent Use of Names and Polymorphism to Achieve an Elegant Definition of Object Petri Nets", Technical Report R95-12, Computer Science Department, University of Tasmania, 1995.

[15] Sánchez P., Letelier P., Ramos I., "Un Algoritmo para la Gestión de Eventos Compartidos de OASIS", Technical Report DSIC-II/31/96, Universidad Politécnica de Valencia, 1996.