An Appropriate Semantics for Distributed Active Object-Oriented Databases on the Basis of CO-NETS Approach*

Nasreddine Aoumeur† Gunter Saake
Institut für Technische Informationssysteme
Otto-von-Guericke-Universität Magdeburg
Postfach 4120, D–39016 Magdeburg
E-mail: {aoumeur|Saake}@iti.cs.uni-magdeburg.de
Tel.: ++49-391-67-18659 Fax: ++49-391-67-12020

Abstract

The purpose of this paper is to present first results towards an appropriate approach for formally specifying in a uniform and simple way all aspects characterizing distributed active object-oriented databases (AOODB). Referred to as CO-NETS*, the approach is based on a complete and a formal integration of OO concepts and constructions into a an appropriate variety of algebraic Petri nets. The* CO-NETS *semantics is expressed in rewriting logic with a full exhibition of a true intra- as well as inter-object concurrency. The suitability of the approach for modeling AOODB may be highlighted as follows. First, it straightforwardly captures different forms of inheritance, object composition and aggregation leading to (a hierarchy of) complex classes we named components. Second, for constructing more complex OODB as interacting components via explicit interfaces, we propose an inter-component interaction pattern. Third, for querying and viewing the resulting* CO-NETS *database, first ideas towards an appropriate query-pattern are forwarded. Fourth, for composing business rules, we propose an appropriate event algebras which may be regarded as suitable extension of the commonly used ECA rules. Last but least, time constraints are captured by associating time-stamps with time-dependent events and transitions.*

keywords: *distributed active OO databases,* CO-NETS*, rewriting logic, specification / validation phase.*

1 Introduction

For fulfilling non-standard requirements in advanced applications—e.g. air-traffic control, computer-integrated manufacturing (CIM), network management, workflow management— (distributed) active object-oriented databases (AOODB) as a combination of properties of active as well as object oriented databases is widely accepted framework [4]. Indeed, while the object orientation offers the required structuring mechanisms (i.e. object, class, inheritance, interaction, etc) for coping with the structural and behavioural complexity in such applications, the activeness is dictated by crucial requirements of timely responses to unforeseen but critical (time-dependent) situations. The great efforts undertaken in this areas have forwarded a very promising AOODB systems, including: REACH [16], FRAMBOISE [6], and TriGS [9].

However, in spite of this number of developed systems, theoretical underpinning of active object oriented databases is more less than satisfactory. Indeed, most of proposals for the crucial specification / validation phase deal only with some AOODB features while ignoring, disadvantaging or integrating but only in *ad-hoc* manner others features. Examples of these partial formal frameworks include: Colored Petri nets [7] for composing events and specifying associated business event-condition-action (i.e. ECA) based rules; situation / activation diagrams [12] for dealing with event detection within the object oriented setting, and object / behavior diagrams [11] for capturing (passive) object concepts. Among the shortcomings of this unsatisfactory state of affairs there are mainly: (1) misunderstanding of related concepts—like which events have to be considered as passive resp. as active [4]; (2) different interpretations for the same concepts in different systems, (3) and especially the development of applications without clean, precise and verified specification leading often to incorrect implementation, difficulty of reuse, etc.

*In Proc. of the 1st. International Conference on Software Engineering Applied to Networking and Parallel / Distributed Computing (SNPD '00) , Fouchel, H. and Roger Y.L. (eds.),International Association for Computer and Information Science (ACIS), Reims, France, pp. 541-548, 2000

†This work is supported by a DAAD schlorship.

With the aim to contribute to this important and difficult problem, we present in this paper a *multi-paradigm* approach that adequately allows coping with most aspects in specifying distributed AOODB. More precisely, for the modeling and validation of distributed active object oriented databases, we propose to soundly combine ideas from high-level (algebraic) Petri nets, object-oriented paradigm and timed rewriting logic. The resulting approach is referred to as CO-NETS approach—an acronym for Concurrent Object Nets. First results of this approach have appeared in [1]

Algebraic Petri nets are chosen, first, due to their well known concurrent behaviour; where different forms of concurrency (step, interleaving, true) may be associated. henceforth by judiciously applying them in specifying databases, distributed aspects may be overcome. Moreover, as just mentioned, Petri nets have been already shown very adequate for dealing with events in active databases. Furthermore, their graphical aspects allowing visual execution of objects permit to represent and reason *explicitly*—in fact most of existing semantics frameworks deal only with the descriptive aspects (i.e. the schema)— about a given database. Last but not least their data abstraction allows for (user-) defining different forms of complex data and reasoning about.

The integration of the object oriented paradigm into such algebraic Petri nets is intended to overcome deficiencies of each other. In fact, while the object-orientation paradigm needs for true concurrency, algebraic Petri nets lack of powerful structuring mechanisms—as required in object oriented databases, like: classification, specialization/aggregation, object composition, etc.

Finally, we have adopted rewriting logic [13] as a semantical framework for our integration for the following reasons. First, rewriting logic is a true concurrency and operational semantics; which allows rapid-prototyping using rewriting techniques and MAUDE [5] in particular. Second, for handling real-time constraints timed rewriting logic [10] has been proposed as a natural extension of this logic; henceforth time-stamps can be soundly associated with transitions and events in our proposed integration. Third, for composing events a powerful yet simple language of events (or messages) was introduced in [15], and that we propose to adapt to our proposal for composing active events.

In some detail, the CO-NETS approach is based on three logically founded layers that have also to be perceived as three manageable and incremental methodological tasks in specifying AOODB.

The data layer: It is used for describing data types that have to be stored in the database (as attributes values, events parameters, etc), and that can be of a simple form like : the integer, boolean, string, etc, or of a complex form like: list, array, tuple, etc, or even as proper user definable application-specific types. For this layer, an ordered sorted algebraic (SOA) interpretation using OBJ [8] notations is adopted. Some advantages of SOA include, in addition to abstraction and different 'in-the-large' construction mechanisms (like: parameterization, importation, etc), the operational semantics based on rewriting techniques.

The object layer: For this layer which is the heart of any data model management, we propose a complete integration of object oriented concepts and algebraic Petri nets as surveyed in [14]. As main features of this integration, we cite particularly:

1. The modeling of a class as a *module* with a hidden part and an observed part with both including structural as well as behavioural aspects. While internal part allows for evolving each class, observed part is used as interface for interacting with the environment and other classes. In each class, object states are modeled as terms with identity and gathered into an appropriate 'object' place, while with each method-invocation (i.e. message type) a corresponding 'message' place is associated. Transitions reflect the body of such methods (i.e. effect of messages on object state to which they are sent); where an appropriate 'splitting /recombination' of object states allows for a full exhibition of intra- as well as inter-object concurrency.
2. The approach provides an incremental construction of components, as a hierarchy of classes, through simple and multiple inheritance (with redefinition, associated polymorphism and dynamic binding), object composition and aggregation. Such components behave with respect to an appropriate *intra-component* evolution pattern that supports intra- as well as inter-object concurrency. Moreover, due to the possibility of splitting / recombining object states out of necessity.
3. For interacting different components and thereby constructing more complex systems as cooperative components, an adequate *inter-component* interaction pattern is proposed; it enhances concurrency and preserves the encapsulated (i.e. hidden) features of each component.
4. By interpreting the CO-NETS behaviour in rewriting logic, rapid-prototyping may be generated using rewrite techniques and current implementation of MAUDE particularly.

Transaction layer: It proposes to enrich in straightforward manner the object layer by the following 'active' aspects. First, after a precise characterization of the no-

tion of active versus passive event, we propose an adequate adaptation of the event algebras proposed in [15]. This allows us to deal with most known operators for composing events in active databases. Second, we show how temporal constraints are straightforwardly supported by associating time-stamps with active time-dependent events and transitions. Third, we present first ideas for directly querying and viewing CO-NET databases using an appropriate adaptation, to the proposed framework of, a simplified SQL-like algebras.

The remaining sections of this paper are as follows. In the second section a short overview of CO-NETS object level is given. The third section introduces the main aspects of the transaction level. We close this paper by some remarks and hints to our future work.

2 The CO-NETS Object Layer

First we give how object states and messages are syntactically described leading to the notion of a (simplified) database schema. Then, for capturing the notion of database as a community of object instances we define the basis of the CO-NETS model. However, due to space limitation we kindly advice the reader to consult [3] about conceptualization of different forms of inheritance and interaction among complex databases.

2.1 Object Structure

Roughly speaking an object state is viewed as a term (with respect to the signature below) of the form $\langle I|atr_1 : val_1, ..., atr_n : val_n, atbs_1 : val'_1, ..., atbs_s : val'_s\rangle$; where I is an observed object identity taking its value from an appropriate abstract data type sorted by OId; $atr_1, ..., atr_n$ are the *local attributes* identifiers having respectively here as current values $val_1, ..., val_n$. The observed (by external components) attributes if any are identified by $atbs_1, ..., atbs_s$. Also, we assume that all attribute identifiers range over a sort denoted AID and their value range over $Value$ as a generic sort. Moreover, to allow object valued attributes we assume that $OId < Value$ (i.e. OId is a subsort of $Value$). The events or messages are just operator symbols having at least one argument ranging its sort over OId; which means that a message (contrary to a usual operator) should be destinated to (or sent by) at least one object. Moreover, in our approach we make a clear distinction between *local* or internal messages used exclusively for evolving the database state in the time and *external* messages (as imported or exported) allowing on the one hand to relate different database components and, on the other hand, to formulate transactions. Thus, hereafter first we give how such object state notion is formally defined using OBJ notations.

```
obj Object-State is
  sort AId .
  subsort OId < Value .
  subsort Attr < Attrs .
  subsort Id-Attrib.  < Object .
  subsort local-att.  obs-att.  < Id-Attrib.  .
  protecting Value OId AId .
  op _:_ :  AId Value → Attribute .
  op _,_ :  Attr Attrs → Attrs .
  op ⟨_|_⟩ :  OId Attrs → Id-Attrs .
  op _⊕_ :  Id-Attrs Id-Attrs → Id-Attrs.
  vars Atr:  Attr ; Atrs1, Atrs2:  Attrs ; I:OId .
  eq1 ⟨I|atrs1⟩ ⊕ ⟨I|atrs2⟩ = ⟨I|atrs1, atrs2⟩ .
  eq2 ⟨I|nil⟩ = I
endo .
```

Comment: In this signature the sort *Id-attributes* with the (assoc. comm.) operator $\oplus$ allows for precisely capturing the notion of parts or components of an object state. The notation $attr_i$ is just a simplification of some pairs of attribute identifiers-values i.e. it corresponds to $atr_{i1} : val_{i1}, atr_{i2} : val_{i2}, ..., atr_{ik} : val_{ik}$. The main axiom **eq1** say that an object state (resp. an already split state) can be split (resp. recombined) at any time (and when it is necessary as we will be later) into different parts (resp. using its different parts).

Example 2.1 *Throughout the rest of this paper we use a fragment of an invoicing system taken from commercial applications. In this databases application, we assume having customers and suppliers both characterized by their customer- and supplier-number (shortly, Nc and Ns), name (N), birth-day (Bd), address (Ad). On the other hand, products are characterized by their product-number (Np), price-unit (Pu) and available quantity (Qa). Each product can be ordered (by a customer) or supplied (by a supplier) by specifying the product-number and the ordered (resp. supplied) quantity. Henceforth, products with the two associated operations (methods) form a class that we denote by PRD. By ignoring the suppliers and customers classes at this level, the messages supply (sp) and order or command (cm) will be regarded as local messages and by consequence will be also all attributes. Moreover, we introduce another method that allows to increasily changing the unit-price denoted as (chg). This class may be described as follows:*

```
obj Product is .
  protecting Nat Real String Object-state.
  subsort Sp  Cm  Chg < Local_Messages.
  subsort Prd  < Local_attributes.
  subsort Id.Prd < string < OId .
  (* Local attributes *)
    op ⟨_|Np : _, Pu : _, Qa : _⟩ :
     Id.Prd Nat Real Real → Prd.
  (* Local messages *)
    op sp :  Id.Prd Real → Sp .
    op cm :  Id.Prd Real → Cm .
    op chg :  Id.Prd Real → Chg .
  vars I:Id.Prd; N:Nat ; U,Q,Qs,Qr :  Real .
endo .
```

2.2 CO-NET Object templates and classes

After having described precisely the form of object states and different kinds of events that have to be included in a given schema of an active OO databases, we present hereafter on the one hand how appropriate behaviour can be associated with such schemas as a basic CO-NET, and on the other hand how databases are captured by the notion of marked CO-NETS. However, we note that with the aim to avoid 'tedious' formal definitions, we restrict ourselves here to the intuitive (semi-formal) definition (more formal definitions can be found in [3]).

2.2.1 CO-NETS Syntax

Given a template signature as defined above, a CO-Net is defined as follows:

- The places of the CO-NETS are *precisely* defined by associating one to one for each message form a place that we called message place. Then, each place contain message instances to be sent (by firing corresponding transition) to associated object(s). Moreover, besides these message places, we associate with the object sort a place that should contain current object states of this class. Finally, we note that in the net observed messages will be depicted with bold circles.
- Local methods (i.e. methods involving only objects and messages included in one class) are conceptualized as transitions. Such transitions make in relation *only concerned part of* object states with some message instances that are sent to such parts of object states. The effects of such methods allow in general to modify some attributes of such objects, and/or consuming such message instances, and/or producing some new messages.
- The input arcs associated with such transitions should be labeled only by the enabling tokens (which are here either multisets of message terms or multiset of Id-Attributes terms); the same is for the output tokens.
- Conditions may be associated with transitions. Such conditions are of to two sorts: safety conditions and preconditions. Preconditions involve attributes and message parameter variables and are interpreted as conditions of applying (i.e. firing) the transitions. In contrast, the safety conditions may involve all relevant (components of the) object states that are involved in a given integrity constraint.
- This definition capture the notion of OODB schema. The database itself as a community of object and message instances is defined as a marked CO-NET. It consists to associate via a marking function with each place instances of associated messages or objects.

2.2.2 CO-NETS Semantics

The (behavioural) semantics allows in particular to formally capture the definition of permissible initial database state, and on the other sides it determines how such database state evolves over time (and in a true concurrent way) only into consistent states. In the OO databases, in addition to the (static) integrity constraints that have to be ensured at the creation of new databases, there is essentially also the uniqueness of object identity to ensured. Moreover, we have to propose a state evolution with exhibition of a maximal of concurrency. For the state evolution, for instance, we propose a general pattern that has to be respected in order to ensure the encapsulation property as well as the preservation of object identity uniqueness. Following such guidelines and in order to exhibit a maximal concurrency, this evolution schema is depicted in Figure 1, and it can be intuitively explained as follows: The contact of the just relevant parts of some object states of a given Cl, —namely $\langle I_1|attrs_1\rangle$;..; $\langle I_k|attrs_k\rangle$— with some messages $ms_{i1}, .., ms_{ip}$— declared as *local or imported* in this class— and under some conditions on the invoked attributes and message parameters results in the following effects: (1) The messages $ms_{i1}, .., ms_{ip}, ms_{j1}, .., ms_{iq}$ vanish; (2) there is state changes of some (parts of) object states participating in the communication, namely $I_{s1}, .., I_{st}$. Such change is symbolized by $attrs'_{s1}, .., attrs'_{st}$ instead of $attrs_{s1}, .., attrs_{st}$; (3) there may be deletion of some objects by sending them explicit delete message; (4) new messages may be sent to objects in Cl , namely $ms'_{h1}, .., ms'_{hr}$.

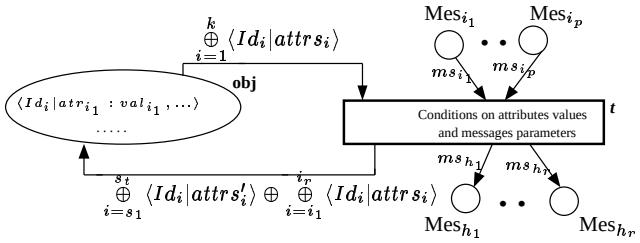

Figure 1. Intra-component evolution pattern in CO-NETS

2.2.3 Rewriting rules governing the CO-Nets behaviour

In the same spirit of the ECATNets behaviour, each CO-Net transition is captured by an appropriate rewriting rule interpreted into rewrite logic. Following the intra-component evolution pattern in Figure 1, the general form of rewrite rules that we associate with it takes the following form:

$T : (Ms_{i_1}, ms_{i_1}) \otimes .. \otimes (Ms_{i_p}, ms_{i_p}) \otimes (Ms_{j_1}, ms_{j_1}) \otimes .. \otimes (Ms_{j_q}, ms_{j_q}) \otimes (obj, \langle I_1|attrs_1\rangle \oplus .. \oplus \langle I_k|attrs_k\rangle)$

$$\Rightarrow$$

$$(Ms_{h_1}, ms'_{h_1}) \otimes ... \otimes (Ms'_{h_r}, ms'_{h_r}) \otimes$$
$$(Ms_{j_1}, ms'_{j_1}) \otimes .. \otimes (Ms'_{j_q}, ms'_{j_q}) \otimes (obj, \langle I_{s_1} | attrs'_{s_1} \rangle \oplus$$
$$.. \oplus \langle I_{st} | attrs'_{st} \rangle \oplus \langle I_{i_1} | attrs'_{i_1} \rangle \oplus .. \oplus \langle I_{i_r} | attrs'_{i_r} \rangle)$$
$$if\ Conditions \text{ and } M(Ad_{Cl}) = \emptyset \text{ and } M(Dl_{Cl}) = \emptyset$$
(**)

Comment: The operator $\otimes$ is defined as a multiset union and allows for relating different place identifiers with their current marking. Moreover, we assume that $\otimes$ is distributive over $\oplus$ i.e. $(p, mt_1 \oplus mt_2) = (p, mt_1) \otimes (p, mt_2)$ with mt_1, mt_2 multiset of terms over $\oplus$ and p a place identifier. The condition $M(Ad_{Cl}) = \emptyset$ and $M(Dl_{Cl}) = \emptyset$, in this rule, means that the creation and the deletion of objects have to be performed at first; In other words, before performing the above rewrite rule the markings in the Ad_{Cl} as well of the Dl_{Cl} places have to be empty. This allows particularly to avoid inconsistency like the manipulation of an object that is already logically deleted (i.e. a corresponding delete message was sent to it) but it is not really or 'physically' deleted (i.e. the firing of the corresponding transition in yet performed). Finally, please note that that the selection of just the *invoked parts* of object states, in this evolution pattern, is quite possible because of the splitting /recombination axiom—that has to be performed before and in accordance with each invoked state evolution.

Example 2.2 *Following the above syntactical and semantical* CO-NETS *features, it is not difficult to describe the corresponding OO database associated with the products. As depicted in Figure 2, in addition to the object place that should contain all product instances, three respective places are associated with the messages. The corresponding transitions reflect the intuitive meaning of each message. The associated rewrite rules associated with the transitions COM, SPL and CHG take respectively the form.*

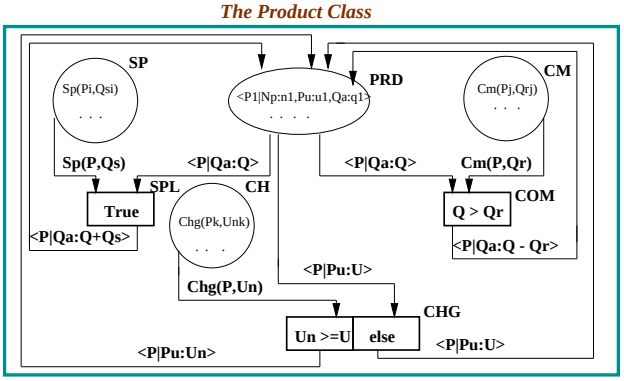

Figure 2. The Product class modeled as a CO-Net

COM: $(CM, Cm(P, Qr)) \otimes (Prd, \langle P|Qa : Q\rangle)$
$\Rightarrow (Prd, \langle P|Qa : Q - Qr\rangle)$ **if** $Q > Qr$.

SPL: $(SP, Sp(P, Qs)) \otimes (Prd, \langle P|Qa : Q\rangle)$
$\Rightarrow (Prd, \langle P|Qa : Q + Qs\rangle)$.

CHG: $(CH, Chg(P, Us)) \otimes (Prd, \langle P|Pu : U\rangle)$
$\Rightarrow$ **if** $Us \geq U$ **then** $(Prd, \langle P|Pu : Us\rangle)$ **else** $(Prd, \langle P|Pu : U\rangle)$[1].

3 The CO-NETS Transaction Level

In what follows, first, we give a more characterization of active events and how to compose them through a very rich algebras that is soundly interpreted in rewriting logic. Second, we report on the natural and founded possibility of specifying temporal events and associated time-dependent behaviour (i.e. transitions). Third, a simplified algebras is presented for querying the modeled database.

3.1 Active Events and their Algebras

3.1.1 Active events Characterization

As commonly accepted in active OO databases community, in contrast to passive events which have to be *explicitely* created, received, sent, or performed by the database user, active events are defined as those appearing autonomously and systematically in the database due a particular configuration of its internal state, and which should be performed immediately. A particular configuration may be reached when some conditions are fulfilled like specific attribute values of objects, specific derived values from the whole (database) state, or time-dependent operations that have to be performed regularly, ect. Moreover, while a response, modeled as a transition firing in our case, to a passive event is typically synchronous, most of active event responses are *asynchronous*. For adequately capturing their behaviour, the use of event-condition-action rules (shortly, ECA) is the widely adopted pattern.

For dealing with these characterizations of active events in our proposal, we propose to associate with each active event or several events cooperating for achieving a same behaviour instead of one transition like passive events rather two transitions that we named *active-condition* transition and *active-response* transition. The first kind of transitions allows for an autonomous apparition of active events (in the corresponding place) when specific conditions are fulfilled by the database state. Therefore, input places in such transitions are just the object (state) place and the corresponding event(s) places. The general pattern of such transitions can be regarded as a particular case of the intra-

[1] We state that the change-price request is ignored when the condition doesn't hold.

component pattern and is depicted in Figure 3. In this pattern, input arcs associated with (active) event places, are annotated by $\emptyset$ which means that the corresponding places have to be empty before any apparition of a new active event. In other words, this have to reflect the fact that such events must always be performed (using the corresponding *active-response* transitions) immediately after their apparition. *Active-response* transition, on the other side, should capture the corresponding effect of these (active) events and hence have to respect the general intra-component evolution pattern depicted in Figure 1. However, as result of including active events (besides passive ones), which as we just mentioned should be performed always at first, the condition $M(Ad) = \emptyset \wedge M(DEl) = \emptyset$ in the rewrite rule **(**)** for the intra-component evolution have to be enlarged for transitions involving passive events to: $M(Ad) = \emptyset \wedge M(DEl) = \emptyset \wedge M(Mes_1) = \emptyset \wedge .. \wedge M(Mes_k) = \emptyset$; where Mes_1,..,Mes_k are the places associated with events that have to be considered as active. In other words, passive messages are performed only and only if *all* active messages (and deletion and creation messages) have already been performed.

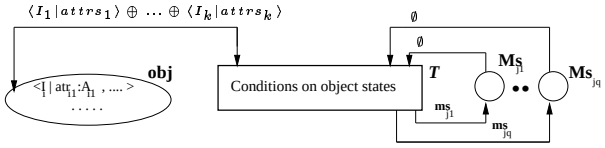

Figure 3. *active-condition* **Transitions**

Example 3.1 *Assume that when the product quantity falls under a certain limit, denoted* **Lmt**, *there is a systematic sending of an (active) event allowing for resolving this (critical) situation. Such autonomous response to this condition is described by the transition LMT in the Figure 4. Also, we assume that the proposed solution in this case is, on one hand, the increasing of this limit quantity by ten (10) units that we assume are always available. On the other hand, as a business politic in this organization, such decreasing in the quantity would have to be interpreted that the corresponding product is largely sold; as a result it proposes to increase its unit price by certain percent (for instance 0.5). This fact is modeled by the transition ALM-RS in figure 4.*

3.1.2 Event algebras and its Semantics

The event algebras that we propose of composing primitive events is mainly inspired by the message algebras proposed by Wirsing and al. in [15], but with slight adaptation to different rewrite rule patterns of our approach. This algebras may be presented as follows; where $+$, ; and $||$ stand respectively for choice, sequence and parallel composition.

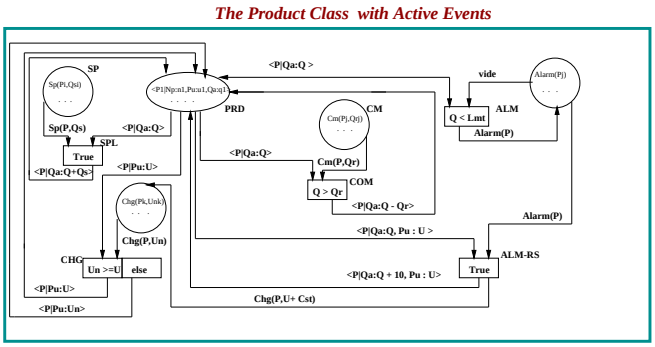

Figure 4. Example *active-condition* **Transitions**

mod CO-NETS_Algebra **is**
protecting CO-NETS-State
op $\_+\_$: Marking Marking $\rightarrow$ Marking .
op $\_;\_$: Marking Marking $\rightarrow$ Marking .
op $\_||\_$: Marking Marking $\rightarrow$ Marking .
op $do\_$ Marking $\rightarrow$ Marking .
vars $m_1, m_2, n_1, n_2 : Msg$.
vars $c, d, c_1, c_2, d_1, d_2, h$: Place_Marking .
rl $((P\_Id1, m_1) + (P\_Id2, m_2)) \otimes (P\_Id3, c) \Rightarrow (P\_Id, d)$
if $(P\_Id1, m_1) \otimes (P\_Id3, c) \Rightarrow (P\_Id, d) \vee$
$(P\_Id2, m_2) \otimes (P\_Id3, c) \Rightarrow (P\_Id, d)$.
rl $(P\_Id1, m_1 \oplus c1)\ ;\ (P\_Id2, m_2 \oplus c2) \Rightarrow$
$(P\_Id3, c3) \otimes (P\_Id4, c4)$
if $(P\_Id1, m_1 \oplus c1) \Rightarrow (P\_Id3, c3) \otimes (P\_Id, c)$
$\vee (P\_Id2, m_2 \oplus c2) \otimes (P\_Id, c) \Rightarrow (P\_Id4, c4)$.
rl $(P\_Id1, m_1 \oplus c1)\ ||\ (P\_Id2, m_2 \oplus c2) \Rightarrow$
$(P\_Id3, c3) \otimes (Pl\_Id4, c4)$
if $(P\_Id1, m_1 \oplus c1) \Rightarrow (P\_Id3, c3)$
$\vee (P\_Id2, m_2 \oplus c2) \Rightarrow (P\_Id4, c4)$.
rl $do\ (P\_Id, m \oplus c) \Rightarrow$
$(\_Id, m \oplus c)\ ;\ do\ (P\_Id, m \oplus c)$.
rl $do\ (P\_Id, m \oplus c) \Rightarrow nil.$
endm

Remark that *do* message composition operator for infinite repetition can be easily adapted to be applied only a fixed number of times. In this case it becomes:

op $do\_number\_$: Nat Marking $\rightarrow$ Marking .
var n : Nat .
rl $n\ do\_number\ (P\_Id, m \oplus c) \Rightarrow$
$(P\_Id, m \oplus c)\ ;\ (n-1)\ do\ (P\_Id, m \oplus c)$.
rl $0\ do\ (P\_Id, m \oplus c) \Rightarrow nil.$

It worth mentioning finally that a composite action can be either described as an appropriate equation or using appropriate notations like the one introduced in [15] which takes the form:

cntrl [composite_message_name] corresponding messages expression.

Example 3.2 *Assume in the product class that, for instance, after each ten (10) successive command operations (i.e. the firing of the transition CM), we increase the price by .0001 or automatically supply the corresponding product with twenty (20) additional units. This fact can now be straightforwardly modeled using a sequential composition*

(i.e. ;) between do_number (with ten time of command operations) and change of the price followed by a supply of the concerned product. More precisely, this transaction denoted by TRANS takes the following form:

cntrl *[TRANS]* $[10\ do\ (CM, CM(P,-)\langle P|Qa:Q, Pu:U\rangle)\ ;\ (CHG, Chg(P, U + .0001 * U))] + (SP, Sp(P, Qa + 20))$

3.2 Modeling of Timing-Constraints

Due to the founded extension of rewriting logic to timed rewriting logic, firstly introduced in [10], and because of our interpretation of CO-NETS semantics in this logic, is it quite natural to deal with time-sensitive events and their associated transitions. Without going into detail, the main ideas in timed rewriting logic consists in associating with each time-dependent rewrite rule a *time-stamp* (as an additional label) representing the time that have to be taken by the rule to be performed. In other words, each (time-dependent) rewrite rule of the form $\mathbf{rl} : l \Longrightarrow r$ that takes t unit of time[2] is rather represented as $\mathbf{rl} : l \stackrel{t}{\Rightarrow} r$.

Example 3.3 *Assume that to optimally managing the product prices, at the first of each month at a given time T there is a reorganization of all product prices. This consists in decreasing the price by 5 percent of all products which have not been sold during months; increasing the price by 1 percent of all products which have been sold in less than ten (10) days or their global sold quantity is greater than* 1000 *units; finally if a given product have not been sold since one year it will be completely removed. This business politic is captured by the transition $RORG$ as depicted in Figure 5. For this aim, we have first rehabilitated the command operation in such a way that we can keep trace of all ordered operations; this is achieved by introducing as a subsort*[3] *of the command $Cm(P,Q)$ message. That is, for ordering a quantity of a given product P, a new message denoted $Cm(P,Q,t)$ where t denotes the current time of the operation, as is captured by the transition COM. After each command operation there is a creation of this time-dependent message. Second, we require that the reorganization operation have to be performed in an interval of twenty (20) minutes as inscribed in the right hand of the transition $RORG$. Hereafter we restrict ourselves to give the corresponding (time) rewrite rule of this transition.*

$$(RORG, Rorg(P, 1stMonth, T)) \otimes (PRD, \langle P|Pu : U\rangle) \otimes (CM, All(Cm, P, Q, t)) \stackrel{[T+20mn, T+40mn]}{\Longrightarrow}$$

[2] Noting that time intervals are also possible [10].

[3] In this way the two sorts of messages are modeled using the **same** place.

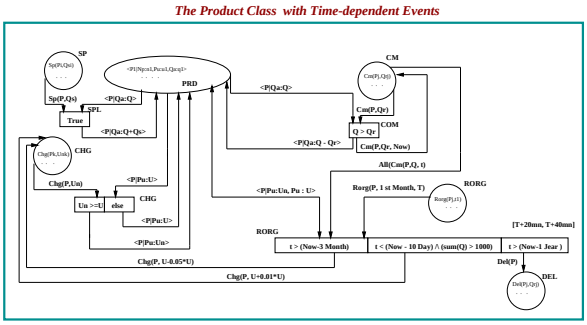

Figure 5. General form of *active-condition* Transitions

$$\textbf{if}\ \ (t > (Now - 3\ Month)\ \ \textbf{then}\ \ \langle P|Pu : U\rangle \otimes (CHG, Chg(P, U - 0.05 * U))\ \ \textbf{else}\ \ \texttt{if}\ \ (t < (Now - \ 10\ Month) \wedge (sum(Q) > 1000)\ \ \textbf{then}\ \ \langle P|Pu : U\rangle \otimes (CHG, Chg(P, U + 0.01 * U))\ \ \texttt{else if}\ \ (t > (Now - 1\ Year)\ \ \texttt{then}\ \ Del(P)$$

3.3 Views and Queries in CO-NETS

Given a database composed of several components as previously described, we propose to adapt and hence perform the following SQL-like query:

select *special-form of objects* **from** *Class1,....,Classn*
where *conditions on attributes objects* (*)

For this aim, we propose following modeling steps: First, because this query pattern is concerned only with object states and not with messages, we hide the message places and all the associated transitions.Second, for receiving the query results (i.e *special-form of objects* in (*)), we introduce *new place(s)* having as sort(s) the form of such query (usually as tuples) that we specify algebraically using the data layer. Third, to capture the body of the query we introduce an appropriate transition, having as input the (object) places concerned by the query and as output places the newly introduced places. We should note here, as depicted in the left hand of Figure 6, that destroyed tokens associated with input arcs should be annotated by empty multisets; which means that the involved objects in a given query must be conserved. Finally, the use of the operator $\sim$ allows to avoid duplication of generated results.

Example 3.4 *Assume we want to select identities and names of products those unit price are greater or equal than 2500. The net associated with this simple query is depicted in the right hand of Figure 6.*

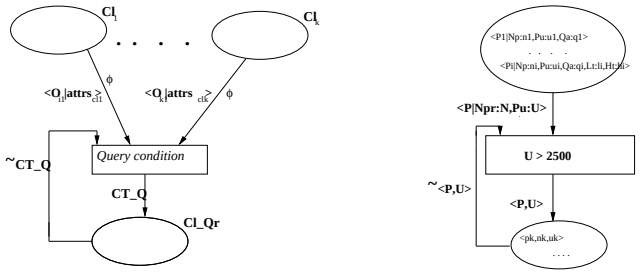

Figure 6. Query processing schema using CO-NETS

4 Concluding Remarks

We presented a logical semantics for modelling distributed active object oriented databases. The conceptual model referred to as CO-NETS is conceived around an appropriate integration of object-oriented concepts and constructions into an algebraic high level Petri net variety. CO-NETS semantics is expressed in (timed) rewriting logic. From methodological guidelines, the model is constructed around three logically related layers. The data layer specified algebraically allows for describing, structuring and reasoning about the data to be stored in the database. The heart of the model is the object layer, where a simple database is conceived as a hierarchical classes. More complex databases can be specified by interconnecting different (database) components through their interfaces. The transaction layer allows, first, for composing different active events using a simple but powerful algebras. Second, it permit for dealing with time-dependent events and behaviour. Finally, we have shown through a simplified query schema how the proposed model allows for querying the database in a machine independent way.

After this crucial step towards a satisfactory semantics for fully distributed and active object oriented databases, we are conscious that much work remains ahead. In this sense we plan for nearest future to deal with a more complex applications characterized as active databases, and this in order to assessing more the proposed framework against realistic applications. Moreover, it will also be interesting to focus on the relation of this framework to commonly used diagrams including Situation /Activation diagrams and Object / Behaviour diagrams and how they can be used in front of our model i.e. in the analysis phase.

References

[1] N. Aoumeur and G. Saake. Towards an Object Petri Nets Model for Specifying and Validating Distributed Information Systems. In M. Jarke and A. Oberweis, editors, *Proc. of the 11th Int. Conf. on Advanced Information Systems Engineering, CAiSE'99*, volume 1626 of *Lecture Notes in Computer Science*, pages 381–395. Springer-Verlag, 1999.

[2] N. Aoumeur and G. Saake. An Appropriate Semantics for Distributed Active Object-Oriented Databases on the Basis of the CO-NETS Approach. Preprint, Fakultät für Informatik, Universität Magdeburg, 2000.

[3] N. Aoumeur and G. Saake. CO-NETS: A Formal OO Framework for Specifying and Validating Distributed Information Systems. Preprint Nr. 2, Fakultät für Informatik, Universität Magdeburg, 2000.

[4] A. P. Buchmann. Active Object Systems. In A. Dogac, M. T. zsu, A. Biliris, and T. Sellis, editors, *Advances in Object-Oriented Database Systems*. Springer Verlag, 1994.

[5] M. Clavel, F. Duran, S. Eker, J. Meseguer, and M. Stehr. Maude : Specification and Programming in Rewriting Logic. Technical report, SRI, Computer Science Laboratory, March 1999. URL : http://maude.csl.sri.com.

[6] H. Fritschi, S. Gatziu, and K. R. Dittrich. FRAMBOISE - an Approach to Framework-Based Active Database Management. In *Proc. of 7th Conference on Information and Knowledge Management (CIKM)*, Waschington DC, November 1998.

[7] S. Gatziu. *Events in an Active Object Oriented Database System*. PhD thesis, Kovac University, 1995.

[8] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J. Jouannaud. Introducing OBJ. Technical Report SRI-CSL-92-03, Computer Science Laboratory, SRI International, 1992.

[9] G. Kappel and W. Retschitzegger. The TriGS Active Object-Oriented Database System - An Overview. *SIGMOD Record*, 27(3), September 1998.

[10] P. Kosiuczenko and Wirsing. Timed Rewriting Logic with an Application to Object-Based Specification. *Science of Computer Programming*, 28:225–246, 1997.

[11] P. Lang, W. Obermair, W. Kraus, and T. Thalhammer. Graphical Editor for the Conceptual Design of Business Rules. In *Proc. of the 14th International Conference on Data Engineering (ICDE)*. IEEE Computer Society Press, 1998.

[12] P. Lang, W. Obermair, and M. Schrefl. Modeling Business Rules with Situation / Activation Diagrams. In *Proc. of the 13th International Conference on Data Engineering (ICDE)*, pages 455–464. IEEE Computer Society Press, 1997.

[13] J. Meseguer. Conditional rewriting logic as a unified model for concurrency. *Theoretical Computer Science*, 96:73–155, 1992.

[14] W. Reisig. Petri Nets and Abstract Data Types. *Theoretical Computer Science*, 80:1–30, 1991.

[15] M. Wirsing and A. Knapp. A Formal Approach to Object-Oriented Software Engineering. In J. Meseguer, editor, *Proc. of the First Inter. Workshop on Rewriting Logic*, volume 4. Electronic Notes in Theoretical Computer Science, 1996.

[16] J. Zimmermann and A. P. Buchmann. REACH. In P. Norman, editor, *Active Rules in Database Systems*. Springer Verlag (New York), 1998.