Towards an Object Petri Nets Model for Specifying and Validating Distributed Information Systems

Nasreddine Aoumeur* and Gunter Saake

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

Abstract. We present first results towards a tailored conceptual model for advanced distributed information systems regarded as open reactive and distributed systems with large databases and application programs. The proposed model, referred to as CO-Nets, is based on a complete integration of object oriented concepts with some constructions from semantical data modeling into an appropriate variant of algebraic Petri Nets named ECATNets. The CO-Nets behaviour is interpreted into rewriting logic. Particularly, it is shown how CO-Nets promote incremental construction of complex components, regarded as a hierarchy of classes, through simple and multiple inheritance (with redefinition, associated polymorphism and dynamics binding). Each component behaves with respect to an appropriate intra-component evolution pattern that supports intra- as well as inter-object concurrency. On the other hand, we present how such components may be interconnected, through their interfaces, with respect to an inter-component interaction pattern that enhances concurrency and preserves the encapsulated features of each component. Moreover, by interpreting the CO-Nets behaviour into rewriting logic, rapid-prototypes can be generated using rewrite techniques and current implementation of the MAUDE language particularly. The CO-Nets approach is presented through a simplified Staff management system case study.

1 Introduction and Motivation

Due to an ever-increasing in size and space of present-day organizations, both system designers and users of information systems —that are intended to represent and then computerize accurately (the most part of) such realities— are nowadays confronted by challenging problems. Indeed, while the size-increasing has resulted in more and more complex and multi-layered systems those components are loosely connected and behaving mostly in a true concurrent way,

* This work is supported by DAAD grant.

M. Jarke, A. Oberweis (Eds.): CAiSE'99, LNCS 1626, pp. 381–395, 1999.

the space dimension is reflected by a high distribution of such systems over different (geographical) sites, where different forms of (synchronous/asynchronous) communication are often required.

With the aim to master such challenging dimensions that go beyond the traditional information systems specification models — including, data-oriented models in general [EM85] [HK87], process-oriented models (CCS [Mil89], Petri nets [Rei85]) or even 'sequential' object oriented (OO) models— recent research advocates particularly the *formal* integration of the *object paradigm* with *concurrency* as basic requirement for advanced conceptual modeling [FJLS96] [ECSD98].

The great efforts undertaken in the last decade, towards providing the (concurrent) OO paradigm with theoretical underpining foundation, have forwarded very promising OO formalisms. More particularly, OO foundations are based on the algebraic setting in the general sense including HSOA [GD93]; DTL (Distributed Temporal Logic) [ECSD98] and rewriting logic [Mes92]. OO formalisms based on (high level) Petri nets have also been proposed recently including: OPNet [Lak95], CLOWN [BdC93], CO-OPNets [Bib97].

However, we claim that much work remains ahead and there is up-to now no ideal approach that fulfills all the expected requirements. Indeed, besides a sound formalization of OO concepts and constructions, an appropriate approach for the distributed information systems should be, among others, appropriate for validation /verification purpose and more or less easy to be understand to non-experts users. Last but not least database aspects like views and queries are also important.

On the basis of these motivations, we present in this paper the first results of our investigation towards a suitable conceptual model, for specifying distributed information systems, that fulfills (as more as possible) the above requirements. The model, referred to as CO-Nets, is based on a complete and a sound integration of OO concepts and constructions into an appropriate variant of algebraic Petri nets [Rei91] named ECATNets [BMB93]. The semantics of the CO-Nets approach is expressed in rewriting logic [Mes92]. In some details, particular to the CO-Nets approach are the following features:

- A clean separation between *data* —specified algebraically[1] and used for describing object attributes (values and identifiers) and message parameters— and *objects* as indivisible and uniquely identified units of structure and behaviour.
- The modeling of simple and multiple inheritance, with possibility of methods overriding, is achieved in a straightforward way using subsorts and an appropriate 'splitting and recombination' axiom (inherent to the CO-Nets semantics) of the object state at a need. In the sequel, a hierarchy of classes will be referred to as a *component*.
- For capturing the notion of a class, rather as a module, we distinguish between local, and hence hidden from the outside features (as a part of the object state as well as some messages) and external, observed and possibly

[1] Using ordered sorted algebra(OSA).

modifiable from the outside features. The local behaviour of each module is ensured by an appropriate intra-component evolution pattern that enhance intra- as well as inter-object concurrency, while the interaction between different (external features of) modules follows an inter-component communication pattern that enhance concurrency and preserve the encapsulated features of each module.
- The CO-Nets semantics is interpreted in rewriting logic that is a *true-concurrent* operational semantics allowing particularly rapid-prototyping using concurrent rewriting techniques in general and current implementation of the MAUDE language specifically.

The rest of this paper is organized as follows: In the second section we informally introduce the simplified staff management case study. In the third section, using this case study and without entering into technical details (that can be found in [Aou99]), we show how the CO-Nets approach allows for specifying (independent) templates and classes. The fourth section deals with the semantics aspects of the CO-Nets approach. In the next section, we introduce how more advanced abstraction mechanisms, with mainly inheritance and interaction, are modeled in the CO-Nets. Some concluding remarks are given in the last section.

For the rest of the paper we assume the reader is familiar with some basic ideas of algebraic Petri nets and rewriting techniques. Good references for these topics are [EM85, Rei91] for the algebraic setting and algebraic Petri nets, and [DJ90, Mes92] for rewriting techniques. We use for algebraic descriptions an OBJ notation [GWM+92]. Moreover, due to space limitation, a deep introduction to the ECATNets model can be found in [Aou99]

2 The Staff Management Case Study

This section presents a simplified form of a general case study dealing with a staff management more tailored to the (Algerian) universities staff systems [Aou89]. The staff management we consider concerns the employees-career management and the employee payment system; however, due to space limitation we present only the first system. Depending on their different functionalities, we find three classes of employees: The lecturer (and the researchers), the administrators and the technical and security staff. For sake of simplicity, we abstract away from this difference, and only deal with their common characteristics.

According to the 'states' through which can pass each employee, the staff management system can be informally described as follows:

- Each person, verifying some conditions like the (minimal) age, the required degree (or necessary formation), etc, can apply for a job at the university. The minimal information that should be provided in this case are: the name, the surname, the birth-day, the diploma, the address, his/her familiar situation and so forth. If (s)he is accepted as a new employee (i.e. if there is sufficiently budget items corresponding to the inquired function and the administrative committee estimate this recruitment positively), (s)he becomes an employee

(as probationer) at the university. In this case some further information is systematically added like the function, the reference number (for uniquely identifying each employee), the department name to which (s)he is appointed and the recruitment date.
- After some period that go from nine months to two years and only if the employee have had in this (probation) period no caution, (s)he is appointed as a titular; in which case, further informations will be added like the number of rungs (initialized by 1), (administrative) responsabilities if any, etc. Also, we note that each titular employee may progress (with one unit) in the rung after a period that go from one to three years.
- Each employee, on probation or titular, can go on a leave; where, two kinds of leaves are possible: regular leaves, that are granted to all employees, with a fixed, same period and date (generally, at the beginning of July for 45 days). The exceptional leave such as sick leave necessitates in addition to the period and the date, the matter of such leave.
- After some professional misconducts, the employee may be subject to disciplinary measures that go from salary diminution or a warning to a complete dismissal.
- Each employee may leave temporary or completely the university when necessary. Partial leaves are for example scientific leave to another university (for researcher) or improvement leave (for the administrators and lecturers). The complete leaves, are for example resignation, pensioned off, etc.

3 CO-Net: Template and Class Specification

This section deals with the modelling of the basic concepts of the object oriented paradigm, namely objects, templates and classes. We first present the structure or what is commonly called the 'object' signature templates [EGS92], then we describe how 'specification' templates and classes are specified.

3.1 Template Signature Specification

The template signature defines the structure of the object states and the form of operations that have to be accepted by such states. Basically, in the CO-Nets approach, we follow the general object signature proposed for MAUDE [Mes93]. That is to say, object states as regarded as terms —precisely as a tuple— and messages as operations sent or received by objects. However, apart from these general conceptual similarities, and in order to be more close to the aforementioned information system requirements, the OO signature that we propose can be informally described as follows:

- The object states are terms of the form $\langle Id | atr_1 : val_1, ..., atr_k : val_k, at\_bs_1 : val'_1, ..., at\_bs_{k'} : val'_s \rangle$; where Id is an observed object identity taking its values from an appropriate abstract data type OId; $atr_1, .., atr_k$ are the local, hidden from the outside, attribute identifiers having as current values

respectively $val_1, .., val_k$. The observed part of an object state is identified by $at\_bs_1, ..., at\_bs_s$ and their associated values are $val'_1, ..val'_s$. Also, we assume that all the attribute identifiers (local or observed) range their values over a suitable sort denoted AId, and their associated values are ranged over the sort $Value$ with $OId < Value$ (i.e. OId as subsort of $Value$) in order to allow object valued attributes.

- In contrast to the indivisible object state proposed in MAUDE that avoid any form of intra-object concurrency, we introduce a powerful axiom, called 'splitting / recombination' axiom permitting to split (resp. recombine) the object state out of necessity. As will be more detailed later, this axiom, that can be described as follows: $\langle Id|attrs_1{}^2, attrs_2\rangle = \langle Id|attrs_1\rangle \oplus \langle Id|attrs_2\rangle$, allows us in particular, first, to exhibit intra-object concurrency[3]. Second, it provides a meaning to our notion of observed attributes by allowing separation between intra- and inter-component evolution (see later). Third, it allows us to simplify drastically the conceptualization of the inheritance.
- In addition of conceiving messages as terms —that consists of message's name, the identifiers of the objects the message is addressed to, and, possibly, parameters— we make a clear distinction between internal, local messages and the external as imported or exported messages. Local messages allow for evolving the object states of a given class, while the external ones allow for communicating different classes using exclusively their observed attributes.

All for all, following these informal description and some ideas from [Mes93], the formal description of the object states as well as the classes structures, using an OBJ [GWM$^+$92] notation, takes the form presented in figure 1:

Remark 1. The local messages to a given class Cl have to include at least the two usual messages: messages for creating a new object state and messages for the deletion of an existing object; we denote them respectively by Ad_{Cl} and Dl_{Cl}.

Example 1. As informally described, each employee (on probation) have to be characterized, at least, by his name (shortly, Nm), surname(Sn), birthday(Bd), address(Ad), diploma(Dp), function(Fc) and recruitment date(Dr). Hence, w.r.t. our state structure, the employee structure takes the form $\langle Id|Nm : N_i, Sn : S_i, Bd : B_i, Ad : Ai, Dp : Dp_i, Fc : F_i, Dr : Dr_i\rangle$; where $N_i, S_i, B_i, A_i, Dp_i, Dr_i$ are the actual values of the attributes and Id is a logical identity representing the employee reference number. Besides this state structure, the main operations (regarded as messages) that we consider are: the departure on leave, $Lv(Id, Dt, Pd, Mt)$, where Id is the identity of the concerned employee, Dt the date, Pd is the period and Mt is the matter. The sanction operation, $Sc(Id, Mt,$

[2] $attr_i$ stands for a simplified form of $atr_{i1} : val_{i1}, ..., atr_{ik} : val_{ir}$.

[3] In the sense that two messages sent to the same object and acting on different attributes can be performed (i.e. rewritten) parallely by splitting the two parts using this axiom.

Dg), with Mt as the matter and Dg is the kind of the sanction[4]. Also, we give the possibility for an employee to change his/her address, $Chg(Id, Nad)$, with Nad as a new address. With respect to the operation of leave, first we associate another message for controlling the respective (date of) return to the work; second we introduce a new attribute denoted as Lv that indicates if an employee is on leave or not. Similarly, we add a new attribute Sc that keeps the sanction information.

```
obj Object-State is
 sort AId .
 subsort OId < Value .
 subsort Attribute < Attributes .
 subsort Id-Attributes < Object .
 subsort Local-attributes External-attributes < Id-Attributes .
 protecting Value OId AId .
 op _:_ : AId Value → Attribute .
 op _,_ : Attribute Attributes → Attributes [associ. commu. Id:nil] .
 op ⟨_|_⟩ : OId Attributes → Id-Attributes .
 op _⊕_ : Id-Attributes Id-Attributes → Id-Attributes
   [associ. commu. Id:nil] .
 vars Attr: Attribute ; Attrs1, Attrs2: Attributes ; I:OId .
 eq1 ⟨I|attrs1⟩ ⊕ ⟨I|attrs2⟩ = ⟨I|attrs1, attrs2⟩ .
 eq2 ⟨I|nil⟩ = I endo.

obj Class-Structure is
 protecting Object-state, s-atr1,...,s-atrn, s-arg11,1,.., s-argl1,l1,
            ...,s-argi1,1,...,s-argi1,i1 ...
 subsort Id.obj < OId .
 subsort Mesl1, Mesl2,...,Mesll < Local_Messages .
 subsort Mese1, Mese2,...,Meseе < Exported_Messages .
 subsort Mesi1, Mesi2,...,Mesii < Imported_Messages .
 sort Id.obj, Mesl1, . . . ,Mesip
 (* local attributes *)
  op ⟨_|atr1 : _, ..., atrk : _⟩ : Id.obj s-atr1 ...s-atrk
     → Local-Attributes.
 (* observed attributes *)
  op ⟨_|atrbs1 :, ..., atrbsk′ : _⟩ : Id.obj s-atbs1 ...s-atbsk′
     → External-Attributes.
 (* local messages *)
  op msl1: s-argl1,1 ...s-argl1,l1 → Mesl1 . ...
 (* export messages *)
  op mse1: s-arge1,1 ...s-arge1,e1 → Mese1 . ...
 (* import messages *)
  op msi1: s-argi1,1 ...s-argi1,i1 → Mesip . ...
endo.
```

Fig. 1. The template signature specified in an OBJ notation

[4] With the convention that $Dg = 1$ corresponds to a warning and $Dg = 2$ to a complete dismissal.

Respecting the general template schema described above, the employee template may be described as follows:

```
obj employee is
 extending Class-structure .
  protecting nat   string   date .
  sorts Emp   LV  SC   CHG   PYM  RET .
 subsort Local-Emp Observed-Emp < Emp .
 subsort Id.emp < OId .
 (* Hidden attributes *)
  op ⟨_|Dr : _, Ad : _, Dp : _, Lv : _, Sc : _⟩ : Id.emp date
    string string  bool nat→Local-Emp.
 (* Observed attributes (by the payment system) *)
  op ⟨_|Nm : _, Sn : _, Bd : _, Fc : _⟩ :
    Id.emp  string string  date  string → Observed-Emp.
 (* Local messages *)
  op Lv : Id.emp   date   nat   string → LV .
  op Sc : Id.emp   string   nat → SC .
  op Chg : Id.emp  string → SC .
  op Ret : Id.emp  date → RET .
 (* Exported messages *)
  op Py : Id.emp   string   nat   → PYM .
endo.
```

3.2 Template and Class Specification

On the basis of the template signature, we define the notion of *template specification* as a CO-Net and the notion of class as a marked CO-Net. Informally the associated CO-Net structure, with a given template signature, can be described as follows:

- The places of the CO-Net are precisely defined by associating with each message generator one place that we called 'message' place. Henceforth, each message place have to contain message instances, of a specific form, sent to the objects (and not yet performed). In addition to these message places, we associate with each object sort one 'object' place that has to contain the current object states of this class.
- The CO-Net transitions reflect the effect of messages on the object states to which they are addressed. Also, we make distinction between local transitions that reflect the object states evolution and the external ones modeling the interaction between different classes. The requirements to be fulfilled for each transition form are given in the subsection below. The input arcs are annotated by the input conditions, while the output arcs are labelled by the created tokens. Both inscriptions are defined as multisets of terms respecting the type of their input and/ot output places—the associated union operation is denoted by $\oplus$.
- Conditions may be associated with transitions. They involve attribute and/or message parameters variables.

Example 2. For the staff management system that we consider here we have just the employee class. Following the above class definition, the CO-Net describing this class is composed of an object place denoted by *Emp* containing the employee state and of five message places denoted by *Leav*, *Punish* and *Chg*, *return* and *del* corresponding respectively to the (go on)leave, (being) punished, change of the address, the return from a leave and possibly the fire. The effect of each message is described by a transition that takes into account just the relevant attributes. For example, the message $Chg(Id, Nad)$ that allows to change the address of an employee identified by Id enters into contact with just the address component of this employee state. This is possible only due to the 'splitting / recombination' axiom. Following this, the CO-Net representing the employee class is depicted in figure 2. The symbol ϕ denotes $\emptyset$ as in the ECATNets model [BMB93], and it means that the invoked tokens should not to be deleted.

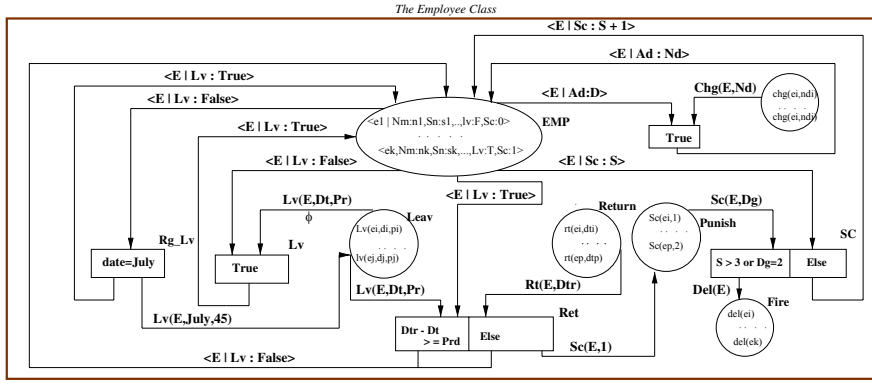

Fig. 2. The Employee Class Modeled as CO-ATNet

4 CO-Nets: Semantical Aspects

After highlighting how CO-Nets templates, as description of classes, are constructed, we focus herein on the behavioural aspects of such classes. That is, how to construct a *coherent* object society as a community of object states and message instances, and how such a society evolves only into a *permissible* society. By coherence it is mainly meant the respect of the system structure, the uniqueness of object identities and the non violation of the encapsulation property.

Objects Creation and Deletion. For ensuring the uniqueness of objects identities in a given class denoted by Cl, we propose the following conceptualization:

1. Add to the associated (marked) CO-Net (modeling a class) a new place of sort $Id.obj$ and denoted by $Id.Cl$ containing *actual objects identifiers* of object states in the place Cl.

2. Objects creation is made through the net depicted in the left hand side of figure 3. The notation ~, borrowed from the ECATNets model, captures exactly the intended behaviour (i.e. the identifier Id should not already be in the place $Id.Cl$). After firing this transition, there is an addition of this new identifier to the place $Id.Cl$ and a creation of a new object, $\langle Id|atr_1 : in_1, ..., atr_k : in_k\rangle$, with $in_1, ..., in_k$ as optional initial attributes values.

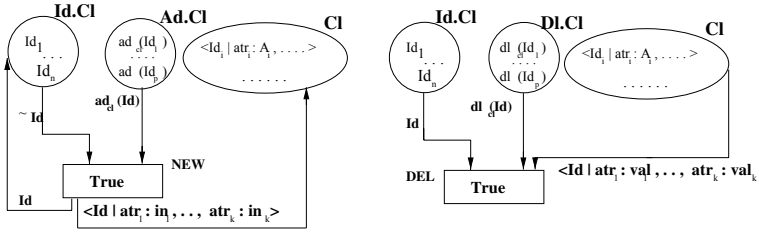

Fig. 3. Objects Creation and Deletion Using OB-ECATNets

Evolution of Object States in Classes. For the evolution of object states in a given class, we propose a general pattern that have to be respected in order to ensure the encapsulation property—in the sense that no object states or messages of other classes participate in this communication — as well as the preservation of the object identity uniqueness. Following such guidelines and in order to exhibit a maximal concurrency, this evolution schema is depicted in Figure 4, and it can be intuitively explained as follows: The contact of the just relevant parts of some object states of a given Cl, —namely $\langle I_1|attrs_1\rangle$[5] ;..; $\langle I_k|attrs_k\rangle$— with some messages $ms_{i1}, .., ms_{ip}, ms_{j1}, .., ms_{jq}$—declared as *local or imported* in this class— and under some conditions on the invoked attributes and message parameters results in the following effects:

- The messages $ms_{i1}, .., ms_{ip}, ms_{j1}, .., ms_{iq}$ vanish;
- The state change 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 $attrs_{s1}, .., attrs_{st}$.
- Deletion of the some objects by explicitly sending delete messages for such objects.
- New messages are sent to objects of Cl , namely $ms'_{h1}, .., ms'_{hr}, ms'_{j1}, .., ms'_{jq}$.

Rewriting Rules Gouverning 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 communication pattern in figure 6, the general form of rewrite rules associated with this intra-component interaction model, takes the following form:

[5] $attrs_i$ is simplified notation of $atr_{i1} : val_{i1}, .., atr_{ik} : val_{ik}$.

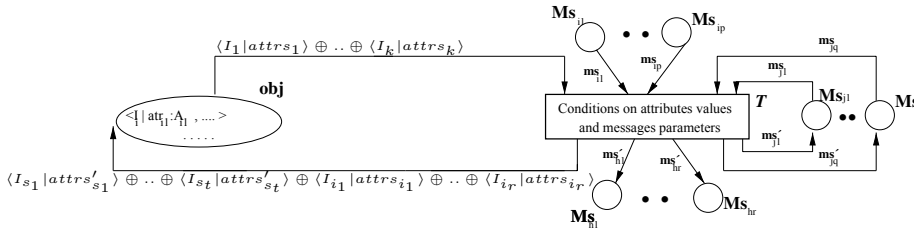

Fig. 4. The Intra-Class CO-Nets Interaction Model.

$$T : (Ms_{i1}, ms_{i1}) \otimes .. \otimes (Ms_{ip}, ms_{ip}) \otimes (Ms_{j1}, ms_{j1}) \otimes .. \otimes (Ms_{jq}, ms_{jq}) \otimes$$
$$(obj, \langle I_1 | attrs_1 \rangle \oplus .. \oplus \langle I_k | attrs_k \rangle) \Rightarrow (Ms_{h1}, ms'_{h1}) \otimes ... \otimes (Ms'_{hr}, ms'_{hr}) \otimes$$
$$(Ms_{j1}, ms'_{j1}) \otimes .. \otimes (Ms'_{jq}, ms'_{jq}) \otimes (obj, \langle I_{s1} | attrs'_{s1} \rangle \oplus .. \oplus \langle I_{st} | attrs'_{st} \oplus$$
$$\langle I_{i_1} | attrs_{i_1} \rangle \oplus .. \oplus \langle I_{i_r} | attrs_{i_r} \rangle)$$
$$if\ Conditions \text{ and } M(Ad_{Cl}) = \emptyset \text{ and } M(Dl_{Cl}) = \emptyset$$

Remark 2. The operator $\otimes$ is defined as a multiset union and allows for relating different places identifiers with their actual 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 performed at first: In other words, before performing this rewrite rules the marking of the Ad_{Cl} as well of the Dl_{Cl} places have to be empty. Finally, noting that the selection of just the *invoked parts* of object states, in this evolution pattern, is possible only due to the splitting /recombination axiom—that have to be performed in front and in accordance with each invoked state evolution.

Example 3. By applying this general form of rule, it is not difficult to generate the rules corresponding to the employee class (depicted in figure 2).

LEAVE [6]$:(Emp, \langle E|Lv : False\rangle) \otimes (Leav, lv(E, Dt, Pr))$
$\Rightarrow (Emp, \langle E|Lv : True\rangle) \otimes (Leav, lv(E, Dt, Pr))$[7]

RETURN $:(Emp, \langle E|Lv : True\rangle) \otimes (Leav, lv(E, Dt, Pr)) \otimes (Return, rt(E, Drt)) \Rightarrow$
$if\ (Dtr - Dt \leq Pr)\ then\ (Emp, \langle E|Lv : False\rangle)$
$else \quad (Emp, \langle E|Lv : False\rangle) \otimes (Punish, Pnsh(E, 1)$[8]$)$

PUNISH $:(Emp, \langle E|Sc : S\rangle) \otimes (Punish, pnsh(E, Dg))$
$\Rightarrow\ if(S + 1 > 3\ or\ Dg = 2)$
$then\ (DEL, del(E))\ else\ (Emp, \langle E|Sc : S + 1\rangle)$

[6] The label corresponds to the transition identifier.
[7] The leave message has not to be deleted in order to use it for controling the corresponding return.
[8] When the employee does not respect the Period of leave (s)he is punished.

CHG $:(Emp, \langle E|Ad : D\rangle) \otimes (Chg, chg(E, Nd) \Rightarrow (Emp, \langle E|Ad : Nd\rangle)$

RG_LEAV $: (Emp, \langle E|Lv : False\rangle)$
$\Rightarrow (Emp, \langle E|Lv : True\rangle) \otimes (Leav, lv(E, July, 45))\ if\ Date = July$

Remark 3. In this application we have the possibility for exhibiting intra-object concurrency. This is the case for the messages Lv, Sc and Chg that can be performed at the same time, when they are sent to the *same employee*.

5 CO-Nets: More Advanced Constructions

So far, we have presented only how the CO-Nets approach allows for conceiving independent classes. In what follows, we give how more complex systems can be constructed using advanced abstraction mechanisms, especially inheritance and interaction between classes. However, due to the space limitation only the simle inheritance case and the interaction pattern (without an illustrative example) would presented.

5.1 Simple Inheritance

Giving a (super) class *Cl* modeled as a CO-Net, for constructing a subclass that inherits the structure as well as the behaviour of the superclass *Cl* and exhibits new behaviour involving additional attributes, we propose the following straightforward conceptualization.

- Define the structure of the new subclass by introducing the new attributes and messages. Structurally, the new attributes identifiers with their value sorts and the message generators are described using the *extending* primitive in the OBJ notation.
- As *object place* for the subclass we use the *same* object place of the superclass; which means that such place should now contains the object states of the superclass as well as the object states of the subclass. This is semantically sound because the sort of this object place is a supersort for objects including more attributes.
- As previously described, the proper behaviour of the subclass is constructed by associating with each new message a corresponding place and constructing its behaviour (i.e. transitions) with respect to the communication model of figure 4 under the condition that at least one of the additional attributes has to be involved in such transitions.

Remark 4. Such conceptualization is only possible because of the splitting / recombination operation. Indeed this axiom permits to consider an object state of a subclass, denoted for instance as $\langle Id|attrs, attrs'\rangle$ with $attrs'$ the additional attributes (i.e. those proper to the subclass), to be also an object state of the superclass (i.e. $\langle Id|attrs\rangle$). Obviously, this allows a systematic inheritance of the

structure as well as the behaviour. The dynamic binding with polymorphism is systematically taken in this modeling. Indeed, when a message is sent to a hierarchy of classes we can know only after the firing of the associated transition to which class in the hierarchy the concerned object belong.

Example 4. As informally described, the titular employees have to be modeled as a subclass of the already modeled (probationer) employee class. So, in addition of being an employee (who can go on leave, etc), a titular employee can receive an increasing of his/her rung and can have some administrative responsibilities. More precisely, following the aforementioned step, we present hereafter the structure as well as the associated CO-Net modeling both classes.

```
obj titular is
 extending employee-with-recruitment .
 sort titular, ADV, ADM, TIT, FRM .
  op ⟨_|Rg : _, Adm : _⟩ : Id.emp  nat   string → Local-titular
 (* Local messages *)
  op Tit : Id.emp → TIT.
  op Adv : Id.emp → ADV .
  op Frm : Id.emp → FRM.
endo.
```

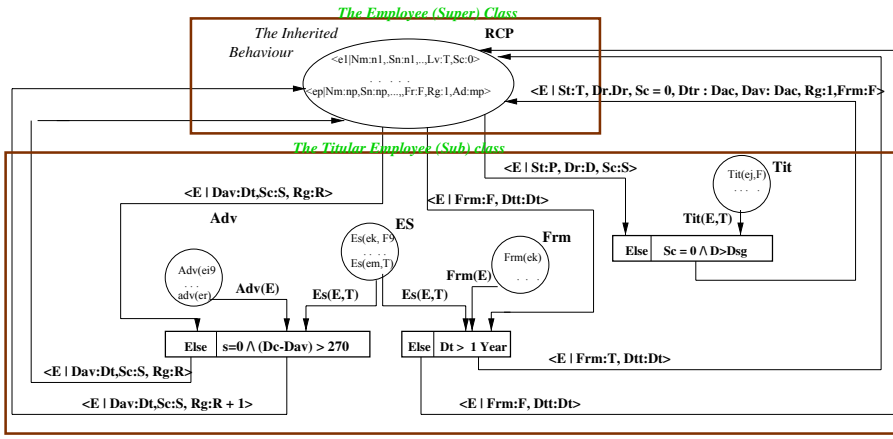

Fig. 5. The Titular Employees as a Subclass of the employee Class

5.2 Interaction between Classes

For interacting independent components, first, we should take into consideration the fact that the internal evolution of each component is ensured by the intra-component evolution pattern specified in figure 4. Second, we should ensure the

encapsulation property, that is to say, the internal part of each object state as well as the local messages have to be hidden from the outside; thus only those *explicitly* declared as observed attributes and external messages have to participate in such inter-component communication.

More precisely, as depicted in figure 6 this inter-component interaction may be made explicit as follows: The contact of some external parts of some objects states namely $\langle I_1|attrs\_ob_1\rangle..\langle I_k|attrs\_ob_k\rangle$, that may belong to different classes namely $C_1, ..., C_m$, with some external messages $ms_{i1}, .., ms_{ip}, ms_{j1}, .., ms_{jq}$ defined in these classes and under some conditions on attributes values and parameters messages results in the following:

- The messages $ms_{i1}, .., ms_{ip}, ms_{j1}, .., ms_{iq}$ vanish;
- The state change of some (external parts of) object states participating in the communication, namely $I_{s1}, .., I_{st}$. Change is symbolized by $attrs'_{s1}, .., attrs'_{st}$ instead of$attrs_{s1}$,.., $attrs_{st}$. The other objects components remain unchanged (i.e. there is no deletion of parts of objects states).
- New external messages (that may involve deletion/creation ones) are sent to objects of different classes, namely $ms'_{h1}, .., ms'_{hr}, ms'_{j1}, .., ms'_{jq}$.

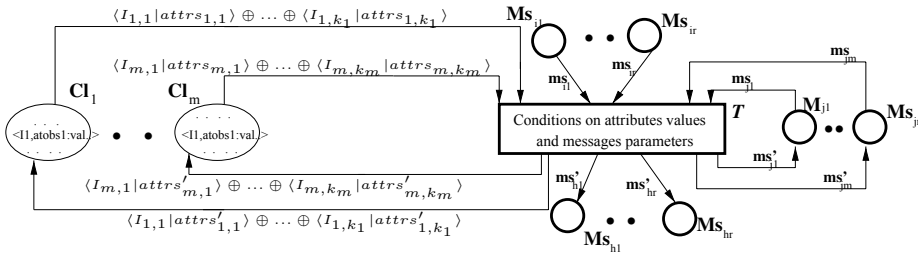

Fig. 6. The Interaction General Model Between classes

6 Conclusion

We proposed an object Petri nets based conceptual model for specifying and validating distributed information systems. The model called CO-Nets is a sound and complete combination of OO concepts and constructions in the ECATNets framework: An algebraic Petri net model mainly characterized by its capability of distinguishing between enabling conditions and destroyed tokens. The semantics of the CO-nets is expressed in rewriting logic allowing us to derive rapid-prototypes using concurrent rewriting in general and the MAUDE language more particularly. Some key features of the CO-Nets approach for specifying complex and distributed information systems are: First, the straightforward modeling of simple and multiple inheritance with the possibility of overriding. Second, the characterization of two communication patterns: an intra-component model for

evolving object states in a hierarchy of classes with the possibility of exhibiting intra-object as well as inter-object concurrency and an inter-component communication model for interacting different components that promotes concurrency and preserve the encapsulated features of each components.

The different aspects of the CO-Nets approach have been explained through a simplified but realistic case study dealing with typical staff management system. This case shows, among other, that the CO-Nets conceptual model, with its different abstractions mechanisms, is well suited for dealing with complex distributed information systems applications.

Our future work is to confirm the appropriateness of the CO-Nets for specifying complex distributed and cooperative information systems by leading more complex case studies. Also, we plan for formally integrating property-oriented verification models following mainly the work in [Lec96].

References

[Aou89] N. Aoumeur. Réalisation d'un Systeme de Gestion du Personnel de l'Université. Memoire d'Ingenieur, Institut d'Informatique, Université D'Oran, 1989.

[Aou99] N. Aoumeur. Towards an Object Petri Net Based Framework for Modelling and Validating Distributed Systems. To appear as Preprint, Fakultät für Informatik, Universität Magdeburg, 1999.

[BdC93] E. Battiston and F. de Cindio. Class Orientation and Inheritance in Modular Algebraic Nets. In *Proc. of IEEE International Conference on Systems and Cybernetics*, pages 717–723, Le Touquet, France, 1993.

[Bib97] Biberstein, O. and Buchs, D. and Guelfi, N. CO-OPN/2: A Concurrent Object-Oriented Formalism. In *Proc. of Second IFIP Conf. on Formal Methods for Open Object-Based Distributed Systems(FMOODS)*, pages 57–72. Chapman and Hall, March 1997.

[BMB93] M. Bettaz, M. Maouche, Soualmi, and S. Boukebeche. Protocol Specification using ECATNets. *Reséaux et Informatique Répartie*, 3(1):7–35, 1993.

[DJ90] J. Dershowitz and J.-P. Jouannaud. Rewrite Systems. *Handbook of Theoretical Computer Science*, 935(6):243–320, 1990.

[ECSD98] H.-D. Ehrich, C. Caleiro, A. Sernadas, and G. Denker. Logics for Specifying Concurrent Information Systems. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 6, pages 167–198. Kluwer Academic Publishers, Boston, 1998.

[EGS92] H.D. Ehrich, M Gogolla, and A. Sernadas. Objects and Their Specification. In M. Bidoit and C. Choppy, editors, *Proc. of 8th Workshop on Abstract Data*, volume 655 of *Lecture Notes in Computer Science*, pages 40–66. Springer-Verlag, 1992.

[EM85] H. Ehrig and B. Mahr. Foundamentals of algebraic specifications 1 : Equation and initial semantics. *EATCS Monographs on Theoretical Computer Science*, 21, 1985.

[FJLS96] B. Freitag, Cliff B. Jones, C. Lengauer, and H. Schek, editors. *Object Orientation with Parallelism and Persistence.* Kluwer Academic Publishers, 1996.

[GD93] J.A. Goguen and R. Diaconescu. Towards an Algebraic Semantics for the Object Paradigm. In *Proc. of 10th Workshop on Abstract Data types*, 1993.

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

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

[Lak95] Lakos, C. From Coloured Petri Nets to Object Petri nets. In *Proc. of 16th Application and Theory of Petri Nets*, volume 935 of *Lecture Notes in Computer Science*, pages 278–287. Springer-Verlag, 1995.

[Lec96] U. Lechner. Object Oriented Specification of Distributed Systems in the μ-Calculus and Maude. In J. Meseguer, editor, *Proc. of the First Inter. Workshop on Rewriting Logic*, volume 4. Electronic Notes in Theoretical Computer Science, 1996.

[Mes92] J. Meseguer. Conditional rewriting logic as a unified model for concurrency. volume 96 of *Theoretical Computer Science*, pages 73–155, Noordwijkerhout, Netherlands, 1992.

[Mes93] Meseguer, J. A Logical Theory of Concurrent Objects and its Realization in the Maude Language. *Research Directions in Object-Based Concurrency*, pages 314–390, 1993.

[Mil89] R. Milner, editor. *Communication and Concurrency*. Prentice Hall, 1989.

[Rei85] W. Reisig. *Petri Nets : An Introduction*. Springer-Verlag, 1985.

[Rei91] W. Reisig. Petri Nets and Abstract Data Types. *Theoretical Computer Science*, 80:1–30, 1991.