Co-nets : A Formal OO Framework for Specifying and Validating Distributed Information Systems

Nasreddine Aoumeur[1] Gunter Saake

ITI, FIN, Otto-von-Guericke-Universität Magdeburg
Postfach 4120, 39016 Magdeburg, Germany
E-mail: {aoumeur|saake}@iti.cs.uni-magdeburg.de
Phone: ++49/391/67-{18066|12994|18800} Fax: ++49/391/67-12020

January 2000

[1]This work is supported by a DAAD scholarship.

Abstract

This paper formally defines a new object oriented (OO) Petri net-based approach for specifying and rapid-prototyping distributed (information) systems—regarded as a community of interacting, through explicit interfaces, and concurrently existing objects. The proposed model, referred to as CO-Nets, is based on a complete integration of OO concepts into an appropriate variant of algebraic Petri nets named ECATNets. The CO-Nets behaviour is interpreted in rewriting logic allowing validation by rapid-prototyping using rewrite techniques. Particular to the CO-Nets approach are mainly: the modeling of a class rather as a module with a hidden part (i.e. structure and behaviour) and an observed imported/exported part; the capability of incrementally constructing complex components, as a hierarchy of modules, through simple and multiple inheritance with possibility of overriding, dynamic binding and associated polymorphism. Each component behaves with respect to an appropriate intra-component evolution pattern that supports intra- as well as inter-object concurrency. For interacting such components, an appropriate inter-component interaction pattern is proposed that enhances concurrency and preserves the encapsulated features of each component.

Contents

Chapter 1

Introduction

The development of large distributed (information) systems is a very complex task; where in addition to the complexity of the informational computation, difficulties arise in relation with concurrency, optimal distribution and interoperabilities of different system components. Aiming to cope with these difficulties, a global approach that supports early phases of *specification* and *validation* become more than necessary. Indeed, with the specification phase a precise and abstract description of the system should be derived, while the validation permits to ensure that crucial properties of the modeled system can be guaranteed before any efficient, secure and ergonomic implementation.

Although the perception of such systems as a *community of interacting objects* is widely accepted in the analysis as well as in the implementation phases, the establishment of the *theoretical underpinning* of the object paradigm —as a necessary condition to carry the crucial specification/validation phases through successfully—and particularly in the presence of *concurrency* is still vivid area of research [FJLS96, ECSD98]. Pioneer proposals within this scope are mainly based on different extensions of the algebraic setting, and they include particularly: Hidden Sorted Algebra [GD93], D-Oids [AZ94], CO-algebra [Rei94, Jac96], ADT-Process [EGS92], and rewriting logic [Mes92]. Recent research also advocates, among other, object oriented Petri net-based formalisms including especially: OPNet [Lak95, Lak96a], CO-OPN/2 [BBG97], CLOWN [BCdC95], COP [SB94], SIM [Ver93] and [BB91].

The object model we intend to propose in this paper fits in the later class, with the distinguished feature that its main conceptual ideas stem from the algebraic foundations to the object paradigm in general and rewriting logic particularly. More precisely, we aim to present and formally define an (yet another) OO Petri net-based approach for specifying and validating distributed systems with more adequacy to the modeling of distributed information systems [AS99a, AS99b]. Referred to as CO-NETS, the model is based on a complete integration of object oriented concepts and constructions into an appropriate variant of algebraic Petri nets named ECATNets [BM92, BMSB93]—a combination of abstract data types and Petri nets distinguished by its semantics interpreted in rewriting logic. The CO-NETS approach is also semantically interpreted in rewriting logic.

In some detail, key features of CO-NETS in modeling distributed (information) systems are the following:

- A clean separation between *data* —specified algebraically[1] and used for describing object attributes (values and identifiers) and messages— and *objects* as indivisible units of structure and behaviour. Moreover, we regard a class rather as a *module* —in the spirit of [EM90] (at the syntactical level)—with a hidden part including structure as well as behaviour, and an observed part as interface for interacting with the environment and other modules.

- An incremental construction of components, as a hierarchy of modules[2], through simple and multiple inheritance (with redefinition, associated polymorphism and dynamic binding). Such components behave with respect to an appropriate *intra-component* evolution pattern that naturally supports intra- as well as inter-object concurrency (i.e. without suffering from the inheritance anomaly problem).

- 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.

- By interpreting Co-nets behaviour in rewriting logic, rapid-prototyping may be generated using rewrite techniques and current implementation of the Maude language [Mes93a] particularly.

The rest of this paper is organized as follows: the second section recalls some notions about rewriting logic and presents in some detail the ECATNets model. Aiming to gradually presenting the Co-nets approach, the third section first introduces (the syntactical aspects of) the Ob-nets model: an object based[3] Petri nets model defined as a natural extension of the ECATNets for describing the notion of object, template[4] and classes as modules. In the fourth section Ob-nets semantics is detailed. The five section defines the Co-nets model as an extension of Ob-nets for dealing with the main advanced OO structuring mechanisms (i.e. inheritance, interaction and objects composition). In the sixth section we situate our model with respect to the recent classification presented in [BBC$^+$97]. The last section presents some concluding remarks.

[1] Using Ordered Sorted Algebra(OSA).

[2] Throughout the rest of the paper, we use indifferently module or class.

[3] Following the classification of P. Wegner [Weg90], where in object based systems there is no inheritance and no interaction.

[4] We follow here the approach presented in [EGS92] that makes a clean separation between templates as description of common structure and behaviour of some kind of objects and a class as a time varying collection of objects (and messages) instances.

Chapter 2

The ECATNets model : An overview

The ECATNets [BMSB93] model has been conceived around three formalisms; the first two ones constitute a net/data model and serve for describing the structural aspects of the modeled system, while the third one allows for defining the semantics (i.e. the behaviour) of the system expressed in rewriting logic.

2.1 Rewriting logic

Rewriting logic, considered nowadays as one of the widely accepted unified model of concurrency [Mes98], has been introduced by J. Meseguer in [Mes90], first, by exploiting the intrinsic *concurrency* in terms rewriting [GKM87], and, second, by observing the inadequacy of interpreting rewrite rules as (oriented) equations where dealing with non Platonic (i.e. reactive) systems. While rewrite rules have the usual form, they are rather interpreted in rewriting logic as a *change* in concurrent systems. To be more precise, we present some definitions borrowed from [Mes92]:

- A *(labelled) rewrite theory* $\mathcal{R}$ is a 4-tuple $\mathcal{R}= (\Sigma, E, L, R)$ where Σ is a ranked alphabet of functions symbols, E is a set of Σ-equations, L is a set called the set of *labels,* and R is a set of pairs $R \subseteq L \times (T_{\Sigma,E}(X)^2)^+$ whose first component is a label, and whose second component is a nonempty sequence of pairs of E-equivalence classes of terms, with $X = \{x_1, .., x_n\}$ a countably infinite set of variables. Elements of R are called *rewrite rules*. A rewrite rule $(r, [t], [t'])([u_1], [v_1]). \ . \ .([u_k], [v_k])$ is denoted as

  $$r : [t] \Rightarrow [t'] \ if \ [u_1] \Rightarrow [v_1] \wedge ... \wedge [u_k] \Rightarrow [v_k].$$

- Given a rewrite theory $\mathcal{R}$, we say that $\mathcal{R}$ *entails* a sequent $r : [t] \Rightarrow [t']$ and write $\mathcal{R} \vdash [t] \Rightarrow [t']$ if and only if $[t] \Rightarrow [t']$ can be obtained by finite application of the following *rules of deduction*:

  1. **Reflexivity** : For each [t] $\in T_{\Sigma,E}(X)$, $\qquad \frac{}{[t]\Rightarrow[t]}$
  2. **Congruence** : For each f $\in \Sigma_n$, n $\in$ N, $\qquad \frac{[t_1]\Rightarrow[t_1']. \ . \ .[t_n]\Rightarrow[t_n']}{f(t_1,...,t_n)]\Rightarrow[f(t_1',...,t_n')]}$

3. **Replacement**[1]: For each rule $r : [t(x_1, .., x_n)] \Rightarrow [t'(x_1, ..., x_n)]$ in R,

$$\frac{[w_1]\Rightarrow[w'_1]...[w_n]\Rightarrow[w'_n]}{[t(\bar{w}/\bar{x})]\Rightarrow[t(\bar{w'}/\bar{x})]}$$

4. **Transitivity** : $\frac{[t_1]\Rightarrow[t_2] \quad [t_2]\Rightarrow[t_3]}{[t_1]\Rightarrow[t_3]}$

- Given a rewrite theory $\mathcal{R}= (\Sigma, E, L, R)$, a (Σ, E)-sequent $[t] \Rightarrow [t']$ is called a *concurrent $\mathcal{R}-$rewrite* (or just a *rewrite*) iff it can be derived from $\mathcal{R}$ by finite application of the rules 1-4. And the sequent is called *one-step concurrent $\mathcal{R}$-rewrite* iff it can be derived from $\mathcal{R}$ by finite application of the rules 1-3 with at least one application of rule 3.

2.2 The ECATNets model

From syntactical viewpoint, the ECATNets are a form of algebraic Petri nets [Rei91], where places are annotated by multisets of ground algebraic terms[2] for an algebraic signature [EM85]. Input arcs are annotated by two multisets of terms: the enabling tokens IC (for input condition), and the destroyed tokens (DT). Output arcs are annotated by created tokens multisets (CT). Finally, Boolean expressions can be associated with transitions denoted by TC (i.e. transition condition). The generic form of ECATNets [BM92] is depicted in figure 2.1(a).

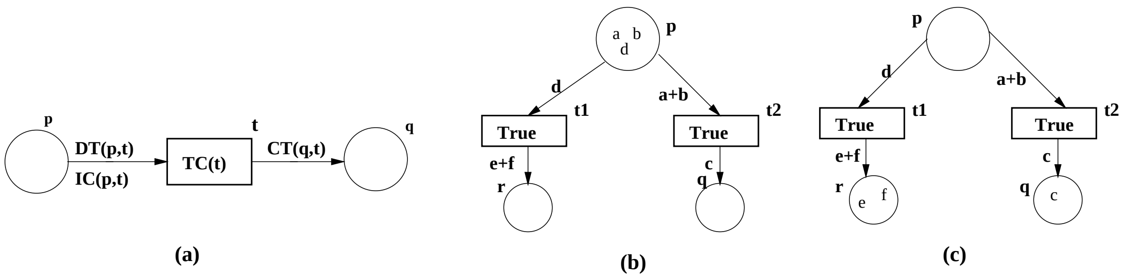

Figure 2.1: The Generic ECATNets and illustrative example

The behaviour of a given ECATNets is defined as follows. For firing a transition t, $IC(p, t)$ must be included into $M(p)$ (for all input places p) and the condition $TC(t)$ has to be evaluated to true. After firing t, there is a deletion of $DT(p, t)$ from each input place and an addition of $CT(t, q)$ to output places q of t. The *formal semantics* of ECATNets is expressed in rewriting logic [Mes92], where the effect of transitions is captured by rewrites rules. The form of such rewrite rule depends essentially of the relation between IC and DT, where three cases are distinguished in [BM92]. We recall the corresponding rewrite rules for these cases after giving some necessary notations.

As described above, the content of each place is a multiset of closed terms w.r.t. a given signature. We denote by $\oplus$ the union operation on multisets[3]. To capture rewrite

[1] In this rule the term $[t(\bar{w} \ / \ \bar{x})]$ denotes the simultaneous substitution of w_i for x_i in t with $\bar{x}$ representing $(x_1, .., x_n)$.

[2] Denoted as $M(p)$ for each place p.

[3] In figures, $\oplus$ is denoted as $+$

rules and ECATNets states, we use multisets where the elements are pairs of the form: *(place_name, multisets of terms).* The union operation on this multiset is denoted by $\otimes$. In order to exhibit full concurrency, a distribution law of $\otimes$ over $\oplus$ is defined as an axiom used as 'splitting' from left to right and as 'recombination' from right to left: $(p, mt_1 \oplus mt_2) = (p, mt_1) \otimes (p, mt_2)$, where mt_1 and mt_2 are multisets of terms generated by $\oplus$. This axiom is referred to as *decomposition* axiom.

Given a transition t, let $\{p_1, .., p_n\}$ its input places and $\{q_1, .., q_m\}$ its output places. We distinguish four cases:

1. $IC(p,t) = DT(p,t)$, for each $p \in \{p_1, ..., p_n\}$. In this case the rule takes the following form:
   $t : (p_1, IC(p_1, t)) \otimes .. \otimes (p_n, IC(p_n, t))$
   $\Rightarrow (q_1, CT(q_1, t)) \otimes .. \otimes (q_m, CT(q_m, t)$ if $TC(t)$.
2. $IC(p,t) \cap DT(p,t) = \emptyset$, for each $p \in \{\ p_1, ..., p_n\}$. In this case the rule takes the form:
   $t : (p_1, IC(p_1, t)) \otimes (p_1, DT(p_1, t)) \otimes .. \otimes (p_n, IC(p_n, t)) \otimes (p_n, DT(p_n, t)$
   $\Rightarrow (p_1, IC(p_1, t)) \otimes (q_1, CT(q_1, t)) \otimes .. \otimes (q_m, IC(q_m, t)) \otimes (q_m, CT(q_m, t)$ if $TC(t)$.
3. $IC(p,t) \cap DT(p,t) \neq \emptyset$, for at least one $p \in \{p_1, .., p_n\}$. It has been shown in [BM92] that this case could be brought to the two shown cases.
4. The first three cases concern the CATNet version [BM92]. For the E(xtended) CATNets, two additional notations have been introduced [BMSB93]. Both notations concern the Input Condition. In the first one IC takes the form: $IC(p,t) =^{\sim}$*multiset of terms* where $^{\sim}$ is a new symbol. The enabling condition in this case means that $IC(p,t)$ has not to be included in $M(p)$. Using the second notation we may have $IC(p,t) = \emptyset$ which simply means that the associated transition may be fired only when its input places are empty. The associated rewrite rules are detailed in [BMSB93].

Example 2.2.1 *For the net depicted in figure 2.1(b), by applying the first case, we obtain the following rules for the transitions* $t1$ *and* $t2$: $t1 : (p, d) \rightarrow (r, e \oplus f)$ $t2 : (p, a \oplus b) \rightarrow (q, c)$
The initial state, represented in Figure 2.1(b), is equivalent to: $(p, a \oplus b \oplus d) \otimes (q, \emptyset) \otimes (r, \emptyset)$.
Construction of the next state as depicted in Figure 2.1(c) may be obtained by the following rewrite sequence. By applying the decomposition axiom, the following equivalent state may be obtained: $\Rightarrow (p, a \oplus b) \otimes (p, d) \otimes (q, \emptyset) \otimes (r, \emptyset)$
By concurrently applying the rewrite rules associated with $t1$ *and* $t2$ *to this state, we obtain the state represented in figure 2.1(c)* $\Rightarrow (q, c) \otimes (r, e \oplus f) \otimes (q, \emptyset) \otimes (r, \emptyset)$

Chapter 3

OB-NETS: Templates and Classes Specification

As a first step towards the CO-NETS approach, the present section describes a formal framework named OB-NETS, for specifying independent templates and classes regarded as modules without dealing with advanced OO abstraction mechanisms. Firstly, the notion of template signature is defined. For introducing the notions of templates and classes specification, we then define respectively the notion of an OB-NET and a marked OB-NET associated with a given template signature.

On the other hand, following most of the aforementioned algebraic-based approaches to the object paradigm, we assume that the *data level*—that serves for defining the object identities, attributes identifiers and values as well as the actions (or messages[1])— is algebraically specified elsewhere; using semantically ordered sorted algebra (OSA) [GD94] and syntactically the OBJ language [Gog92]. Moreover, for easing the presentation we assume the following (data) sorts are given: *bool* for the boolean, OId for capturing the object identities, AId for the object identifiers and $Value$ as a generic sort for the attribute values.

3.1 Template Signature

The template signature allows for describing the form of object states as well as the message forms that have to act on such states. In our approach, the structure of object states that we adopt is mainly inspired by the —term as tuple— structure proposed in the MAUDE language [Mes93a], but with some additional features especially required when dealing with cooperative and distributed information systems in the object oriented setting [PS98]. Particularly, by introducing an appropriate *splitting/merging* axiom, we 'liberate' the object state –as tuple– from its *indivisibility*[2], allowing mainly the exhibition of *intra-object* concurrency. The second contribution, that is intrinsically related to the first one, is the introduction of the notion of observed attributes allowing us to easily capturing the notion of class as module.

[1] Throughout this paper we use the term of message instead of action or event to make in evidence distributive aspects of the approach.

[2] Recalling that, due to the *indivisibility* of the object state as tuple, in MAUDE the intra-object concurrency is not possible.

Roughly speaking, the object state is conceived as a term of the form $< Id|atr_1 : val_1, .., atr_k : val_k, atrbs_1 : valbs_1, .., atrbs_{k'} : valbs_{k'} >$, where Id is of sort OId and captures the identity of the object; the attributes (identifiers) $atr_1, .., atr_k$ are assumed to be the *local* (i.e. hidden from the outside) part of the object; their respective actual values are $val_1, .., val_k$. The rest part of the structure of an object i.e. $atrbs_1, .., atrbs_{k'}$ is assumed to be *observed* by other classes; in this observed part, $valbs_1, .., atrbs_{k'}$ denote the actual values. Both local and observed attribute identifiers are assumed to be of AId sort and their respective value of $Value$ sort. Besides this notion of observed attributes—motivated mainly by real-world information system applications and already materialized, for instance, in the ALBERT language [DB95]—, we introduce an appropriate 'axiom', that will be fully exploited when dealing the semantics of the model, allowing for *splitting* and/or *merging* the object state out of necessity. Principally, this axiom that takes the following form: $< Id|attrs_1$[3]$, attrs_2 >=< Id|attrs_1 > \oplus < Id|attrs_2 >$, provides us for naturally exhibiting intra-object concurrency, for straightforwardly capturing the inheritance and finally for interacting different modules without violating their encapsulated features.

The actions or messages are conceived as terms—that consists of message's name, the identifiers of the objects the message is addressed to, and, possibly, parameters. Moreover, in the OB-Nets approach we distinguish between *local* message encapsulated from the outside and used for evolving the object states of the concerned class, the *imported* one used as services provided by other classes and *exported* messages being services produced by the concerned class and being used by other classes.

All for all, the template signature that we adopt may be described in an OBJ notation[4] as given in figure 3.1, and it is formally defined as follows:

Definition 3.1.1 *A template signature is defined as a pair* $(S_D \cup S_O \cup S_{Msg}, Ms_l \cup Ms_o)$ *with:*

- S_D *is a set of (data) sorts with at least* $\{bool, OId, AId, Value\} \subset S_D$. *For allowing object-attribute values,* OId *is defined as a subsort of* $Value$ *(i.e.* $OId < Value$*).*
- S_O *is a set of (object) sorts, but we assume here it is restricted to one object sort denoted by* $obj(ect)$. *With each object sort is associated an operation indexed by* $OId \times (AId \times Value)^+ \times obj$ *that captures its attributes. More precisely, by assuming that* S_D *contains the following set of attributes sorts:* $\{s\_atr_1, ..., s\_atr_k, s\_atrbs_1, ..., s\_atrbs_{k'}\}$ *denoting respectively sorts associated with attributes identifiers, namely,* $atr_1, .., atr_k$, $atrbs_1, .., atrbs_{k'}$ *(defined as constants of sort* AId*), the form of such operation is:* $< \_|atr_1 : \_, ..., atr_k : \_, atrbs_1 : \_, ..atrbs_{k'} : \_ >$[5] *indexed by* $Id.obj \times s\_atr_1 \times ... \times s\_atr_k \times s\_atrbs_1 \times .... \times s\_atrbs_{k'} \times obj$*; with* $Id.obj < OId$ *denoting the proper identifiers sorts of* obj*.*
- S_{Msg} *is set of (message generator) sorts. To be explicit, we assume that* S_{Msg} *contains the following set of message sorts* $\{Mes_{l1}, ..., Mes_{l1}, Mes_{i1}, ..., Mes_{ii}, Mes_{e1}, .., Mes_{ee}\}$.
- *The message operations in the set* $Ms_o \cup Ms_l$ *are defined as* $S_D^+ \times S_{Msg}$*–indexed op-*

[3]With $attrs_i$ denotes $atr_{i1}, .., atr_{ik}$.

[4]With borrowing some ideas from the notion of configuration defined for the MAUDE language.

[5]As detailed in the corresponding OBJ description, the operator '$\_ : \_$' is a binary indexed by $AId \times Value$ associating with each attribute identifier its actual value, while the operator '$\_, \_$' relates different attribute-value pairs in an no materialized way (i.e. it is declared as associative and commutative).

erations, where S_D^+ containing at least an argument of sort OId[6]. With each message sort $Mes_i \in S_{Msg}$ is associated exactly one message operation from $Ms_o \cup Ms_l$ that we denote by ms_i. Ms_l denote the set of local messages that have to include as parameters exactly one of sort OId, while Ms_o denote the 'external' messages (that can be split into imported and exported) and thereby at least two of their arguments should be of sorts OId.

```
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,...,Mesee < 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.
```

Figure 3.1: The template signature specified in an OBJ notation

Example 3.1.2 *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 suppliers-number(shortly, Nc and Ns), name(N), birth-day(Bd), address(Ad). On the other hand, products are*

[6]This allow to make a clear difference between an operation from the data level and a message that should be sent to or received by at least one object.

characterized by product-number(Np), price-unit(Pu) and the 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, the 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[7]*, the messages supply(sp) and order or command(cm) will be regarded as local messages and by consequence also the attributes. Moreover, for explicitly exhibiting the notion of intra-object concurrency, we introduce another method that allows for increasingly changing the unit-price denoted as (chg). According to our general form of classes, Prd 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 (* variables to be used in the net *)  .
endo .
```

3.2 Template Specification

Given a template signature denoted by TS that capture the structural aspects of a (OO information) system, its behaviour is constructed by associating an (OB-) *Net* with this signature—leading to the notion of template specification that we denote by $TSP =< TS, Net >$. Informally speaking, the net associated with a given template signature is constructed as follows. The places of the net are precisely defined by associating with each message generator, namely ms_i, one place named 'message' place and denoted (and typed by) Mes_i. Also, with the object state sort *obj* an (object) place is associated. OB-NETS transitions which may include conditions should reflect the effect of messages on the object states (i.e. the methods body) to which they are addressed.

Before we present the formal definition of an OB-NET reflecting this intuitive construction, some preliminary notations and explanations are necessary.

Note 3.2.1

1. *Subsequently, we denote by $T_{obj}(X)$ (resp. $T_{Msg}(X)$) the set of terms of S_O (resp. S_{Msg}) sort(s). By $T_{obj}(\emptyset)$ (resp. $T_{Msg}(\emptyset)$) we denote their associated objects (resp. messages) ground terms.*

2. *$MT_{obj}(X)$ (resp. $MT_{Msg}(X)$) will denote the multiset of terms defined over $T_{obj}(X)$ (resp. $T_{Msg}(X)$) and generated by the union operation denoted by $\oplus$. Moreover, like in [BMSB93], we denote by $MT_{obj}(X)^+$ (resp. $MT_{Msg}(X)^+$) the multiset elements of $MT_{obj}(X)$ (resp. $MT_{Msg}(X)$) preceded by the unary operator $\tilde{}$.*

[7]We will return to these classes when dealing with interaction between classes.

3. *For capturing the* OB-NETS *state and the associated rewrite rules, another multiset is defined w.r.t. a union operator over $P \times MT_{obj}(X)^+ \cup MT_{Msg}(X)^+$ that we denote by $\otimes$; where P denote the place identifiers that are from $\{obj, Ms_o \cup Ms_l\}$. Henceforth, with each place is associated a multiset reflecting its actual marking.*

4. *These different (multiset) constructions can be explicitly algebraically defined as an extension of our class-description specification. In this intuitive specification, the two constraint sorts ensure that the marking (terms) sort is compatible with its corresponding place.*

```
obj OB-Net-Spec is .
  extending class-description .
  subsort obj  Mes_l1  ...  Mes_ee  < Place.
  subsort Place < Place_Marking .
  sort Place_Identifier .
  op obj  Mes_l1  ...  Mes_ee  :  → Place_Identifier .
  op _⊕_:  Place Place-Marking → Place-Marking.  [assoc.  comm.  id:]
  op (_,_):  Place-Identifier Place-Marking → Marking.
  op _⊗_:  Marking Marking → Marking.
  vars p :  Place_Identifier
  vars T :  Place
  scr (p,T) :  Undefined if p=obj ∧ T:{ Ms_l1,..,Ms_ee} .
  scr (p,T) :  Undefined if p∈ {Ms_l1,..,Ms_ee} ∧ T:obj .
  eq (p,T1 ⊕ T2) = (p,T1) ⊗ (p,T2)  .
endo .
```

Definition 3.2.2 *Given a template signature as previously defined, an* OB-NET *is a structure (P, T, IC, DT, CT, TC) where:*

- *P is a set of places precisely named and typed as follows: $P = \{obj, Mes_{l1}, .., Mes_{ee}\}$;*
- *T is a set of transitions such that $P \cap T = \emptyset$;*
- *$IC : (P \times T) \to MT_{obj}(X)^+ \cup MT_{Msg}(X)^+$ is a partial function s.t. for every $(p,t) \in Dom(IC)$, if $p = obj$ then $IC(p,t) \in MT_{obj}(X)^+$ else if $p = Mes_i$ then $IC(p,t) \in MT_{Mes_i}(X)^+$ for each $i \in \{1, .., m\}$;*
- *$DT : (P \times T) \to MT_{obj}(X) \cup MT_{Msg}(X)$ is a partial function s.t. for every $(p,t) \in Dom(DT)$, if $p = obj$ then $DT(p,t) \in MT_{obj}(X)$ else if $p = Mes_i$ then $DT(p,t) \in MT_{Mes_i}(X)$ for each $i \in \{1, .., m\}$;*
- *$CT : (P \times T) \to MT_{obj}(X) \cup MT_{Msg}(X)$ is a partial function s.t. for every $(q,t) \in Dom(CT)$, if $q = obj$ then $CT(q,t) \in MT_{obj}(X)$ else if $p = Mes_i$ then $CT(q,t) \in MT_{Mes_i}(X)$ for each $i \in \{1, .., m\}$;*
- *$TC : T \to T_{bool}(X)$. is a function s.t. for every $t \in T$, $TC(t) \in T_{bool}(X)$ where $X(t)$ is the set of variables occurring in $IC(p,t)$ when defined, $DT(p,t)$ and $CT(q,t)$ for every $p, q \in P$;*

The concept of a class, as a population of object and message instances, generated from a given template specification, is captured by the notion of a *marked* OB-NET.

Definition 3.2.3 *A marked* OB-NET *is an* OB-NET *with a function $M : P \to MT_{obj}(\emptyset) \cup MT_{Msg}(\emptyset)$ s.t. if $p = obj$, then $M(obj) \in MT_{obj}(\emptyset)$ (i.e. the obj place contains the objects states of the class) else $M(Mes_i) \in MT_{Mes_i}(\emptyset)$ (i.e. such places contain message instances waiting for their execution or firing using the corresponding transitions).*

Examples of (marked) Ob-nets will be given after we define the semantics that should be respected when constructing such nets.

Chapter 4

OB-NETS : Semantical Aspects

Given an OB-NET (resp. a marked OB-NET) describing a template (resp. a class), its semantics should provide us, on the one hand, the *permissible* situations (i.e. the permissible object states and behaviour) that an object society may be in. On the other hand, it should allow us for (formally) deducing, in a true concurrent way, from an initial society any other permissible reachable one. By permissible community is mainly meant the respect of the uniqueness of object identities and the encapsulation property—during the state evolution of such society.

4.1 Objects creation and deletion

Viewing a marked OB-NET as a society of objects means that each object has its proper identity that persists through a change. In order to ensure the uniqueness of object identities, we propose the following conceptualization:

1. Associate with each marked OB-NET modeling a class denoted Cl a new place of sort $Id.obj$ ($< OId$) containing *actual object identifiers* of the objects in the object place (obj).

2. For the creation of new objects, we introduce a new message sort denoted Ad_{Cl} and an associated symbol operation (i.e. creation message) denoted ad_{Cl} and indexed by $Id.obj \times Ad_{Cl}$.

3. Each object state creation should be performed through the net depicted in the left hand side of figure 4.1. The notation $\sim$ capture exactly the intended behaviour (i.e. the identifier Id should not already be in the place $Id.obj$). After firing this transition, there is an addition of the new identifier Id to the place $Id.obj$ and creation of a new object namely $< Id|atr_1 : in_1, ..., atr_k : in_k >$, where $in_1, ..., in_k$ are optional initial attributes values.

Remark 4.1.1

- *This object creation model is very general and can be adapted to specific cases. For instance, when restriction conditions should be associated with initial values, the signature of the creation message should be modified to:* $ad_{Cl} : Id.obj\ s\_atr_1 .. s\_atr_k \rightarrow ad_{Cl}$ *which allow to associate conditions with the transition. The object deletion is modeled following the same reasoning as depicted in the right hand of figure 4.1.*

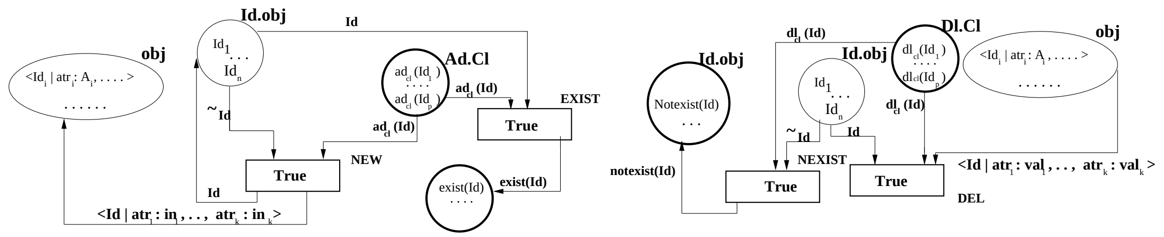

Figure 4.1: Objects Creation and Deletion Using OB-NETS

- *It is worth noting that, due to the splitting / merging axiom, we should avoid deletion of just a part of a given object state. For this aim, we require that before performing any deletion of an object I (i.e. firing the transition DEL) we should 'gather' all eventually split part. We achieve that by applying the above axiom rather as a rewrite rule:*
  $< I|attrs_1 > \oplus < I|attrs_2 > \Rightarrow < I|attrs_1, attrs_2 >$
  until finding the corresponding normal form (that always exist in this case); this ensure that the whole object state will be deleted after.

4.2 Evolution of Object States

For the evolution of object states in a given class, an appropriate general pattern for the form of 'local' transitions should be respected in order to: ensure the encapsulation property—in the sense that no object state or message from other classes participates in the such state evolution—, to preserve the object identities uniqueness and also to exhibit a maximal of (intra- and inter-object) concurrency.

Following such guidelines, and before we present this general pattern, let us start with an interesting particular case inspired by the *general event communication* proposed for MAUDEs rewrite rules[1] (with respect to a single class). This evolution schema is depicted in figure **??**, and it can be intuitively explained as follows. The contact of the only relevant parts[2] of some object states of a given class Cl, —namely $< I_1|attrs_1 >, .., < I_k|attrs_k >$— with some messages $ms_{i_1}, ..., ms_{i_p}, ms_{j_1}, ..., ms_{j_q}$—declared as *local* or *imported* in this class— and under some conditions on the invoked attributes and message parameters results in the following effect:

- the messages $ms_{i_1}, .., ms_{i_p}, ms_{j_1}, .., ms_{j_q}$ vanish;
- state change of some (parts of) objects participating in the communication, namely $I_{s_1}, .., I_{s_t}$. Such change is symbolized by $attrs'_{s_1}, .., attrs'_{s_t}$ instead of $attrs_{s_1}, .., attrs_{s_t}$. The other (unchanged part of) object states are denoted by $attrs_{i_1}, .., attrs_{i_r}$ so that $\{i_1, ...i_r\} \cup \{s_1, .., s_t\} = \{1, .., k\}$[3].

[1] Recalling that this general pattern of rewrite rules takes the form: $M_1...M_n < O_1 : C_1|attrs_1 > ... < O_m : C_m|attrs_m > \Rightarrow < O_{i1} : C'_{i1}|attrs'_{i1} > ... < O_{ik} : C'_{ik}|attrs'_{ik} >< Q_1 : D_1|attrs''_1 > ... < Q_p : D_p|attrs''_p > M'_1...M'_q$ if C.

[2] Such selection of only the concerned attributes is possible due to the 'splitting/merging' axiom that have to be performed in front of each object evolution depending on the invoked method.

[3] In other words, there is no *implicit* creation or deletion of (part) of object states— that may leads to inconsistency w.r.t. the above described creation/deletion schema.

- new messages (local or exported) are sent to object of the class Cl, namely $ms'_{h_1}, .., ms'_{h_r}, ms'_{j_1}, .., ms'_{j_q}$, which may include (explicit) deletion and/or creation of some objects.

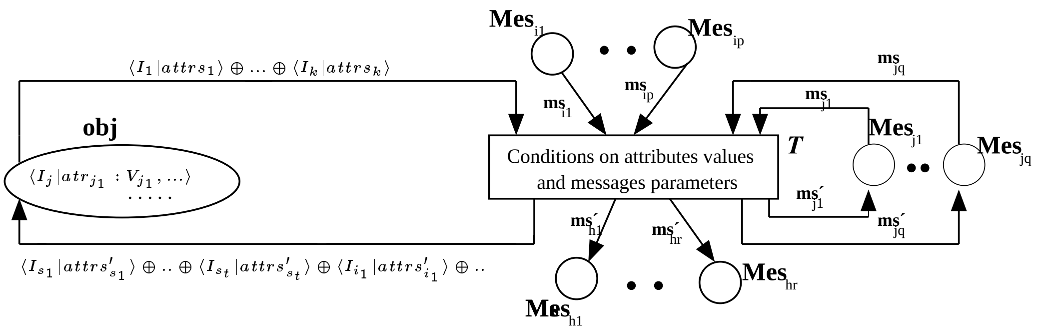

Figure 4.2: The Intra-Component Evolution Pattern

4.2.1 The general object state evolution pattern

The most abstract permissible communication pattern that can be characterized for evolving objects in a given class Cl is obtained by keeping, in the particular case above, the different IC, DT and CT associated with different arcs without any specific instantiation. Following that, this general evolution form is depicted in figure 4.3, where according to the OB-Nets definition:

- $IC_{obj} \subset MT_{obj}(X)^+$, while DT_{obj} and $CT_{obj} \subset MT_{obj}(X)$
- $\{Ms_{i1}, .., Ms_{ip}, Ms_{j1}, .., Ms_{jq}, Ms_{h1}, .., Ms_{hr}\} \subset \{Mes_{i1}, .., Mes_{ee}\}$
- $IC_{msi} \subset MT_{Mesi}(X)^+$, while DT_{msi} and $CT_{msi} \subset MT_{Mesi}(X)$ for $i \in \{i_1, .., i_p, j_1, .., j_q, h_1, .., h_r\}$

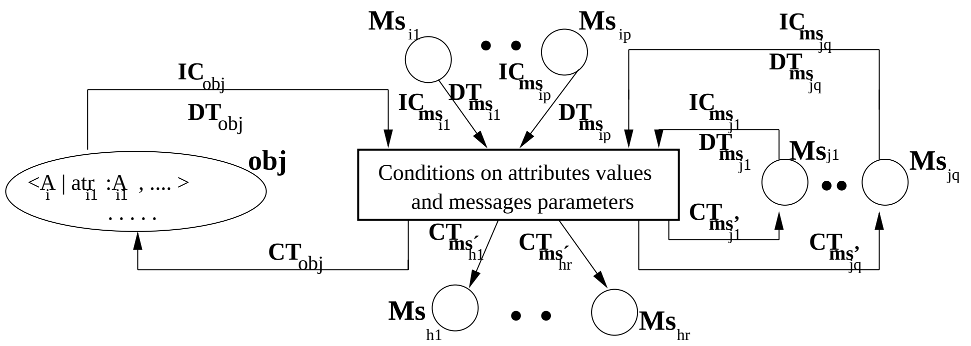

Figure 4.3: The General Intra-Component Evolution Pattern in OB-NETS

Remark 4.2.1 *With respect to this general evolution form, the particular case depicted is figure* **??** *then corresponds to the case where* $IC_{obj} = DT_{obj}$ *and* $IC_{Msi} = DT_{Msi}$— *with* $i \in \{i_1, .., i_p,\ j_1, .., j_q, h_1, .., h_r\}$. *As we have shown in [AS99b], this general form is very useful when dealing with distributed information systems. Typical example is the modification of object states only if some other states and/or messages don't already exist* *(which necessitate the use of the operator $\sim$). Also, by stating DT to $\emptyset$, which means that no deletion is performed on object and/or messages participating in the evolution, we result in a suitable model for querying/viewing the database associated with an information system.*

4.3 Rewriting rules governing OB-NETS behaviour

Like the ECATNets behaviour, each OB-NETS transition is captured by an appropriate rewriting rule interpreted in rewrite logic. More precisely and w.r.t. the communication pattern in figure 4.3, the form of OB-NETS rewrite rules depends like in the ECATNets on the relation between IC and DT and on the presence or not of the $\sim$ notation.

1. $IC(p,t) = DT(p,t), p \in \{obj, Ms_{i1}, .., Ms_{ip}, Ms_{j1}, .., Ms_{jq}, Ms_{h1}, .., Ms_{hq}, Ad.Cl, Dl.Cl\}$.
   In this case the associated rewrite rule takes the form :

   $t\ :\ (Obj, IC_{obj}) \otimes (Ms_{i1}, IC_{ms_{i1}}) \otimes .. \otimes (Ms_{ip}, IC_{ms_{ip}}) \otimes (Ms_{j1}, IC_{ms_{j1}}) \otimes .. \otimes (Ms_{jq}, IC_{ms_{jq}})$
   $\Rightarrow (Obj, CT_{obj}) \otimes (Ms_{h_1}, CT_{ms'_{h_1}}) \otimes .. \otimes (Ms_{h_r}, CT_{ms'_{h_r}}) \otimes (Ms_{j_1}, CT_{ms'_{j_1}}) \otimes .. \otimes (Ms_{j_q}, CT_{ms'_{j_q}})\ if\ Condition \wedge M(Ad.Cl) = \emptyset \wedge M(Dl.Cl) = \emptyset$ [4]

   With respect to the particular case presented in figure **??** we have the form:
   $t\ :\ (Obj, < I_1|attrs_1 > \oplus .. \oplus < I_k|attrs_k >) \otimes (Ms_{i1}, ms_{i1}) \otimes .. \otimes (Ms_{ip}, ms_{ip}) \otimes (Ms_{j1}, ms_{j1}) \otimes .. \otimes (Ms_{jq}, ms_{jq})$
   $\Rightarrow (Obj, < I_{s1}|attrs'_{s1} >) \oplus .. \oplus < I_{st}|attrs'_{st} > \oplus < I_{i_1}|attrs_{i_1} >) \oplus .. \oplus < I_{i_r}|attrs_{i_r} >) \otimes (Ms_{h1}, ms'_{h1}) \otimes .. \otimes (Ms_{hr}, ms'_{hr}) \otimes (Ms_{j1}, ms'_{j1}) \otimes .. \otimes (Ms_{jq}, ms'_{jq})\ if\ Condition \wedge M(Ad.Cl) = \emptyset \wedge M(Dl.Cl) = \emptyset$.

2. $IC(p,t) \cap DT(p,t) = \emptyset, p \in \{obj, Ms_{i1}, .., Ms_{ip}, Ms_{j1}, .., Ms_{jq}, Ms_{h1}, .., Ms_{hr}, Ad.Cl, Dl.Cl\}$:

   $t\ :\ (Obj, IC_{obj} \oplus DT_{obj}) \otimes (Ms_{i1}, IC_{ms_{i1}} \oplus DT_{ms_{i1}}) \otimes .. \otimes (Ms_{ip}, IC_{ms_{ip}} \oplus DT_{ms_{ip}}) \otimes (Ms_{j1}, IC_{ms_{j1}} \oplus DT_{ms_{j1}}) \otimes .. \otimes (Ms_{jq}, IC_{ms_{jq}} \oplus DT_{ms_{jq}}) \Rightarrow (Obj, IC_{obj} \oplus CT_{obj}) \otimes (Ms_{i1}, IC_{ms_{i1}}) \otimes .. \otimes (Ms_{ip}, IC_{ms_{ip}})$
   $\otimes (Ms_{j1}, IC_{ms_{j1}}) \otimes .. \otimes (Ms_{jq}, IC_{ms_{jq}}) \otimes (Ms_{h1}, CT_{ms_{h1}}) \otimes .. \otimes (Ms_{hr}, CT_{ms_{hr}})$
   $\otimes (Ms_{j1}, CT_{ms'_{j1}}) \otimes (Ms_{jq}, CT_{ms'_{jq}})\ if\ Condition \wedge M(Ad.Cl) = \emptyset \wedge M(Dl.Cl) = \emptyset$.

3. $IC(p,t) \cap DT(p,t) \neq \emptyset$ as for ECATNets this case can be expressed using the two first cases.

4. By assuming (for simplicity) that only the input condition associated with the message place Ms_{i1} is preceded by the notation $\sim$, the associated rewrite rule is then:

   $T : (Obj, IC_{obj} \oplus DT_{obj}) \otimes (Ms_{i1}, {}^{\sim} IC_{ms_{i1}} \oplus DT_{ms_{i1}}) \otimes .. \otimes (Ms_{ip}, IC_{ms_{ip}}) \otimes$

[4] This later condition (i.e. $M(Ad.Cl) = \emptyset \wedge M(Dl.Cl) = \emptyset$) ensures that any delete and create message should be performed at first; and this in order to avoid any form of inconsistency like the manipulation of an object already (logically) deleted (i.e. there is a sent message for deleting this object but not already performed).

$(Ms_{ip}, DT_{ms_{ip}}) \otimes (Ms_{j1}, IC_{ms_{j1}}) \otimes (Ms_{j1}, DT_{ms_{j1}}) \otimes .. \otimes (Ms_{jq}, IC_{ms_{jq}}) \otimes (Ms_{jq}, DT_{ms_{jq}})$
$\Rightarrow (Obj, IC_{obj} \oplus CT_{obj}) \otimes (Ms_{i2}, IC_{ms_{i2}}) \otimes .. \otimes (Ms_{ip}, IC_{ms_{ip}}) \otimes (Ms_{j1}, IC_{ms_{j1}}) \otimes .. \otimes (Ms_{jq}, IC_{ms_{jq}}) \otimes (Ms_{h1}, CT_{ms_{h1}}) \otimes .. \otimes (Ms_{hr}, CT_{ms_{hr}}) \otimes (Ms_{j1}, CT_{ms'_{j1}}) \otimes (Ms_{jq}, CT_{ms'_{jq}})$ *if Condition* $\wedge\ M(Ad.Cl) = \emptyset \wedge M(Dl.Cl) = \emptyset \wedge IC_{Ms_{i1}} \not\subset (M(Ms_{i1}) \ominus$[5] $DT_{Ms_{i1}})$

Example 4.3.1 *Following the above syntactical and semantical features of an* OB-NET*, the behaviour associated with the product signature is depicted in figure 4.4; where, 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*

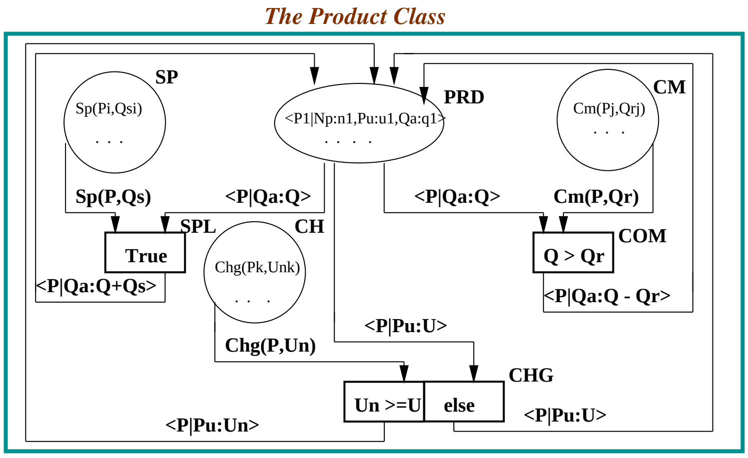

Figure 4.4: The Product class modeled as an OB-NET

and CHG take respectively the form.

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

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

CHG: $(CH, Chg(P, Us)) \otimes (Prd, < P|Pu : U >) \Rightarrow$ *if* $Us \geq U$ *then* $(Prd, < P|Pu : Us >)$ *else* $(Prd, < P|Pu : U >)$[6].

An associated simplified OB-NET *(database) state is depicted in the left-hand side of figure 4.5 and formally corresponds to:*

$(SP, sp(p2, 30) \oplus sp(p4, 10)) \otimes (CM, cm(p3, 25) \oplus cm(p5, 20)) \otimes (CHG, ch(p4, 300) \oplus ch(p3, 600) \oplus ch(p5, 80)) \otimes (PRD, < p4|Np : n4, Pu : 320, Qa : 2 > \oplus < p1|Np : n1, Pu : 200, Qa : 25 > \oplus < p2|Np : n2, Pu : 340, Qa : 23 > \oplus < p3|Np : n3, Pu : 450, Qa : 30 > \oplus < p5|Np : n5, Pu : 50, Qa : 10 >)$

[5] Where $M(Ms_{i1})$ stands for the marking of the place Ms_{i1} and $\ominus$ stands for the multisets difference.
[6] We state that the change-price request is ignored when the condition doesn't hold.

Using the associativity and the commutativity of both $\oplus$ *and* $\otimes$ *and the splitting/merging axiom, we will deduce the OB-Net in the right hand of figure 4.5 by just* **one-step concurrent** $\mathcal{R}$**-rewrite**—*by exhibiting several intra- and inter-object concurrency.*
First we should 'split' the different object states in the place PRD w.r.t. the left hand sides of the rewrite rules. This yields the following sub-state :

$(PRD, < p4|Np : n4 > \oplus < p4|Pu : 320 > \oplus < p4|Qa : 2 > \oplus < p1|Np : n1, Pu : 200, Qa : 25 >$ [7] $\oplus < p2|Np : n2 > \oplus < p2|Pu : 340 > \oplus < p2|Qa : 23 > \oplus < p3|Np : n3 > \oplus < p3|Pu : 450 > \oplus < p3|Qa : 30 > \oplus < p5|Np : n5 > \oplus < p5|Pu : 50 > \oplus < p5|Qa : 10 >)$

Now using the distribution law of the $\otimes$ *over* $\oplus$ *and their communicative properties, we obtain the following equivalent OB-Net state:*

$(SP, sp(p2, 30)) \otimes (PRD, < p2|Qa : 23 >) \otimes (CM, cm(p3, 25)) \otimes (PRD, < p3|Qa : 30 >) \otimes (CHG, ch(p3, 600)) \otimes (PRD, < p3|Pu : 450 >) \otimes (SP, sp(p4, 10)) \otimes (PRD, < p4|Qa : 2 >) \otimes (CHG, ch(p4, 300)) \otimes (PRD, < p4|Pu : 320 >) \otimes (CM, cm(p5, 20)) \otimes (PRD, < p5|Qa : 10 >) \otimes (CHG, ch(p5, 80)) \otimes (PRD, < p5|Pu : 50 >) \otimes (PRD, < p1|Np : n1, Pu : 200, Qa : 25 >) \otimes (PRD, < p2|Np : n2 >) \otimes (PRD, < p3|Np : n3 >) \otimes (PRD, < p4|Np : n4 >) \otimes (PRD, < p5|Np : n5 >)$

By concurrently applying the replacement rule to different sub-terms dealing either with the same object state like the supplying of the object p4 and the change of its unit price or on different object states, we obtain directly the final state—after 'merging' the different object states— as depicted in the right hand side of figure 4.5.

$(PRD, < p1|Np : n1, Pu : 200, Qa : 25 > \oplus < p2|Np : n2, Pu : 340, Qa : 53 > \oplus < p3|Np : n3, Pu : 600, Qa : 5 > \oplus < p4|Np : n4, Pu : 320, Qa : 12 > \oplus < p5|Np : n5, Pu : 80, Qa : 10 >) \otimes (CM, cm(p5, 20))$

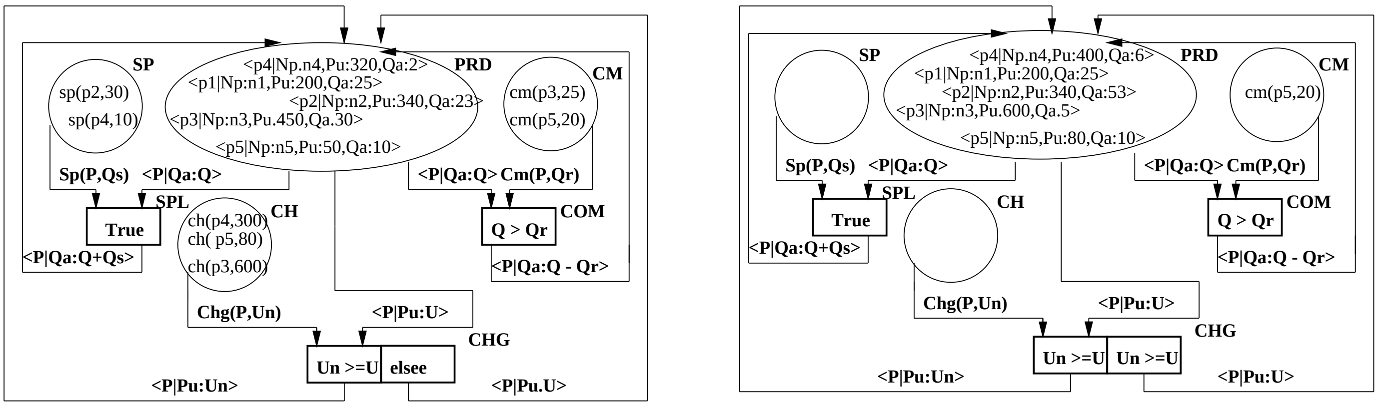

Figure 4.5: A Simplified product database with its state evolution

[7]No message is sent to this object state so there is no need to split it.

Chapter 5

CO-NETS : A Model for Describing OO systems

This section extends the OB-NETS model for dealing with the main OO abstractions mechanisms, namely, the different forms of inheritance and the interaction between autonomous components—where each component is defined as a hierarchy (through inheritance) of classes. The resulting model will be referred to as CO-NETS (a short for, Concurrent Object-oriented Net-based).

5.1 CO-NETS : Simple and Multiple Inheritance

Although having received several satisfactory formalizations in the sequential setting, the concept of inheritance—as incrementally constructing new class(es) by (mainly) extending the structure as well as the behaviour of already existing one(s)—introduces a challenging problems in the presence of concurrency widely known as the *inheritance anomaly* [MY93, Mes93b, LLNW96]. Besides that problem, most of the (net-based approaches) proposals for the inheritance resort to some bit complex development.

By mainly pursuing further the splitting/merging axiom, we propose a conceptualization for different forms of inheritance that we argue doesn't suffer from the mentioned shortcomings. That is to say, it harmonious coexists with the intra- as well as the inter-object concurrency and does not necessitate any tedious and complex formalization. Firstly, the simple inheritance is presented. Secondly, we deal the multiple inheritance and, finally, we comment on the possibility of overriding.

5.1.1 Simple inheritance

We assume given a template specification denoted $T =< TS, TNet >$, where TS is a template signature and $TNet$ its associated (marked) OB-NET. Describing another template denoted $HT =< HS, HNet >$ as *specialization* of T means that first HS recognizes TS and may include additional attributes and messages generators; second $HNet$ should behave exactly as $TNet$ (even for objects with the additional attributes) and may exhibit additional behaviour induced by the newly introduced attributes and messages into HS.

As well-known, the first condition can be straightforwardly formalized as an inclusion (template signature) morphism [EGS92, GD93]. Such morphism is captured by the *extending* primitive in the OBJ language.

Example 5.1.1 *Let's define as a subclass of products those that are perishable, characterized for instance by additional attributes like: the Low-temperature(Lt) and the High-temperature(Ht) tolerated by such products. We also assume that the current temperature is regularly compared to both limits, and when it is not included between them, the corresponding product is removed—the associated message will be denoted by Chk. The template signature of this Product subclass that we denote by Prs-Product is specified as follows:*

```
obj Prs-Product is .
  extending Product .
  subsort Prs-Prd < Prd.
  sort Chk.
  subsort Id.Prs < Id.Prd¹ .
  (* Local attributes *)
    op < _|Lt : _, Ht : _ > :  Id.Prs Real Real → Prs-Prd.
  (* Local messages *)
    op chk :  Id.Prs Real → Chk .
  vars I:Id.Prd; N:Nat ; U,Q,Qs,Qr :  Real .
endo .
```

After having described how the structure of an inherited template is captured, we focus now on the behavioural inheritance i.e. how to construct the $HNet$ from a given $TNet$. Intuitively, the inherited net $HNet$ is constructed from the $TNet$ as follows:

- For inheriting the behaviour of the (super-)template specification TS, we have to keep the $TNet$ *unchanged* except for the *object place* that we allow it to contain (in addition to the object states of the superclass) object states of the subclass (i.e. those with the additional attributes). This is semantically sound because the sort of this object place, i.e. *obj*, is a supersort for any object states including more attributes. Noting that such conceptualization is sufficient for inheriting the behaviour due to the splitting/merging axiom. Indeed, this axiom allows us to consider an object state of a subclass— denoted for instance $< Id|attrs, attrs' >$ with $attrs'$ as additional attributes (i.e. those proper to the subclass)—to be also an object state of the superclass (i.e. $< Id|attrs >$).

- As previously defined, the proper behaviour of the subclass is constructed by associating with each new message a corresponding place and constructing its behaviour (i.e. transitions) w.r.t the described general state evolution pattern.

Assuming that the sub-template HS contains as object sort obj_1 ($< obj$) and as message sorts $\{Mes'_1, .., Mes'_k\}$, the formal definition is as follows:

Definition 5.1.2 *Given a template specification* $T =< TS, TNet >$ *with* $TNet = (P, T, IC, DT, CT, TC)$, *another template specification* $H =< HS, HNet >$ *is a specialization of* T *iff:*

1. *HS is an extension of TS as previously defined.*

2. *HNet referred to as an* CO-NET *is structure,* $(P_1, T_1, IC_1, DT_1, CT_1, TC_1)$, *where:*

- $P_1 = P \cup \{Ms'_1, ...Ms'_q\}$

- $T_1 = T \cup \{T'_1, ...T'_r\}$
- $IC_1 : (P_1 \times T_1) \to MT_{obj}(X)^+ \cup MT_{obj1}(X)^+$ *is a partial function such that for every* $(p,t) \in Dom(IC_1)$, *if* $p = obj$ *and* $t \in T$ *then* $IC_1(p,t) = IC(p,t)$ *else if* $p = obj$ *then* $IC_1(p,t) \in MT_{obj1}(X)^+$. *If* $p \in P - \{obj\}$ *and* $t \in T$ *then* $IC_1(p,t) = IC(p,t)$ *else if* $p \in Mes'_j$ *and* $t \in \{T'_1, ...T'_r\}$ *then* $IC_1(p,t) \in MT_{Mes'_j}(X)^+$ *for each* $j \in \{1, .., k\}$ *else* $IC_1(p,t)$ *is undefined .*
- $DT_1 : (P_1 \times T_1) \to MT_{obj}(X) \cup MT_{obj1}(X)$ *is a partial function such that for every* $(p,t) \in Dom(DT_1)$, *if* $p = obj$ *and* $t \in T$ *then* $DT_1(p,t) = DT(p,t)$ *else if* $p = obj$ *then* $DT_1(p,t) \in MT_{obj1}(X)$. *If* $p \in P$ *and* $t \in T$ *then* $DT_1(p,t) = DT(p,t)$ *else if* $p \in Mes'_j$ *and* $t \in \{T'_1, ...T'_r\}$ *then* $DT_1(p,t) \in MT_{Mes'_j}(X)$ *for each* $j \in \{1, .., k\}$ *else* $DT_1(p,t)$ *is undefined .*
- $CT_1 : (P_1 \times T_1) \to MT_{obj}(X) \cup MT_{obj1}(X)$ *is a partial function such that for every* $(q,t) \in Dom(CT_1)$, *if* $q = obj$ *and* $t \in T$ *then* $CT_1(q,t) = CT(q,t)$ *else if* $q = obj$ *then* $CT_1(q,t) \in MT_{obj1}(X)$. *If* $q \in P$ *and* $t \in T$ *then* $CT_1(q,t) = CT(q,t)$ *else if* $q \in Mes'_j)$ *and* $t \in \{T'_1, ...T'_r\}$ *then* $CT_1(q,t) \in MT_{Mes'_j}(X)$ *for each* $j \in \{1, .., k\}$ *else* $CT_1(q,t)$ *is undefined.*
- $TC : T_1 \to T_{bool}(X)$. *is a function s.t. for every* $t \in T$, $TC_1(t) = TC(t)$ *else if* $t \in T'_1, ..., T'_r$ *then* $TC_1(t) \in T_{bool}(X)$ *where* $X(t)$ *is the set of variables occurring in* $IC_1(p,t)$ *when defined,* $DT_1(p,t)$ *and* $CT_1(p,t)$ *for every* $p \in P_1$.

Definition 5.1.3 *A Marked* CO-NET *defining a subclass is an* CO-NET *with a function* $M : P \to MT_{obj}(\emptyset) \cup MT_{Ms}(\emptyset) \cup MT_{obj1}(\emptyset) \cup MT_{Mes'_j}(\emptyset)$ *such that for* $p = obj$; $M(obj) \in MT_{obj}(\emptyset) \cup MT_{obj1}(\emptyset)$ *(i.e such place contains superclass as well as subclass object states); else* $M(p) \in MT_{Mes}(\emptyset)$ *for* $p \in P$ *else* $M(p) \in MT_{Mes'}(\emptyset)$.

Example 5.1.4 *The* CO-NETS *associated with the subclass perishable-product (with its superclass) is depicted in figure 5.1. In this net the inherited part remain unchanged except that the place Prd contains now (normal) product as well as perishable object states. Thus, while the rewrite rule for the inherited part correspond exactly to the ones already defined for the class Prd, the new rewrite rule associated with the method Chk takes the following form:*
$CHK : (Prd, < P|Lt : L, Ht : H >) \otimes (chk, ch(P,T)) \Rightarrow$ *if* $(T < L \vee T > H$ *then* $(Prd, < P|Lt : L, Ht : H >) \otimes (Dl.Prd, dl_{Prd}(P))$ *else* $(Prd, < P|Lt : L, Ht : H >)$

Remark 5.1.5 *The same reasoning can naturally be repeatedly applied when a (sub)class have to be defined as a subclass of another subclass leading to a (simple) hierarchy of classes. The dynamic binding with the polymorphism is systematically taken into account in this conceptualization. In fact, when a message is sent to a hierarchy of classes, its corresponding class is known only after firing the associated transition. As we have mentioned above, our proposal for the inheritance in addition to be straightforward does not suffer from the inheritance anomaly. This can be easily verified, for instance, for the Perishable-product specification; where both intra- and inter-object concurrency coexist in harmonious with the inheritance. Let's assume having an (perishable) object state* $< p8|Np : p8, Pu : 500, Qa : 30, Lt : 10, Ht : 28 >$ *and the following messages sent to it:* $cm(p8, 18)$, $chg(p8, 630)$ *and* $chk(p8, 18)$. *The corresponding* CO-NETS *state takes the form:*

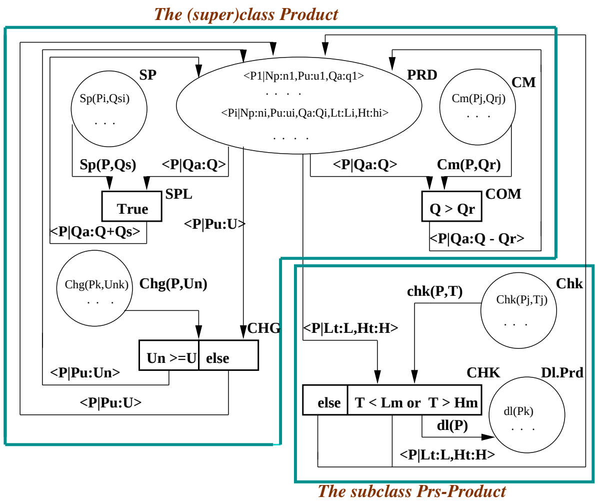

Figure 5.1: The definition of Perishable Products as a subclass of Product

$(Prd, < p8|Np : p8, Pu : 500, Qa : 30, Lt : 10, Ht : 28 >) \otimes (CM, cm(p8, 18)) \otimes (CHG, chg(p8, 630)) \otimes (CHK, chk(p8, 18))$

For performing concurrently these messages, we should first split the object state p8 adequately. Such splitting yields the following equivalent Co-nets *state:*

$(Prd, < p8|Np : p8 > \oplus < p8|Pu : 500 > \oplus < p8|Qa : 30 > \oplus < p8|Lt : 10, Ht : 28 >) \otimes (CM, cm(p8, 18)) \otimes (CHG, chg(p8, 630)) \otimes (CHK, chk(p8, 18))$

By applying the decomposition rule and the commutativity of $\otimes$ *we result in the following state:*

$(CHG, chg(p8, 630)) \otimes (Prd, < p8|Pu : 500 >) \otimes (CM, cm(p8, 18)) \otimes (Prd, < p8|Qa : 30 >) \otimes (CHK, chk(p8, 18)) \otimes (Prd, < p8|Lt : 10, Ht : 28 >) \otimes (Prd, < p8|Np : p8 >)$

The concurrent application of the three rewrite rules associated with the three message yields the following final state (after the merging operation).

$(Prd, < p8|Np : p8, Pu : 630, Qa : 12, Lt : 10, Ht : 28 >)$

5.1.2 Multiple Inheritance

Although it is quite possible to extend the proposed solution to multiple inheritance, the gathering of all object places—that requires this solution— associated with different (and independent) superclasses into a single place may leads to a very complex and unnatural net. In order to avoid such complexity, we propose a slight different conceptualization that keep all superclasses unchanged.

- In multiple inheritance, an object state of a given subclass, denoted *subCl*, has to be composed from the proper attributes of this class plus those of the superclasses denoted Cl_1,..,Cl_n. For this reason, we propose to *distribute* this state over the different associated object places. That is to say, if the state object in the *subCl* is of the form $< I|attrs_{Cl1}, ..., attrs_{Cln}, attrs_{subCl} >$, then the following parts $< I|attrs_{Cl1} >$, ... , $< I|attrs_{Cln} >$, $< I|attrs_{subCl} >$ should respectively belong to the object places associated with the classes Cl_1,..,Cl_n, *subCl*. Noting that, such (different) object state components should have the *same identity* (denoted here by I).
- As illustrated in the left hand-side of figure 5.2, the transitions in the subclass have

to take into consideration all the different components of an object state. In other words, the input arcs as well as the output arcs of these transitions are not only those arcs from the object place of the subclass but also from the superclasses ones.

- As depicted in the right hand of figure 5.2, the creation process of objects for such subclass have to be slightly modified. Indeed, first, we have to control the uniqueness of the object identity in different superclasses. Second, we have to add different components of the created object to their corresponding places. Similarly, the deletion process have to be adapted in consequence.

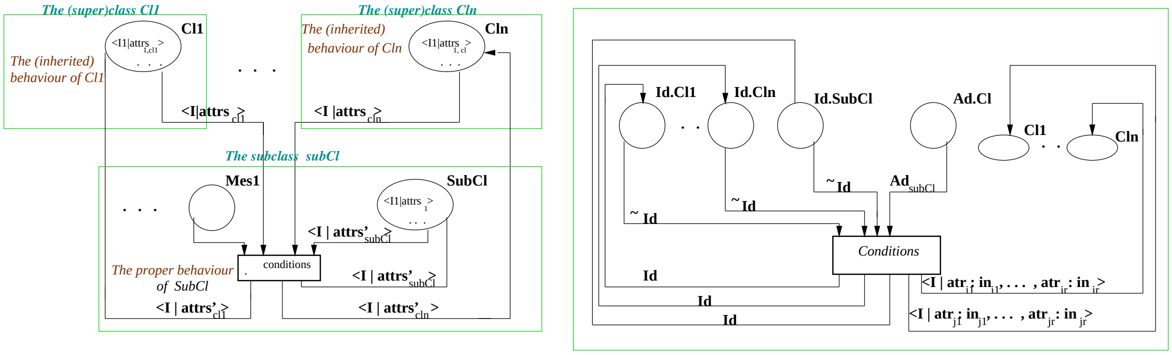

Figure 5.2: A Sketch on how Multiple Inheritance is conceptualized using the CO-Net Model.

For formally capturing this intuitive conceptualization, we denote by $MT{=}{<}MS,\ CO\text{-}Net{>}$ the sub-template specification that we intend to define as a specialization of more than one independent (super) templates denoted as $T_i = < TS_i, CO\text{-}Net_i >$ with $i \in \{1, .., s\}$. Denoting by obj_i the corresponding object sorts in T_i, the object sort in MT that we denote by obj_m should verify $obj_m < obj_i$ for all $i \in \{1, .., s\}$—this because obj_m have to include in addition to proper attributes those in the super-template. Finally, we denote by $\{Mes_{m_1}, .., Mes_{m_q}\}$ the proper messages in MS.

Definition 5.1.6 *a multiple inherited CO-Net, defined from s* $CO\text{-}Net_i = < P_i, T_i, IC_i, DT_i, CT_i, TC_i >$ *is a CO-Net* $= < P, T, IC, DT, CT, TC >$ *with:*

- $P = P_1 \cup P_2 \cup ... \cup P_s \cup \{Mes_{m_1}, ..., Mes_{m_q}\} \cup obj_m$.
- $T = T_1 \cup T_2 \cup ... \cup T_s \cup \{T'_1, ..., T'_h\}$.
- $IC : (P \times T) \to MT_{obj_m}(X)^+ \cup MT_{obj_i}(X)^+ \cup .. \cup MT_{obj_s}(X)^+ \cup MT_{Msg}(X)^+$ *is a partial function such that for every* $(p, t) \in Dom(IC)$, *if* $t \in T_i$ *then* $IC(p, t) = IC_p$ *else if* $p = obj_i$ *and* $t \in \{T'_1, .., T'_h\}$ *then* $IC(p, t) \in MT_{obj_i}(X)^+$ *else if* $p \in \{Mes_{m1}, .., Mes_{mq}\}$ *and* $t \in \{T'_1, .., T'_h\}$ *then* $IC(p, t) \in MT^+_{Mes_{m_i}(X)}$ *else* $IC(p, t)$ *is undefined.*
- $DT : (P \times T) \to MT_{obj_m}(X) \cup MT_{obj_i}(X) \cup .. \cup MT_{obj_s}(X) \cup MT_{Msg}(X)$ *is a partial function such that for every* $(p, t) \in Dom(DT)$, *if* $t \in T_i$ *then* $DT(p, t) = DT_p$ *else if* $p = obj_i$ *and* $t \in \{T'_1, .., T'_h\}$ *then* $DT(p, t) \in MT_{obj_i}(X)$ *else if* $p \in \{Mes_{m1}, .., Mes_{mq}\}$ *and* $t \in \{T'_1, .., T'_h\}$ *then* $DT(p, t) \in DT_{Mes_{m_i}(X)}$ *else* $IC(p, t)$ *is undefined.*
- $CT : (P \times T) \to MT_{obj_m}(X) \cup MT_{obj_i}(X) \cup .. \cup MT_{obj_s}(X) \cup MT_{Msg}(X)$ *is a partial function such that for every* $(q, t) \in Dom(CT)$, *if* $t \in T_i$ *then* $CT(q, t) = CT_p$ *else if* $q = obj_i$ *and* $t \in \{T'_1, .., T'_h\}$ *then* $CT(q, t) \in MT_{obj_i}(X)$ *else if* $q \in \{Mes_{m1}, .., Mes_{mq}\}$ *and* $t \in \{T'_1, .., T'_h\}$ *then* $CT(q, t) \in MT_{Mes_{m_i}(X)}$ *else* $CT(q, t)$ *is undefined.*

- $TC : T \rightarrow T_{bool}(X)$. *is a function s.t. for every* $t \in T_i$, $TC1(t) = TC_i(t)$ *else if* $t \in T'_1, ..., T'r$ *then* $TC1(t) \in T_{bool}(X)$.

5.1.3 Inheritance with Overriding

Firstly, It is worth to mention that according to our conceptualization, each class or subclass is endowed with a (identifiers) place that contains the identities of current objects of such class. Taking this into account, the overriding of a given method, modeled as a transition, when defining a subclass consist in the two following steps:

- To disable the object states proper to the subclass to inherit such a method (i.e. transition), we propose simply to add to the input places of this transition the identifiers place associated with this subclass, and to annotate the corresponding arcs by $\sim I$ where I is the involved object state in such method. This means that the transition will be firable only for object states of the superclass.
- Construct the new behaviour (i.e. a new transition) by taking as input message(s) place(s) those involved in the transition to be redefined; this is because a redefined method by definition should have the same name.

Example 5.1.7 *Assume we wish to redefine the method that change the price (chg) in the subclass perishable-products in such a way that such change is only possible if, for instance, the difference between the two temperature is less than 20. Following the two above steps the redefined subclass is depicted in figure 5.3. More precisely, to the transition CHG we have add as input place Id.Prs-Prd with* $\sim P$ *for the corresponding arc, and we have constructed the new transition (R-CHG) with the same input places as the redefined transition CHG.*

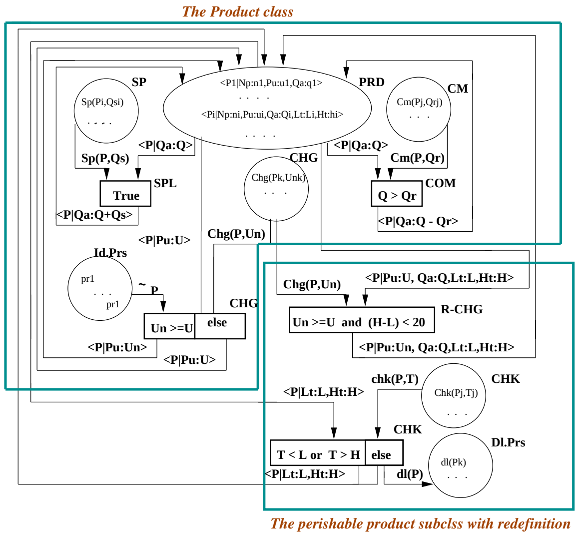

Figure 5.3: The Redefinition of the change price method in the subclass Prs-Product

Chapter 6

CO-NETS : Components Interaction

So far, we have only dealt with independent components as hierarchies of modules through simple and multiple inheritance with possibility of overriding. Nevertheless, in the templates and classes specification we have already introduced the notion of observed attributes as well as the notion of external (i.e. as import/export) messages. On the basis of this object 'interface', we define the notion *template (and class) community* as communicating and cooperative components. While each component is structurally and behaviorally defined exactly as presented up to now, the main feature of a given community is the introduction of an appropriate *inter-component* interaction pattern that allows for communicating such components in such a way that enhance intra- and inter-object concurrency and respect the encapsulated features of each component.

This interaction pattern, as depicted in figure 6.1, describes then the general schema of 'external' transitions and may be explained as follows. The contact of some and only *external* parts of some objects states namely $< I_1|attrs\_ob_1 >, ... < I_k|attrs\_ob_k >$, that may belong to different classes namely $Cl_1, ..., Cl_m$, with some external (i.e. declared as imported or exported) messages $ms_{i1}, .., ms_{ip}, ms_{j1}, .., ms_{jq}$ defined in such 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 $I_{s1}, .., I_{st}$. Such 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 $ms'_{h1}, .., ms'_{hr}, ms'_{j1}, .., ms'_{jq}$.

Remark 6.0.8 *Once again this interaction is possible only due to the splitting /merging axiom that allows for separating between local and external attributes in a given object state. Thus, in the same way as for the inheritance, it is not difficult to prove that this conceptualization promote intra- as well as inter-object concurrency. Notice that the object places of different participating classes are containing only the observed part of each object state. This may be simply formalized using a (restriction) morphism from the object sort of each class to its observed part sort—Following the OBJ description in figure 3.1 this morphism should be from Object to External-Attributes. This allows to hide the local part*

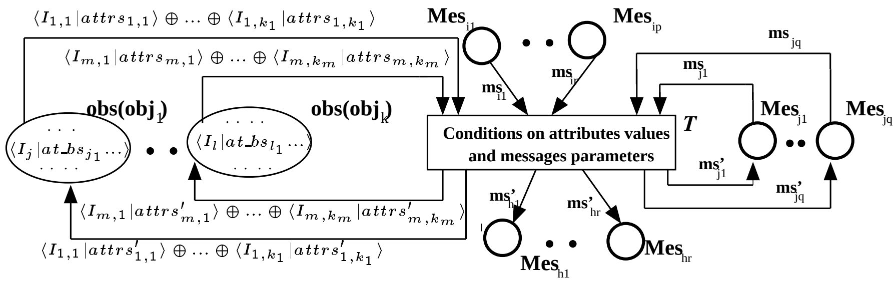

Figure 6.1: The Inter-Component Interaction pattern.

of attributes from the outside. The same reasoning should be applied to the messages places, where just the external ones have to appear in this inter-component interaction. Recalling also that, when no observed attributes are declared and thanks to the axiom 'eq2' in the object-state description in figure 3.1 just the identity of invoked object participate in the interaction. In the same way as for the general form of the intra-component pattern, the more abstract inter-component pattern may deduced by replacing the inscription of input (resp. output) arcs from ms_i *(resp.* ms'_i*) into* IC_{ms_i} *and* DT_{ms_i} *(resp.* IC_{ms_i} *and* DT_{ms_i}*). The replacement of the inscription from and to the different object places* $obj_1,..,obj_m$ *from the depicted form (i.e.* $< I_{i,1}|attrs_{i,1} > \oplus .. \oplus < I_{i,i}|attrs_{i,i} >$*) to* $IC_{obs(obj_i)}$ *and* $DT_{obs(obj_i)}$ *for the input arcs and* $CT_{obs(obj_i)}$ *for the output arcs. Where* $obj(obj_i)$ *correspond to the morphism that hide the local attributes parts of each object. Due to the existence of more than one component that are independent, the object creation process presented in figure 4.1 have to be slighly modified in order to avoid objects with same identities belonging to different components. For this aim, we propose to add an additional input/output common place, for all components, that should contains all the identities in the whole system and control the uniqueness of identities exactly as the identifiers place of each component.*

Finally, by assuming for simplicity that each component in constructed from just one a class (i.e. with no inheritance) the notion of template community can be made more precise by gathering the different template specifications together as follows:

Definition 6.0.9 *A template community is 3-tuple* $TCM = (S_D \cup S_O \cup S_{Msg}, Ms_L \cup Ms_O, Nets)$*, with*

- S_D *a set of the data sorts.*
- S_O *is a finite of set of object sorts. Namely,* $S_O = \{obj_1, obj_2, .., obj_m\}$*. Each object sort is defined as an operation as defined in the template signature.*
- S_{Msg} *is a collection of m set of (messages) sorts. Namely,* $S_{Msg} = \{S_{Msg_1}, .., S_{Msg_1}\}$*, where each set of sorts* S_{Msg_1} *is defined each given in the template signature definition.*
- $Ms_L \cup Ms_O$ *is also a collection of set of message operators. Each set is defined w.r.t. a given* obj_i *sort as given in the template signature definition.*
- $Nets$ *is a set of* CO-NETS *defined as:*$\{CO - Nets_1, .., CO - Nets_m, CM - Nets\}$*. Each* $CO - Net_i$ *represents the behaviour of the object of sorts* obj_i*. The additional (communication) net* $CM - Nets$ *reflect the interaction between objects of different types. This net is constructed w.r.t. to the interaction pattern in figure 6.1.*

Example 6.0.10 *With this possibility of interacting independent component, is is no more difficult to conceive a complete invoice system including, in addition to the product component, the suppliers and the customers classes. Moreover, the 'invoice', sent to the supplier after each successful order, may be modeled as class. However, for interacting such component the designer (with the users and managers) of the system have to define precisely what informations can exchanged between such components. In other words, w.r.t. each component, which attributes and/or messages may observed. For the product component we state that only the unit-price as attributes have to be observed and the messages supply and command have to be declared as imported messages. Without going into detail, we depict hereafter the adapted product class and in figure 6.0.10 a sketch of this invoice system.*

obj *Product* **is**
  sort *Prd, Id.Prd, Id.Cust, Id.Supl, Sp, Cm, Chg .*
  subsort *Id.Prd* $<$ *OId .*
  protecting *Nat Real String Object-state .*
  Local attributes
    op $< \_|Np:\_, Qa:\_>$: *Id.Prd Nat Real* $\rightarrow$ *Prd .*
  Observed attributes
    op $< \_|Pu:\_>$: *Id.Prd Real* $\rightarrow$ *Prd .*
  Local messages
    op *chg : Id.Prd Real* $\rightarrow$ *Chg .*
  imported messages
    op *sp : Id.Prd Id.Supl Real* $\rightarrow$ *Sp .*
    op *cm : Id.Prd Id.Cust Real* $\rightarrow$ *Cm .*
  vars *I: Id.Prd, N:Nat , U Q, Qs, Qr : Real .*
endo.

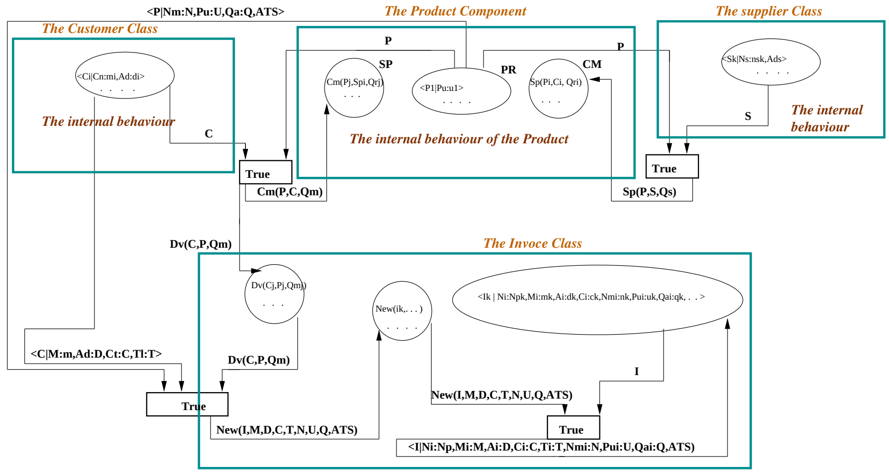

Figure 6.2: The Invoice System Specified as a CO-NET

Chapter 7

Comparison with related work

As pointed out the introduction of this paper, there is actually several ongoing work dealing with this vivid area of research, namely: the foundation of (the concurrent) object paradigm. Therefore, any attempt towards an exhaustive comparison of proposals in this area seems to be too premature. However, it is nowadays quite possible to assess an object oriented framework w.r.t. some more or less widely accepted object criterion—that become more exhaustive more the application domains of such framework is well defined. In our case, CO-NETS are essentially dedicated to be used for specifying and validating distributed and cooperative information systems. Henceforth, we found the classification criterion defined within the European DEVA project [BBC+97] is quite close to our objectives. In this work, in addition to the characterization of the most exhaustive object criterion, the authors have also situated most of the (formal) object approaches (including algebraic-based as well as net-based ones) w.r.t. such criterion. The criterion have been classified into three main classes: the object-based criterion, the object-oriented criterion and the theoretical foundation one. The application of these criterion on the CO-NETS approach results in the following table. In this table, we can notice that except for the lack of notion of object as classes (i.e. absence of any form of reflection) all the other criterion are fulfilled.

Object-Based Criteria	**Value**	**Comments**
object and state	Yes	
encapsulation	Yes	
synchronous requests	Yes	
asynchronous requests	Yes	
data abstraction	Yes	ordered sorted algebra
data structure of objects	Yes	state are tokens and behaviour are nets
object identity	Yes	as a user definable data types
intra-concurrency	Yes	
inter-concurrency	Yes	following two communications models
Object-Oriented Criteria	**Value**	**Comments**
classes as templates	Yes	modeled as an CO-Nets
classes as collections	Yes	modeled as a marked CO-Nets
inheritance	Yes	simple, multiple and with redefinition
sub-typing	Yes	using ordered sorted algebra
Multiple inheritance $\neq$ sub-typing	Yes	
classes as objects	No	
collection of objects	Yes	
dynamic/static instantiation	Yes	
genericity and parametrization	Yes	at the data level.
Foundations	**Value**	**Comments**
Semantics	Yes	true concurrent (rewriting logic) semantics
Calculus	Yes	using rewriting techniques

Chapter 8

Conclusion

In this paper we have proposed a new form of OO Petri net-based model referred to as CO-NETS. This OO formalism consists of a sound and complete integration of OO concepts into an appropriate variety of algebraic Petri nets named ECATNets. The main features of the CO-NETS approach include: a clean separation between the data level and the object level, the modeling of class rather as a module with explicit interface, a straightforward modeling of different forms of inheritance without affecting the full exhibition of intra- as well as inter-object concurrency, the characterization of two communication pattern for respectively evolving (state) objects in an independent component—regarded as a hierarchy of classes— and for interacting objects in different components. While both patterns promote intra- as well as inter-object concurrency, the inter-component interaction pattern abstract away from all local features of the interacted components.

On the other hand, we have situated our model w.r.t. to recent similar proposals through its assessment w.r.t. the exhaustive OO criterion proposed recently in [BBC+97]. This comparison have shown that our model deals adequately with most of OO concepts. However, advantages of some other similar proposals, and more specifically the OPN [Lak95] model, including their integration of *analysis methods* and some forms of reflection capabilities [Lak96b] are of great interest in modeling and verifying concurrent systems and then have to be adapted to our model.

Besides that, being conscious that the presented work is far from being ideal for coping with all aspects in modeling and validating distributed (information) systems, further investigations are planned for the nearest future. Firstly, although the CO-NET approach have been successfully applied on realistic staff management case study [AS99b] and on providing the Mondel specifications with a satisfactory interpretation [AS99a], more large case studies should be drawn up mainly in the field of information systems—as central interest in our laboratory. Secondly, we plan to formally integrate property-oriented verification models. In this sense at least two directions have to be investigated. The first one is to follow the work in [Cos97] that relates labelled transitions systems—as general framework including rewriting logic [Ast99]—with temporal logic. On the other it is also quite possible to take advantages of the work in [Lec96] and [Den98] that relates directly rewriting logic to the temporal setting.

Bibliography

[AS99a] N. Aoumeur and G. Saake. Towards a New Semantics for Mondel Specifications Based on the CO-Nets Approach. In J. Desel and K. Pohl and P. Schuerr, editor, *Proc. of Modellierung'99*, pages 107–122. B.G. Teubner-Verlag, Karlsrule, Germany, March 1999.

[AS99b] 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, Heidelberg, Germany*, Lecture Notes in Computer Science, Vol. 1626, pages 381–395. Springer-Verlag, Berlin, 1999.

[Ast99] E. Astesiano. Algebraic Specification of Concurrent Systems. In H. J et al K., editor, *IFIP 14.3 Volume on Foundations of System Specification, Chapter 1. To appear in Springer LNCS*, 1999.

[AZ94] E. Asteziano and E. Zucca. D-Oids : A Model for Dynamic data Types. *Mathematical Structures in Computer Science*, 5(2):257–282, 1994.

[BB91] M. Baldassari and G. Bruno. PROTOB : an Object Oriented Methodology for Developing Discrete Event Dynamic Systems. *Computing Languages*, 16:39–63, January 1991.

[BBC+97] O. Biberstein, D. Buchs, E. Canver, P. Dauchy, M-C. Gaudel, N. Guelfi, F. von Henke, C. Khoury, B. Marre, Schwier D., and G. Vidal-Naquet. Comparison of Object-Oriented Formal Methods. Technical report of the esprit long term research project 20072: Design for validation, University of Newcastle Upon Tyne, Department of Computer Science, 1997.

[BBG97] O. Biberstein, D. Buchs, and N. Guelfi. 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.

[BCdC95] E. Battiston, A. Chizzone, and de Cindio. Inheritance and Concurrency in CLOWN. In *Proc. of the "Application and Theory Petri Nets" Workshop on Object-Oriented Programming and Models of Concurrency* , 1995.

[BM92] M. Bettaz and M. Maouche. How to Specify Non Determinism and True Concurrency with Algebraic Term Nets, Lecture Notes in Computer Science. Vol. 655, pages 164–180, 1992.

[BMSB93] M. Bettaz, M. Maouche, M. Soualmi, and S. Boukebeche. Protocol Specification using ECATNets. *Reséaux et Informatique Répartie*, 3(1):7–35, 1993.

[Cos97] Costa, G. and Reggio, G. Specification of Abstract Dynamic Dat Types: A Temporal Logic. *Theoretical Computer Science*, 193, 1997.

[DB95] P. Du Bois. *The Albert II Language: On the Design and the Use of a Formal Specification Language for Requirements Analysis.* PhD thesis, Computer Department, University of Namur, Namur(Belgique), September 1995.

[Den98] G. Denker. From Rewrite Theories to Temporal Logic Theories. In H. Kirchner and C. Kirchner, editors, *Proc. of Second International Workshop on Rewriting Logic*, Electronic Notes in Theoretical Computer Science, Vol. 15, 1998.

[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*, Lecture Notes in Computer Science, Vol. 655, pages 40–66, Springer-Verlag, 1992.

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

[EM90] H. Ehrig and B. Mahr. Fundamentals of Algebraic Specifications 2 : Module Specifications and Constraints. *EATCS Monographs on Theoretical Computer Science*, 1990.

[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.

[GD94] J. Goguen and R. Diaconescu. An Oxford Order Sorted Algebra. *Mathematical Structures in Computer Science*, 4(4), 1994.

[GKM87] J. Goguen, C. Kirchner, and M. Meseguer. Concurrent Term Rewriting as a Model of Computation. In Keller, R. and Fasel, J., editor, *Proc. Workshop on Graph Reduction, Santa Fe, New Mexico*, Lectures Notes in Computer Science, Vol. 279, pages 53–93, 1987.

[Gog92] Goguen, J. and Winkler, T. and Meseguer, J. and Futatsugi, K. and Jouannaud, J.P. Introducing OBJ. Technical Report No. SRI-CSL-92-03, Computer Science Laboratory, SRI International, 1992.

[Jac96] J. Jacobs. Objects and Classes, Co-algebraically. In B. Freitag, C.B. Jones, C. Lengauer, and H.-J. Schek, editors, *Object-Orientation with Parallelism and Persistance*, Kluwer Acad. Publ., 1996.

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

[Lak96a] C. Lakos. The consistent use of names and polymorphism in the definition of objects Petri nets. In *Proc. of 17th Application and Theory of Petri Nets*, Lecture Notes in Computer Science, Vol. 1091, pages 380–399, Springer-Verlag, 1996.

[Lak96b] C. Lakos. Towards a Reflective Implementation of Object Petri Nets. In *Proc. of TOOLS Pacific 1996*, pages 129–140, Monash Printing Service, 1996.

[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*, Vol. 4, Electronic Notes in Theoretical Computer Science, 1996.

[LLNW96] U. Lechner, C. Langauer, F. Nickel, and M. Wirsing. (Objects + Concurrency) & Reusability – A Proposal to Circumvent the Inheritance Anomaly. In *ECOOP'96 - Object-Oriented Programming*, Lecture Notes in Computer Science, Vol. 1098, pages 232–248, Springer Verlag, 1996.

[Mes90] J. Meseguer. Rewriting Logic as a unified Model of Concurrency. In Baeten, J. C. M. and Klop, J. W., editor, *13 th Proc. of the CONCUR'90*, Lectures Notes in Computer Science, Vol. 458, pages 384–400, 1990.

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

[Mes93a] J. Meseguer. A Logical Theory of Concurrent Objects and its Realization in the Maude Language. *Research Directions in Object-Based Concurrency*, pages 314–390, 1993.

[Mes93b] J. Meseguer. Solving the Inheritance Anomaly in Concurrent Object-Oriented Programming. In *ECOOP'93 - Object-Oriented Programming*, Lecture Notes in Computer Science, Vol. 707, pages 220–246, Springer Verlag, 1993.

[Mes98] J. Meseguer. Research Directions in Rewriting Logic. In U. Berger and H. Schwichtenberg, editors, *Computational Logic, NATO Advanced Study Institute, Marktoberdorf, Germany*, Springer-Verlag, 1998.

[MY93] S. Matsuoka and A. Yonezawa. Analysis of Inheritance Anomaly in Concurrent Object-Oriented Languages. *Research Directions in Object-Based Concurrency*, pages 107–150, 1993.

[PS98] M. P. Papazoglou and G. Schlageter, editors. *Cooperative Information Systems : Trends and Directions.* Academic Press, Boston, 1998.

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

[Rei94] H. Reichel. An approach to object semantics based on terminal co-algebras. *Mathematical Structure in Computer Science*, 1994.

[SB94] C. Sibertin-Blanc. Communicative and cooperative nets. In E. Astesiano, R. Reggio, and A. Tarlecki, editors, *Proc. of the 15th International Confernce on the application and Theory of Petri Nets*, Lecture Notes in Computer Science, Vol. 815, Springer-Verlag, 1994.

[Ver93] P.A.C. Verkoulen. *Integrated Information Systems Design : An Approach Based on Object Oriented Concepts and Petri Nets.* PhD thesis, Technical University of Eindhoven, The Netherlands, 1993.

[Weg90] P. Wegner. Concepts and paradigms of Object-Oriented Programming. *OOPS Messenger*, 1:7–87, 1990.