Prototyping Object Specifications Using the CO-Nets Approach

Nasreddine Aoumeur Stefan Conrad Gunter Saake
ITI, FIN, Otto-von-Guericke-Universität Magdeburg
Postfach 4120, D–39016 Magdeburg
E-mail: {aoumeur|conrad|saake}@iti.cs.uni-magdeburg.de

Abstract

The CO-Nets approach, that we are developing, is an object oriented Petri net-based framework for specifying as well as prototyping—through graphical animation accompanied by a concurrent computation based on its semantics expressed in rewriting logic—distributed information systems. Taking benefits of these (validation) capabilities, we presents how prototyping and implementation of systems specified using widely accepted information systems languages, namely the TROLL language, can be directly drawn up. This is mainly achieved through an intuitive translation of such specifications into the CO-Nets approach where graphical animation and formal computation are carried out. Moreover, because of the capabilities of the CO-Nets approach for conceiving such systems as autonomous but yet cooperative components, it becomes semantically sound to enrich these languages with syntactical constructions for more modularity leading to more efficient rapid-prototyping.

1 Introduction

Characterized as being open and reactive systems with large databases and application programs, information systems are ubiquitous in most of existing organizations. With the aim of specifying and afterwards implementing (and maintaining) them accurately, very promising object specification languages have been forwarded in the last decade. They include particularly ALBERT [DB95], LCM [FW93] and TROLL [JSHS96]. These languages allow for naturally conceiving such systems as a community of interacting objects related by different abstractions mechanisms including specialization, object composition and aggregation.

However, on the one hand, most of these languages are highly declarative and then lack for an intrinsic operational semantics that allows for bridging the gap between the specification and its corresponding implementation through an efficient rapid-prototyping of the specification. Indeed, besides of being a crucial step towards a correct, secure and efficient implementation, the prototyping phase also allows for checking and then correcting the specification against missing, errors due to misunderstanding between the specifier and the user in a very early phase of the system development. On the other hand, most of these languages according to their associated semantics deal only poorly with the distribution—that has to be reflected by the exhibition of intra- as well as inter-object true concurrency with synchronous as well as asynchronous communication— and modularity as a way of composing different system components without affecting their internal behaviour.

With the aim to contribute to these challenging aspects, we are developing an adequate object-oriented Petri net-based framework for specifying and validating distributed information systems regarded as autonomous and cooperative components. Referred to as CO-Nets, the model is based on a complete and sound integration of the OO concepts and constructions into an appropriate variant of algebraic Petri nets. Particular to the CO-Nets approach, we mainly cite: its semantics that is expressed in rewriting logic [Mes92], which allows a formal and true concurrent computation using rewriting techniques; its capability of modeling classes rather as modules with local features (including structure and behaviour) that are hidden to other classes

and observed, external features as interface for communicating with the environment and other classes; the straightforward modeling of different forms of inheritance (i.e. simple, multiple, with overriding and with associated polymorphism and dynamics binding) leading to the notion of component as a hierarchy of classes. Such components behave w.r.t. an intra-component evolution pattern that enhances intra- as well as inter-object concurrency. For interaction of such components through their interfaces, leading to complex systems, a suitable inter-component interaction pattern is proposed, that promotes (intra- and inter-object) concurrency and keeps encapsulated all local features of the interacting components.

The present work describes first steps on how (distributed) information systems, specified in one of the mentioned languages, particularly the TROLL language, can be easily validated using the CO-Nets capabilities, namely the graphical animation of the constructed nets with a formal computation based on concurrent rewriting techniques. This is mainly achieved through a straightforward translation of such specifications into the CO-Nets approach. Moreover, for *efficiently* prototyping systems specified in this language, we propose to improve the resulted specification by distinguishing between local and external features; and henceforth to enrich the TROLL language with new constructions for dealing with this modularity.

The rest of this paper is organized as follows: the second section presents a simplified 'Bank' object specification using the TROLL language. The main steps for translating TROLL specification into the CO-Nets approach are described in the third section, that we illustrate by the 'Bank' example. In the fourth section, first we present how the object specification have to be improved in order to derive more efficient prototypes, secondly we illustrate how concurrent computation may be efficiently achieved using a concrete object community. Some concluding remarks and hints to our future work close this paper. However, we note that due to space limitation, for an overview of the CO-Nets approach we advice the reader to consult [AS99a] and [AS99b] (that is under :http://link.springer.de/link/service/series/0558/bibs/1626/16260381.htm)

2 Object Specification : An Example

The example of object specification, through which we discuss the translation into the CO-Nets, consists in a small universe of discourse consisting of one or more bank objects and account objects belonging to these banks. The specification language we employ is the TROLL language TROLL [JSHS96] which offers a wide spectrum of modeling concepts for describing structure of objects as well as object dynamics. In this simplified specification, each object class is characterized by the sort of its object identifiers that follows the *identification* clause, by the list of its attributes identifiers, that follows the *attributes* clause, with their corresponding types and optionally initial, restricted or constant values. Finally, the different events with their effect and the condition that gouverns their application are described after the *events* clause.

```
object class Bank
  identification  ByBankID: (Name, No) .
  attributes
    Name:      string .
    No:        nat constant
                   restricted No>=1 and No<=99999999 .
  components
    Acct:      Account set .
  events
    Open(BankName:string, BankNo:nat)
      birth
      changing Name    := BankName,
               No      := BankNo .
    OpenNewAccount(HID:|Customer|, AN:nat)
```

```
      calling   Account(Self, AccountNo).Open(Self, AN, HID)
      changing Acct := insert(Acct, Account(Self, AN).Self) .
    Transfer(AN1:nat, AN2:nat , M:money)
      enabled  in(Account(Self, AN1), Acct) and
               in(Account(Self, AN2), Acct) and
               M > 0.00
      calling   Account(Self, AN1).Withdrawal(M) ,
               Account(Self, AN2).Deposit(M) .
end object class Bank

object class Account
  identification  ByAccountID: (Bank, No) .
  attributes
    No:       nat constant
                  restricted No>=1 and No<=99999999 .
    Bank:     |Bank| .
    Holder:   |Customer| .
    Balance:  money initialized 0.00 .
    Interest:  money initialized 0.00 .
    Limit:    money initialized 0.00
                    restricted Limit<=0.00 .
  events
    Open(BID:|Bank|, AN:nat, HID:|Customer|)
      birth
      changing Bank   := BID,
               No     := AN,
               Holder := HID .
    Withdrawal(W:money)
      enabled  W >= 0.00 and Balance - W >= Limit
      changing Balance := Balance - W .
    Deposit(D:money)
      enabled  D >= 0.00
      changing Balance := Balance + D .
    IncreaseInterest(NI:Interest)
      enabled  Interest <   NI .
      changing Interest := NI .
end object class Account
```

3 Translating Object Specification into CO-Nets

Because of the object oriented setting which the CO-Nets approach as well as the TROLL are based on, (OO) systems specified using TROLL may be translated in straightforward way into the CO-Nets approach. Indeed, by firmly respecting the TROLL concepts that make no distinction between local / hidden object (structural and behavioural) features and external / observed ones[1], the main TROLL concepts (that are addressed in the example) can be expressed in terms of those of the CO-Nets following two steps: template signature translation and template behaviour translation. These two steps are intrinsically related to the CO-Nets approach that makes a clear, but yet coherent distinction between the structural aspects of a given template specification and the behavioural aspects that are captured by associating a (CO-) net.

Accordingly, the translation summarized in the table below can be made more explicit as follows.

[1]We will see in the next section how such distinction, naturally supported by the CO-Nets approach, allows for an optimal and efficient prototyping

Template signature Translation : This concerns mainly the translation of attributes and events declaration.

1. The different attributes associated with a given TROLL object class are directly specified as a (object state) term with adding to it the object identity declared in the identification clause. The possibilities of restricting, initializing or fixing some attributes values have to be expressed as conditions in the transition associated with the object creation.
2. With each event a message is associated. That is to say, just the identity of the invoked object is added as a new argument to each action (name with all its other arguments).

Template behaviour translation : Following the CO-Net approach, in addition to the object place that has to contain the different object state, with each event (now a message) generator a corresponding place is associated. The behaviour of each event is captured by an appropriate transition, where:

1. The place associated with this event is taken as input place, while the event (calling) itself labels the corresponding input arcs;
2. The clause *enabled* is expressed either as conditions in this transition or as appropriate instantiations in the label of the input arc from the object place ;
3. The *changing* clause is modeled as an appropriate labelling of the output arc that goes to the object place.
4. Finally the calling clause is captured by output arcs labelled by the corresponding called message and destinate to their associated (message) place.

Troll concepts	**Mapping to the CO-Nets concepts**
Attributes	Object state as term with addition of the identity
—constant	As constant in the corresponding (algebraic) structure
—restricted, initialized	as conditions in the net for object creation
events	messages with expliciting the identity of the invoked object
— enabled	Transition condition
— calling	messages sent
— changing	Transition effect

Example 3.1 *By applying these translating ideas to our Bank specification running example, we result in the following two corresponding CO-Net template signatures and the associated (interacted) (CO-)nets. For instance, in the* **Account** *template signature all the attributes identifiers (with their corresponding sorts) are gathered in the object state as term with adding the identity to it. On the other hand, the four events respectively Open, Withdraw, Deposit and IncreaseInterest are expressed as messages with adding to their arguments the object identity.*

In the associated net, the four 'message' places corresponds to the four events declaration, while the object place allows for capturing the **Account** *object instances. Four transitions reflecting the behaviour of these events are conceived; where, the enabled condition is translated into transition condition.*

The modelling of the **Bank** *template (and its corresponding class) are similarly constructed except for two points: firstly, as shown the figure 1, the* **Bank** *description interacts directly with the* **Account** *class by sending an appropriate message expressed in the* **calling** *clause of the corresponding* TROLL *specification. Secondly, the test of the existence of the two accounts in the transfer process is directly captured by the suitable form* $\langle B|ACset : S1.AN1.S2.AN2.S3\rangle$*; where*

$S1, S2, S3$ *are variables of sort list[nat]* [2] *constructed using the asso. comm. operator denoted by* $\cdot$ [3].

Remark 3.2 *It is worth mentioning that for each transition, the input as well as the output arcs are inscribed just by the* **relevant part** *of the invoked object state(s). For instance, in the DEP(OSIT) transition only the attribute balance is invoked (i.e.* $\langle C|bal : V\rangle$ *in the input arc and* $\langle C|bal : V + M\rangle$ *in the output arcs). As it describes the next, this constitutes the key ideas for a full exhibition of the intra- (and inter-) object concurrency .*

```
obj Bank is
  extending Class-structure .
  protecting money list[nat] nat  string.
  sort Id.Bank Bank TRANSF OPEN-AC .
  subsort Id.Bank < OId .
  (* the Bank object state declaration *)
         op ⟨_|BnM : _, BnN : _, AcSet : _⟩:  Id.Bank  String  nat  List[nat] → Bank
  (* Messages declaration *) .
         op OpenNwAc :  Id.Bank  → OPEN-NAC
         op Transfer :  Id.Bank  nat  nat  money → TRANSFR
  Vars AN, AN1, AN2 :  nat ; S, S1, S2, S3 :  List[nat] ; B : Id.Bank ; M : money endo.


obj Account is
  extending Class-structure .
  protecting money nat  string  Id.Bank  Id.Customer  interest
  sort Id.Account  Account
  sort OPEN-AC  WITHDRW  DEPOSIT  INTRS.
  subsort Id.Account < OId .
  (* the Bank object state declaration *)
         op ⟨_|No : _, bk : _, Hd : _, bal : _, Lmt : _, Ints : _⟩:  Id.Account  nat  Id.bank
         Id.Customer  money  money  Interest→ Account
  (* Messages declaration *) .
         op OpenAc :  Id.Account  Id.Bank  nat  string→ OPEN-AC
         op Wdw :  Id.Account  money → WITHDWR
         op Dep :  Id.Account  money → DEPOSIT
         op IncI : Id.Account  interest → INTRS
  vars H : Id.Customer ; C : Id.Account ; W, D , L : money ; I, NI : Interest
endo.
```

4 Animating and Validating the Translated Specification

4.1 On the Improvement of Object Specifications

Following the CO-Nets semantics and the generic form of rewrite rules given in subsection 3.2, it is not difficult to generate the rewrite rules governing the behaviour of given CO-Net modelling an object specification (dynamic). However, for achieving an *efficient* rapid-prototyping it is crucial to take advantages of the CO-Nets approach in modeling (object) systems not as whole rather as *autonomous* and *interacting*—exclusively through explicit interface— components, conceived as a hierarchy of classes.

To be more precise, instead of generating directly all the rewrite rules associated with different transitions—as a behaviour of the whole system— and after performing formal computation

[2] Declared imported as **protecting** with all the other used data types.

[3] Detail of the corresponding algebraic specification is omitted here(see, [Mes93] for instance).

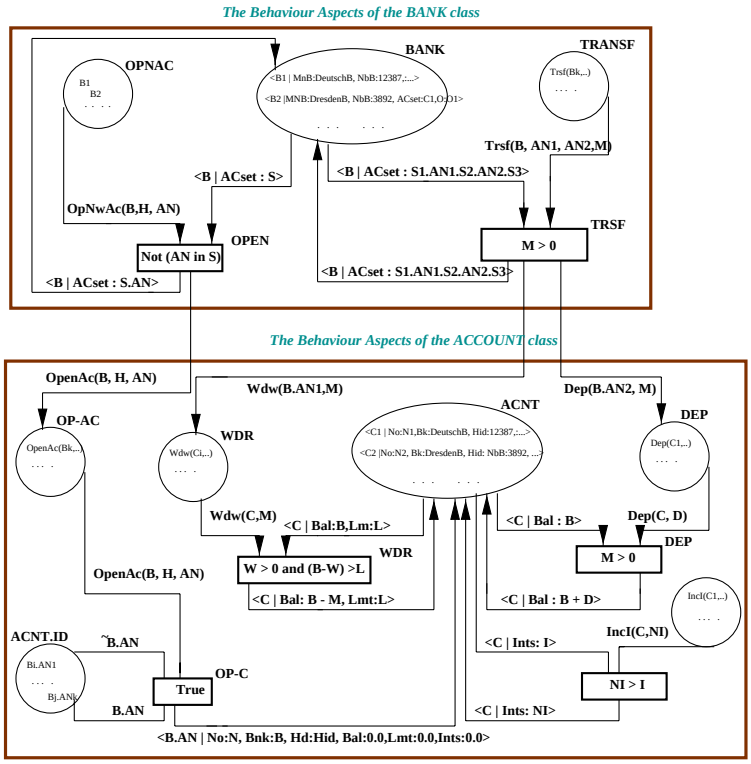
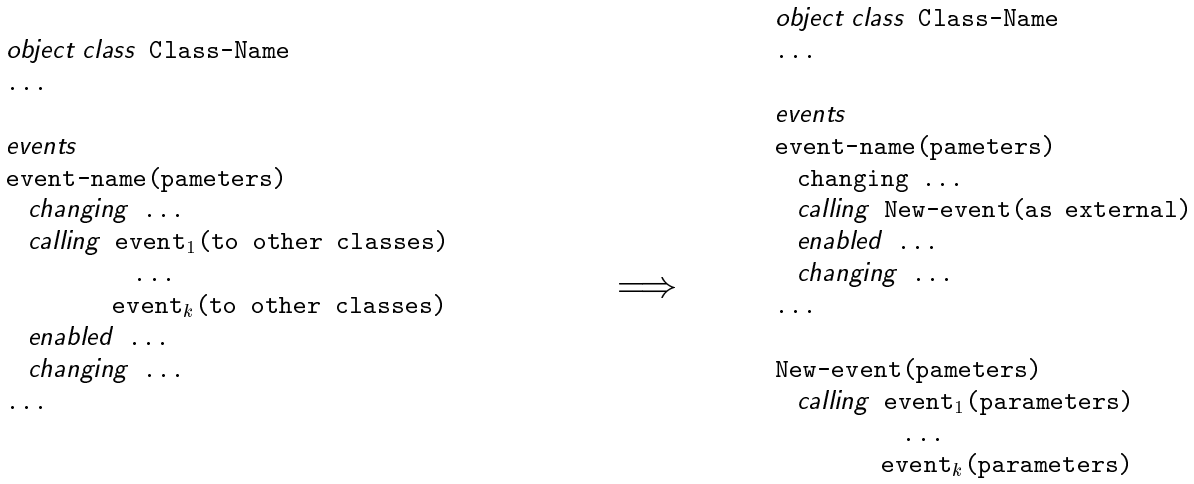


Figure 1: The Modelling of the **Bank** and **Account** Specifications Using CO-Net

on the basis of an initial community of object and message instances, first we propose to improve the object specification by making an explicit distinction between what is local to each class and thus hidden to the outside from what can be exchanged (and eventually modified) with other classes. However, in most cases such distinction is not sufficient for completely respecting the CO-Nets intra- and the inter-component patterns during the translation. Indeed, in the TROLL *calling* clause, it is quite possible to call event(s) declared in *another* class(es), which violates the encapsulation property and thereby the intra-component evolution pattern. For this aim, we propose to introduce (new) intermediate 'external' events that are responsible for calling the 'external' events of other classes. A sketch of this operation may be illustarted as depicted in the schema hereafter; where, for each event behaviour, we have to replace the list of called (external) events (i.e. $event_1, \ldots, event_k$) by a new event (denoted here by *New-event*) of which the behaviour is exactely the calling of these external events.

This TROLL specification rehabilitation has to be expressed in the corresponding CO-Nets by the introduction of new places for such new (external) messages and the introduction of new transitions for capturing their new behaviour (which is just the sending of messages). At the same time we have to drop all the output arcs (associated with the events $event_1, \ldots, event_k$) from the transition that reflects the behaviour of '*event-name*'. In this way, the resulting adapted CO-Net respects both intra- and inter-component evolution and interaction pattern. In other words, in the intra-component pattern only (external or internal) messages declared in the class in question are participating in the evolution, while in the intra-component pattern only external messages possibly from different classes are participating in the communication.



The advantages of such distinction towards a more efficient rapid-prototyping of the (improved) object specification may be highlighted as follows:

- The analysis of the behaviour of a given component (as a class in the simple case), by concurrent rewriting and simultaneous graphical animation from a given initial component community, is *completely independent* from the other components. The efficiency intervenes here by the limited number of manipulated rewriting rules (associated with the transitions) compared to the case where the whole system (i.e. all the components) is analyzed.
- The analysis of the communication and the effect of interaction between different components on the whole system is also achieved independently by taking into account only the interface of each component.

Example 4.1 *Following these guidelines, first we have to distinguish between the local and the observed features in both components defined as classes—namely, the* **Bank** *and the* **Account** *classes. For the observed attributes it is quite logical to conceive the Bank name—in the* **Bank** *class— and the Account number in the class* **Account** *as observed attributes. For the messages, however, as pointed out the declaration of three messages in the* **Account** *class, that the* **Bank** *class address—namely, Open-Ac, Withdraw and Deposit— as external (i.e. imported), is no more sufficient. In fact, even with this declaration the transitions OPEN as well as TRANSF (in the* **Bank** *class) do not respect nor the intra-component neither the inter-component pattern and thereby violate the encapsulation property. This is because both transitions include local messages and at the same time external messages of other classes (here, the* **Account** *class).*

In order to avoid this inconsistency, it suffices to apply the extension described above. In this case, as depicted in figure 2, we have included two new messages named Open-OK and Transf-OK. The second step consists in adding the respective 'external' transitions that should play exactly the role of the dropped (output) arcs from the 'inconsistent' transitions. For instance, the new 'external' transition TRANF allows for expressing any successful transfer by sending the withdraw and deposit messages (that are both external).

4.2 Prototyping (Improved) Object Specifications: The example

Hereafter, first we derive the different rewriting rules associated with both **Bank** and **Account** classes as independent components as well as the rules gouverning their interaction. Then, using

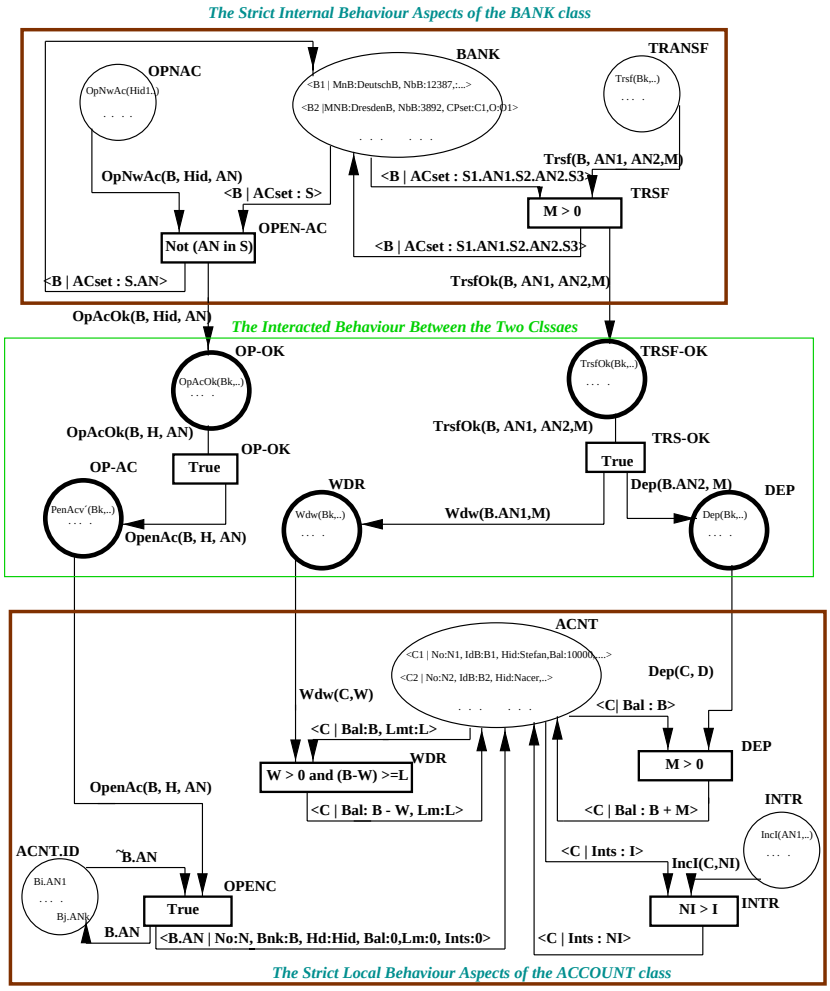


Figure 2: The Bank Specification with Explicit interface.

a simplified **Account** community we show how formal computation based on (intra-and inter object) concurrent rewriting is achieved.

Example 4.2 *By applying the general forms of rewrite rules, it is not difficult to generate the rules governing both* **Bank** *and* **Account** *classes as well as their interaction.*

- **The Behaviour of the Bank class**

  OPEN-AC: [4] $(OPNAC, OpNwAc(B, Hid, AN)) \otimes (BANK, \langle B|ACset : S\rangle))$
  $\Rightarrow (BANK, \langle B|ACset : S.AN\rangle)) \otimes (OP-OK, OpAcOk(H, Ac))\ if\ Not(AN \in S)$

  TRSF: $(BANK, \langle B|ACset : S1.AN1.S2.AN2.S3\rangle) \otimes (TRSF, Trsf(B, AN1, AN2, M))$
  $\Rightarrow (BANK, \langle B|ACset : S1.AN1.S2.AN2.S3\rangle) \otimes (TRSF-OK, TrsfOk(B, AN1, AN2, M))$
  $if\ (M > 0)$

- **The Behaviour of the Account class**

  OPENC: $(OP-AC, OpenAc(B, Hid, AN)) \otimes (ACNT.ID, SetAC)$
  $\Rightarrow (ACNT, \langle B.AN|No : AN, Bnk : B, Hd : H, Bal : 0.0, Lmt : 0.0, Ints : 0.0\rangle) \otimes$
  $(ACNT.ID, SetAC \oplus B.AN)\ if\ Not(B.AN \in SetAC)$

[4] The label corresponds to the transition identifier.

WDR: $(WDR, Wdr(C,W)) \otimes (ACNT, \langle C|Bal : B, Lmt : L\rangle)$
$\Rightarrow (ACNT, \langle Ac|Bal : B - M, Lmt : L\rangle)\ if\ (W > 0) \wedge\ (B - W) \geq L$

DEP: $(DEP, Dep(C,D)) \otimes (ACNT, \langle C|Bal : B\rangle)$
$\Rightarrow (ACNT, \langle C|Bal : B + D\rangle)\ if\ (M > 0)$

INTR: $(INTR, IncI(C,NI)) \otimes (ACNT, \langle C|Ints : I\rangle)$
$\Rightarrow (ACNT, \langle C|Int : NI\rangle)\ if\ (NI > I)$

- **The Interacted Behaviour of the Two classes**

OP-OK: $(OP-OK, OpAcOk(B,H,AN)) \Rightarrow (OP-AC, OpenAc(B,H,AN))$

TRSF-OK: $(TRS-OK, TrsfOk(B, AN1, AN2, M))$
$\Rightarrow (WITHDW, Wdw(AN1, M)) \otimes (DEP, Dep(AN2, M))$

In order to show how such rewrite rules can be applied to an initial object community with message instances), we restrict ourselves to a simplified account community as depicted in figure 3.

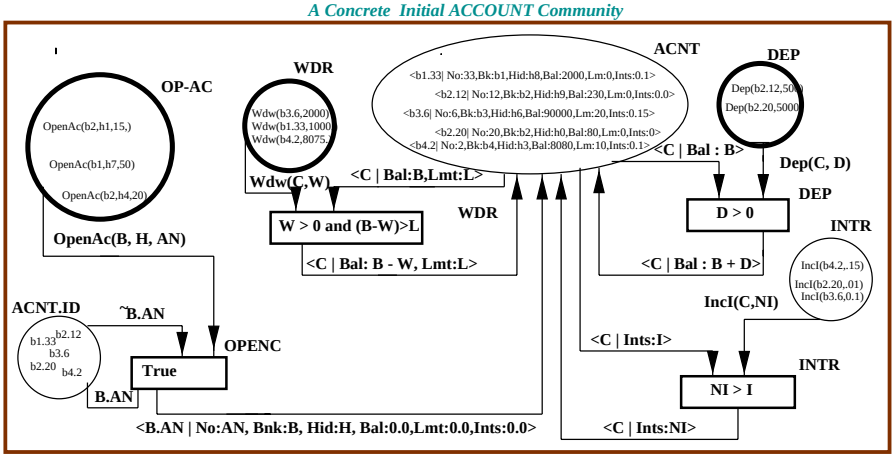


Figure 3: A Simplified (initial) Account Community.

This CO-Net initial state corresponds formally to:

$(OP-AC, OpenAc(b1,h1,50) \oplus OpenAc(b2,h3,15) \oplus OpenAc(b2,h4,20)) \otimes (WDR, Wdr(b3.6,2000) \oplus Wdr(b1.33,1000) \oplus Wdr(b4.2,8075)) \otimes (DEP, Dep(b2.12,50) \oplus Dep(b2.20,5000)) \otimes (INTR, IncI(b4.2,0.15) \oplus IncI(b2.20,0.1)) \otimes (ACNT.ID, b2.12 \oplus b1.33 \oplus b3.6 \oplus b2.20 \oplus b4.2) \otimes (ACNT, \langle b1.33|No : 33, Bk : b1, Hid : h8, Bal : 2000, Lmt : 0, Ints : 0.1\rangle \oplus \langle b2.12|No : 12, Bk : b2, Hid : h9, Bal : 230, Lmt : 0, Ints : 0.1\rangle \oplus \langle b3.6|No : 6, Bk : b3, Hid : h6, Bal : 90000, Lmt : 20, Ints : 0.15\rangle \oplus \langle b2.20|No : 20, Bk : b2, Hid : h0, Bal : 80, Lmt : 0, Ints : 0.1\rangle \oplus \langle b4.2|No : 2, Bk : b4, Hid : h3, Bal : 8080, Lmt : 10, Ints : 0.1\rangle)$

Using the associativity and the commutativity of both $\oplus$ and $\otimes$ and the splitting/merging axiom, we will deduce the CO-Net state depicted in figure 4 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 ACNT w.r.t. the left hand sides of the corresponding rewrite rules. This yields the following sub-state :

$(ACNT, \langle b1.33|No : 33, Bk : b1, Hid : h8, Ints : 0.1\rangle \oplus \langle b1.33|Bal : 2000, Lmt : 0\rangle \oplus \langle b2.12|No : 12, Bk : b2, Hid : h9, Lmt : 0, Ints : 0.1\rangle \oplus \langle b2.12|Bal : 230\rangle \oplus \langle b3.6|No : 6, Bk : b3, Hid : h6, Ints : 0.15\rangle \oplus \langle b3.6|Bal : 90000, Lmt : 20\rangle \oplus \langle b2.20|No : 20, Bk : b2, Hid : h0, Lmt : 0\rangle \oplus \langle b2.20|Bal : 80\rangle \oplus \langle b2.20|Ints : 0.1\rangle \oplus \langle b4.2|No : 2, Bk : b4, Hid : h3, Bal : 8080, Lmt : 10, Ints : 0.1\rangle \oplus \langle b4.2|Bal : 8080, Lmt : 10\rangle \oplus \langle b4.2|Ints : 0.1\rangle)$

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

$(OP-AC, OpenAc(b2, h1, 12)) \otimes (OP-AC, OpenAc(b1, h3, 34)) \otimes (OP-AC, OpenAc(b2, h4, 20)) \otimes (ACNT.ID, b2.12 \oplus b1.33 \oplus b3.6 \oplus b2.20 \oplus b4.2) \otimes (WDR, Wdr(b3.6, 2000)) \otimes (ACNT, \langle b3.6 | Bal : 90000, Lmt : 20\rangle) \otimes (WDR, Wdr(b1.33, 2000)) \otimes (ACNT, \langle b1.33 | Bal : 20000, Lmt : 0\rangle) \otimes (WDR, Wdr(b4.2, 8075)) \otimes (ACNT, \langle b1.33 | Bal : 8080, Lmt : 0.1\rangle) \otimes (DEP, Dep(b2.12, 500)) \otimes (ACNT, \langle b2.12 | Bal : 230\rangle) \otimes (DEP, Dep(b2.20, 5000)) \otimes (ACNT, \langle b2.20 | Bal : 80\rangle) \otimes (INTR, IncI(b4.2, 0.15)) \otimes (ACNT, \langle b4.2 | Ints : 0.1\rangle) \otimes (INTR, IncI(b2.20, 0.01)) \otimes (ACNT, \langle b2.20 | Ints : 0\rangle) \otimes \otimes (ACNT, \langle b1.33 | No : 33, Bk : b1, Hid : h8, Ints : 0.1\rangle \oplus \langle b2.12 | No : 12, Bk : b2, Hid : h9, Lmt : 0, Ints : 0.1\rangle \oplus \langle b3.6 | No : 6, Bk : b3, Hid : h6, Ints : 0.15\rangle \oplus \langle b2.20 | No : 20, Bk : b2, Hid : h0, Lmt : 0\rangle \oplus \langle b4.2 | No : 2, Bk : b4, Hid : h3\rangle$[5])

By concurrently applying the rewrite rules associated with the **Account** *class using the replacement inference rule (of the rewriting logic [Mes92], see the Appendix) to different sub-terms dealing either with the same object state like the withdraw with the increase of interest of the object $b4.2$ and the deposit with the increase of interest of the object $b2.20$ or 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.*

$(OP-AC, OpenAc(b2, h4, 20)$[6]$) \otimes (ACNT.ID, b2.12 \oplus b1.33 \oplus b3.6 \oplus b2.20 \oplus b4.2 \oplus b2.15 \oplus b1.50) \otimes (WDR, Wdr(b4.2, 8075)) \otimes (INTR, IncI(b3.6, 0.1))(ACNT, \langle b1.33 | No : 33, Bk : b1, Hid : h8, Bal : 1000, Lmt : 0, Ints : 0.1\rangle \oplus \langle b2.12 | No : 12, Bk : b2, Hid : h9, Bal : 730, Lmt : 0, Ints : 0.1\rangle \oplus \langle b3.6 | No : 6, Bk : b3, Hid : h6, Bal : 89000, Lmt : 0.15, Ints : 0.15\rangle \oplus \langle b2.20 | No : 20, Bk : b2, Hid : h0, Bal : 5080, Lmt : 0, Ints : 0.2\rangle \oplus \langle b2.15 | No : 15, Bk : b4, Hid : h3, Bal : 0, Lmt : 0, Ints : 0\rangle \oplus \langle b1.50 | No : 50, Bk : b4, Hid : h3, Bal : 0, Lmt : 0, Ints : 0\rangle)$

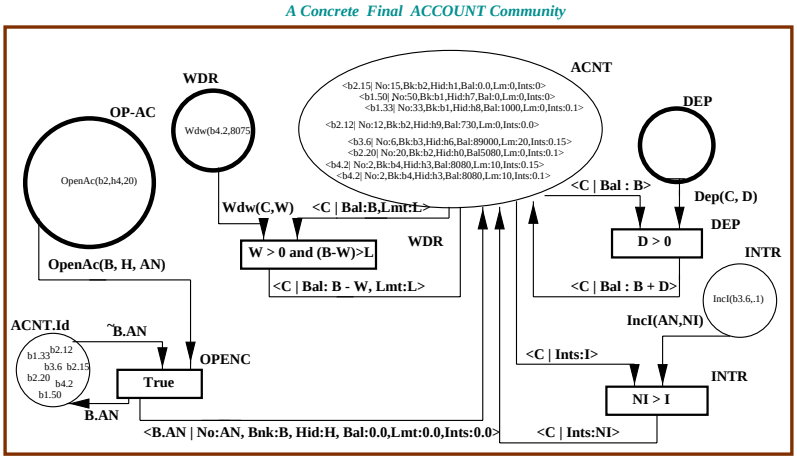


Figure 4: The Final state of the simplified Account community

5 Conclusion

The prototyping phase in developing (distributed) information systems, following an object oriented approach, is one of the crucial tasks for bridging the gap between a reliable object specification and secure and efficient implementation of such systems. Aiming to significantly contribute for achieving this difficult task, we proposed in this paper a two-steps approach

[5] The non invoked parts of different object states in the effect of messages.

[6] Because the object state identified by $b2.20$ already exists.

based on two formal frameworks. Firstly, information systems are specified using widely accepted object specification languages, namely the TROLL language, which propose a variety of constructions and abstraction mechanisms. Secondly, the resulted specification is translated into the CO-Nets approach, an appropriate object Petri net-based model that we are developing, where animation and concurrent formal computation is achieved using advantages of Petri nets and the CO-Nets semantics based on rewriting logic.

The proposed translation from object specification into CO-Nets, that we have illustrated through a simplified example, was shown to be natural and straightforward. Moreover, for efficiently rapid-prototyping the resulted CO-Net specification, we have proposed to enhance the original object specification by distinguishing between local and external features of each class composing the specification. It allowed us particularly to derive (intra- and inter-object) true concurrent formal computation, based on rewriting techniques, visualized by simultaneous graphical animation for each class separately. On the other hand, the interaction between the different class is validated without violating the (explicitly declared as) encapsulated features of each classes which allows to easily understand the different interaction between the components.

However, due to the increasing in size and space complexity of real-world applications, we are conscious that after this first step much work remains ahead. Firstly, we have to generalize the proposed CO-Nets approach for dealing with all constructions in object specifications and particularly in-large abstractions mechanisms (like specialization and aggregation). Secondly, more complex case studies have to be undertaken in order to find out the different sorts of errors and missing that can be detected by the proposed prototyping. Besides that, we are working for a more advanced version of CO-Nets that allows for dealing with 'run-time' evolution of object specification as it has been shown for TROLL specifications using the evolving temporal logic [CRSS98].

References

[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, Karlsrule, Germany, March 1999. B.G. Teubner-Verlag.

[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*, volume 1626, pages 381–395, Berlin, 1999. Springer-Verlag.

[CRSS98] S. Conrad, J. Ramos, G. Saake, and C. Sernadas. Evolving Logical Specification in Information Systems. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 7, pages 199–228. Kluwer Academic Publishers, Boston, 1998.

[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.

[FW93] R. B. Feenstra and Wieringa. LCM 3.0: A Language for describing Conceptual Models. Technical report, Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam, 1993.

[JSHS96] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. TROLL – A Language for Object-Oriented Specification of Information Systems. *ACM Transactions on Information Systems*, 14(2):175–211, April 1996.

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

[Mes93] 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.