DYNAMIC INTERACTION OF INFORMATION SYSTEMS
*Weaving Architectural Connectors on Component Petri Nets**

Nasreddine Aoumeur, Gunter Saake
ITI, FIN, Otto-von-Guericke-Universität Magdeburg, Postfach 4120, D–39016 Magdeburg, Germany

Kamel Barkaoui
Laboratoire CEDRIC, CNAM, 292 Saint Martin, 75003 Paris - France

Keywords: High-Level Nets, rewriting logic, specification/validation, architectural techniques, dynamic evolution.

Abstract: Advances in networking over heterogenous infrastructures are boosting market globalization and thereby forcing most software-intensive information systems to be fully distributed, cooperating and evolving to stay competitive. The emerging composed behaviour in such interacting components evolve dynamically/rapidly and unpredictably as market laws and users/application requirements change on-the-fly both at the coarse- type and fine-grained instance levels.
Despite significant proposals for promoting interactions and adaptivity using mainly architectural techniques (e.g. components and connectors), rigorously specifying / validating / verifying and dynamically adapting complex communicating information systems both at type and instance levels still remains challenging. In this contribution, we present a component-based Petri nets governed by a true-concurrent rewriting-logic based semantics for specifying and validating interacting distributed information systems. For runtime adaptivity, we enhance this proposal with (ECA-business) rules Petri nets-driven behavioral connectors, and demonstrate how to dynamically weaving them on running components to reflect any emerging behavior.

1 INTRODUCTION

Advances in networking over heterogenous infrastructures are forcing most software-intensive systems to be fully distributed, cooperating and evolving in facing the fierce struggle to stay competitive. This situation is more acute for contemporary information systems where market globalization and volatility are pressing business laws, policies (makers) and users requirements to adapt/evolve on-the-fly, unpredictably and rapidly. These facts have been urging organizations to shift from their traditional integrated form with centralized control to *loosely-coupled* networked applications owned and managed by diverse business partners.

With the huge difficulties of directly and satisfactory implementing such complex communicating systems even with the availability of advanced networked infrastructures and standards (e.g. Web-Services and Service-oriented computing), more research efforts are nowadays devoted to foundational early phases of specification, validation and verification. Additionally, the tedious challenges of adaptivity must be tackled at these requirement early phases; otherwise any initial certified specification becomes rapidly outdated and useless, where afterwards the standing-alone programmers will be on charge of uncontrolled/unwished/untractable maintenance.

Architectural techniques over component-orientation belong to the mostly investigated and appropriate software-engineering conceptual means to address the adaptivity through their first-class explicit connectors (Cheng and Garlan, 2001; Szyperski, 1998) and their reconfigurations. Nevertheless, due to the growing complexity and volatility of such complex interacting software-intensive systems, it seems we are still far from a satisfactory architectural-based proposal. Indeed, existing proposals either focus on the coarse-grained component level (adding/removing/replacing components) while ignoring the component instance level (e.g. internal statefull functionalities) or vice-versa. As will be

*This research is partially supported by a DFG (German Science Foundation) Project Under Contract SA 465/31-1.

Aoumeur N., Saake G. and Barkaoui K. (2007).
DYNAMIC INTERACTION OF INFORMATION SYSTEMS - Weaving Architectural Connectors on Component Petri Nets.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - ISAS*, pages 152-158
DOI: 10.5220/0002402501520158

demonstrated in this paper both levels are vital for expressing dynamic change and evolution.

With respect to information systems, the authors are not aware of any approach that handles dynamic adaptivity by respecting cross-organizational business rules (Wan-Kadir and Loucopoulos, 2003) at the architectural level at both the type and instance levels. Business rules are the main driving force for reshaping inter-organizations goals, policies and functioning/regulations and are therefore rapidly evolving to enhance the competitiveness.

This paper puts forwards a (business-) rule-based architectural approach for specifying / validating and dynamically adapting complex gross-organizational information systems. Methodologically, given a (UML-based) semi-formal description of the applications, we first propose a formal specification based on a variant of component-based Petri nets, we introduced in (Aoumeur and Saake, 2002). In this so-called CO-NETS framework, IS components are conceived as a hierarchy of modular classes we explicit observed interfaces. For validating such components, a true-concurrent rewriting logic-based is proposed for governing different transitions behavior. We further enrich CO-NETS with the concept of rule-driven architectural connector behavior. Such architectural connectors reflect different ubiquitous cross-organizational and thus inter-component domain business rules. As we separately and explicitly specify such connectors, we are able to dynamically weaving the right rules when required to reflect the enforcement of new/modified policies, laws and other requirements. Crucial to point out is that the proposed dynamic weaving is non-intrusive, that is, we do not delve inside component functionalities or change/adapt any existing component internal behavior.

The rest of the paper is organized as follows. In the second section we review the main CO-NETS features and illustrate them with a simplified banking system. The third section introduce the concept of (business) rule-based architectural connectors at a semi-formal level. The fourth main section focuses on the modeling of such connectors within CO-NETS and how they are dynamically woven on interacting components. This paper is closed by some remarks and insights about future extensions. We should note that the paper's presentation is kept at an informal level with some hints when required to the formal counterpart.

2 CO-NETS COMPONENTS: OVERVIEW WITH ILLUSTRATIONS

2.1 CO-NETS Component Structure

The component structure defines the structure of object states and operations to be accepted by such states. The CO-NETS signature that we are proposing can be informally described as follows:

- Object states are algebraic terms of the form:

  $\langle Id | l_1 : v_1, .. l_k : v_k, f_1(Id), .., f_l(Id), s_1 : v'_1, .., s_t : v'_t \rangle$

  Where Id represents an observed object identity. $l_1, .., l_k$ denotes local attributes with respective current values $v_1, .., v_k$. The attributes $f_1(Id), .., f_l(Id)$ are considered as hidden (defined in a co-algebraic way). Observed attributes can be also be defined such as $s_1, ..., s_t$.
- To separate at any time local attributes from observed ones and exhibit intra-concurrency, we introduce a simple deduction rule we call 'object-state splitting / merging' rule that permits to split (resp. recombine) the object state as required.
- We also make an explicit distinction between *internal* messages and *external* as imported / exported messages.

With respect to this perception each template signature is henceforth endowed with an explicit interface composed of observed attributes and messages, and that we subsequently refer to as (basic) *component* signature.

2.1.1 The Account and Customer Components

We assume having accounts and customers that we naturally regarded as two separate yet interacting components. For instance, for the account component signature, each (current) account is composed of: a balance (shortly `bal`) as observed, and a minimal limit (`lmt`), a boolean value (`Red`) valuated to true when below limit.

```
obj account is
  extending object-state .
  protecting Account-data .
  sorts Acnt .
  subsort Id.Acnt < OId .
  subsort DEB CRD TRS < Obs_Msg.
  subsort ChgL < Loc_Msg.
  subsort loc_Acnt obs_Acnt < Acnt < object .
  (* attributes as functions*)
    op Pin :  Id.Acnt → string .
  (* Local attributes *)
op ⟨_ | Bal : _, Lmt : _, Hs : _⟩ :  Id.Acnt
```

```
        Money  Money  History → loc_Acnt.
  (* observed attributes *)
op ⟨_ | Hd : _⟩ :  Id.Acnt OId → obs_Emp .
  (* Local messages *)
op Chgl :  Id.Acnt  Money → ChgLm .
  (* observed messages *)
op Deb :  Id.Acnt  Money  Date → DEB.
op Crd :  Id.Acnt  Money  Date → CRD.
op Trs :  Id.Acnt  Id.Acnt  Money  Date → TRS.
  vars B, L, W, D : Money ; C : Id.Acnt .
endo .♦
```

2.2 CO-NETS Component Behaviors

On the basis of a given component structure, we define the notion of component specification as a CO-NET in the following straightforward way.

- CO-NETS places are precisely defined by associating with each message declaration or method one (message) place, that is, such messages places contain associated message instances sent or received by objects but not yet performed. Also, with each object sort a (object) place is associated.
- CO-NETS transitions reflect the effect of messages on object states they are addressed to. Conditions may be associated to them restricting their application.

Application to the Account and Customer Components. The associated CO-NETS account component is depicted in the left-hand side of Figure 1. This component is composed of current accounts as a superclass and saving accounts as a subclass (where the interest could be increased through `Tinc` and money be moved from a saving account to a current account using `Tmvt`). The right-side of this figure represents the customer component. As described above, in this net in addition to the object place `ACNT` that contains all account instances three message places namely `ChgL`, `DEB` and `CRD` have been conceived.

2.3 CO-NETS Component Interactions

By taking benefit of explicit component interfaces (i.e. observed attributes and import / export messages), we present how interacting behaviorally different components, leading to cooperative information systems composed of several truly distributed and independent yet *cooperating* CO-NETS components. As depicted in Figure 2, the general schema of 'external' transitions reflect this communication where Just relevant *external* parts of component states enter into contacts with observed messages and resulting (under conditions) in: (1) the messages $ms_{i_1},..,ms_{i_r}$ being consumed; (2) states of some external parts of objects participating in the communication being changed; and (3) new external messages (that may involve deletion/creation ones) being sent to objects at different components, namely $ms_{h_1},..,ms_{h_r}$.

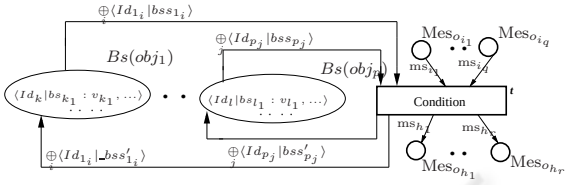

Figure 2: The Inter-component interaction pattern.

Application to the Running Example. In Figure 3 the interaction of the two components is described using exclusively their observed features. We note here that besides those explicitly defined as observed features, attributes as functions are also observed, but their contents cannot be accessed either at local or at this interaction level. This concerns in particular the *Pin* attribute as function. Here are some comments on this observed interaction behaviour. The transition `Twrd`, for instance, captures the sending of a withdraw order from the customer, namely `C-Deb(C,A,M)`, that have to be received by the withdraw message in the concerned account (i.e. the account of this customer as account hold). This means that we have to select from the observed account attribute through the read-arc the inscription: $\langle A|Hd:C\rangle$.

2.4 Animating and Validating CO-NETS Specifications

One of the most advantage of the CO-NETS approach is its operational semantics expressed in rewriting logic; moreover, by introducing the state splitting / recombining axiom there is a natural exhibition of intra-object concurrency. More precisely, each transition is governed by a corresponding rewrite rule interpreted in rewrite logic (Meseguer, 1992) (or more precisely an instantiation of this logic we refer to as CO-NETS rewrite theory. The main ideas consist in: (1) associating with each marking *mt* its corresponding place p as a pair (p,mt); (2) introducing a new multiset generated by a union operator we denote by $\otimes$ for reflecting CO-NETS states as multisets over different pairs (p_i,mt_i), , that is, a CO-NETS state is described as $(p_1,mt_1)\otimes(p_1,mt_2)\otimes...$; (3) allowing the distributivity of $\otimes$ over $\oplus$ for exhibiting a maximal of concurrency, that is, if mt_1 and mt_2 are two marking multisets then we always have: $(p,mt_1\oplus mt_2)=(p,mt_1)\otimes(p,mt_2)$; (4) enabling object states' split-

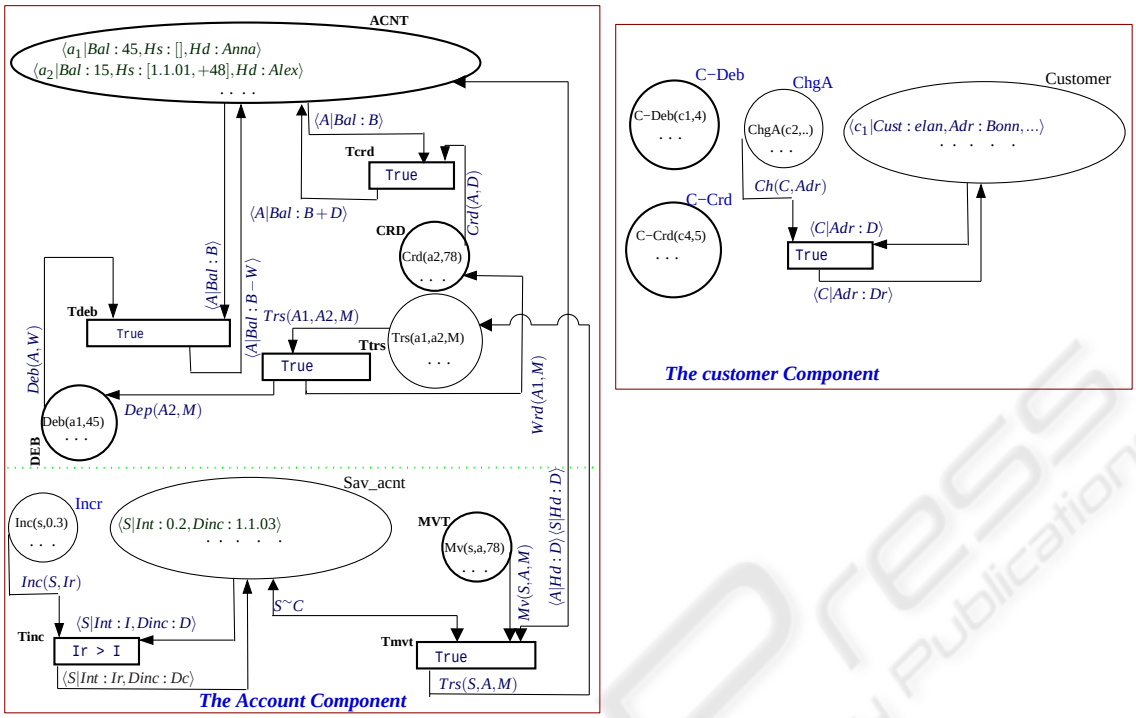

Figure 1: A Simplified Banking specification within CO-NETS.

ting and recombining at a need for exhibiting intra-object concurrency.

Application to the Account Component. By applying the afore-described general form of rewrite rules, it is not difficult to generate the transition rules associated with the CO-NETS specification of the banking example. The rewrite rule associated with the debit transition `Tdeb` is:

Tdeb $:(DEB, Deb(C,W)) \otimes (ACNT, \langle C|Bal : B\rangle)$
$\Rightarrow (ACNT, \langle C|Bal : B-W\rangle)$

3 ECA-RULE BASED ARCHITECTURAL CONNECTORS

Architectural connectors must generally endowed with: (1) *roles* expressing functional requirements from candidate components to interact; and (2) *glues* for expressing the behavior governing such inter-component interaction. As we are targeting complex information systems are main domain application, instead of just exchanging and ordering messages at this interaction level we propose a more knowledge-intensive behavioral interaction patterns expressed in terms of *business rules*. For such cross-organisations business rules, we propose the most adopted form which the Event-Conditions-Actions (ECA) paradigm. More precisely the general pattern, we decide to follow for expressing inter-component interactions is as follows:

ECA-behavioral glue <*glue-Identity*>
interface participants <*list-of-participants*>
invariant <*possible interaction constraints to respect*>
constants/attributes/operations
<*extra-required elements for the interaction*>
interaction rules: <*Rule-Name*>
at-trigger <`(set-of-)events`>
under <`conditions`>
reacting <`set-of-actions`>

This behavioral rule requires of course from different participating entities explicit interfaces including different events, messages and other properties (such as constants, variables ,etc). When needed, we explicitly specify such interfaces before giving this glue.

3.1 ECA Behavioral Glues: Illustration

To stay competitive banking systems are offering different incitive packages for their customers, ranging from simple agreed-on contracts (e.g. different formulas for withdrawing / transferring moneys) to highly sophisticated complex offers (i.e. staged hous-

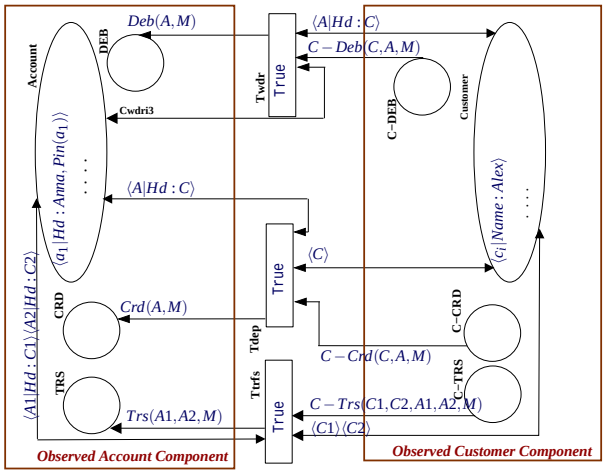

Figure 3: The Customer account CO-NETS components interaction.

ing loans, mortgages, etc.) depending on their profiles, trust, experiences, etc.

Let us illustrate simple cases of customer-bank agreements using our running example, through two customer-tailored variants of withdrawals. The usual case for any ordinary customer is to check what is called standard withdrawal, where the customer has to be the owner of a corresponding account and the withdrawal amount should not go beyond the current balance.

```
ECA-behavioral glue Std-Withdraw
participants Acnt: Account; Cust: Customer
invariants Cust.own(Acnt) = True

interaction rule : Standard
at-trigger Cust.withdraw(M)
   under (Acnt.bal() ≥ M)
   acting Acnt.Debit(M)
end Std-withdraw
```

A more flexible withdrawal for moderately previleged customers is to endow them with a credit those amount depends on profile and trust. Customers enjoying such agreements can now withdraw amounts going beyond the current balance. The modelling of such flexible agreed-on withdrawal takes the following slightly modified behavioral glue.

```
ECA-behavioral glue VIP-Withdraw
participants Acnt: Account; Cust: Customer
attribute Cust.credit : Money
invariants Cust.own(Acnt) = True

interaction rule : VIP
at-trigger Cust.withdraw(M)
   under (Acnt.bal() +Cust.credit ≥ M)
   acting Acnt.Debit(M)
end VIP-withdraw
```

4 MODELLING ECA BEHAVIORAL CONNECTORS AS EXTENDED CO-NETS

With CO-NETS capabilities in capturing statefull components behavior, we present in the following how ECA-driven architectural connectors enhance these potentials towards more dynamic adaptivity and evolution. Following the same intuitive guidelines for constructing CO-NETS components from informal component-based applications, the modelling steps for integrating such architectural connectors into already specified CO-NETS components could sketched in the following. First, we have derive from a given ECA-based architectural connector description, a more precise corresponding component signature specification by algebraically specifying different properties (attributes, messages, events, etc.). Secondly, by gathering different connector attributes and participants into states, we then associate such each state type a corresponding place and with each messages and operations also a place. This results in the skeleton of the Petri net for such architectural connectors. Finally, we have to inject the rules into such skeleton by assigning conditions to transition conditions, events as input arc-inscriptions and actions as output ones. In a more detail, these translating steps could be explained as follows:

1. Define architectural connector structure algebraically using the CO-NETS component signature pattern. That is, first, gather all component participants interface identities with possible other attributes into a glue state type-as-tuple.
2. Specify all involved messages, events, constants

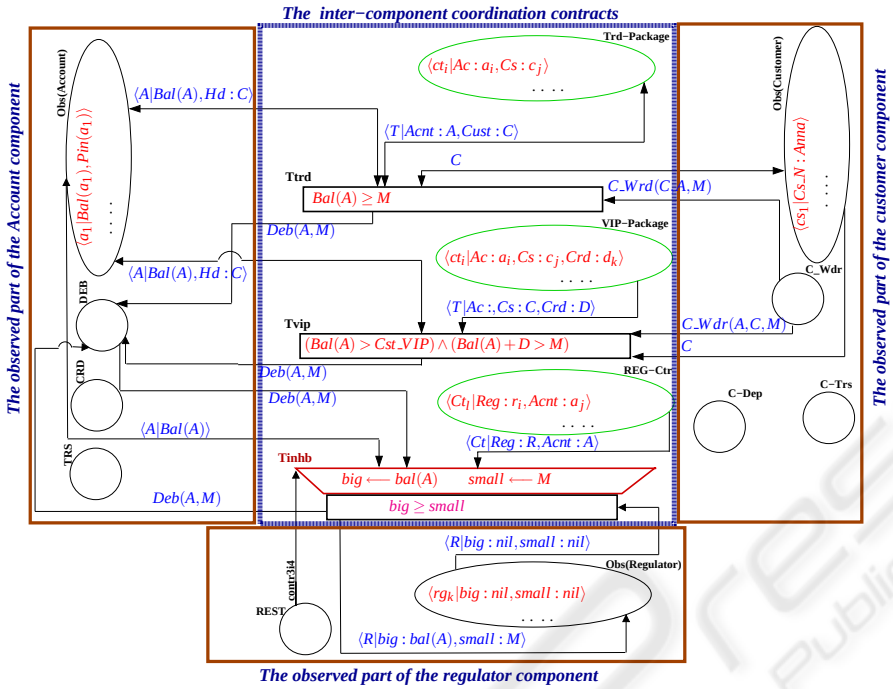

Figure 4: ECA-Architectural connectors woven on the CO-NETS banking system.

and invariants in a given architectural connector using a precise algebraic setting.

3. Associate with each state type a given "glue-state" place, and with each local messages and events a given place.
4. The transitions for architectural connectors have to be constructed for reflecting the different ECA-rules. The general pattern for such interacting transitions is to be conceived following now the new general pattern we depict in Figure 4. This pattern expresses the idea of dynamically weaving exogenous behavior of interacting components. Thats is: (1) Involved component interfaces in a given ECA architectural behavior are captured using observed state parts (places) from such CO-NETS components and imported/exported messages (places); (2) These interface elements (e.g. observed states and messages) enter into contact to reflect the ECA-rule in question; (3) To allow renaming and refinement while weaving such architectural behavior, we add in the interaction transition new boxes for eventual term assignments.

4.1 Illustration Using the Running Example

We approach the two already conceived architectural within the CO-NETS framework following the above steps. That is, as shown below first their algebraic signatures are derived—the ECA-rule themselves are skipped and replaced by just informal text as they are to be specified through the architectural Petri net afterwards.

```
obj  Std-Withdrawal .
  extending object-state .
  using Id.Acnt  Id.Cust .
  subsort StdGlue  < object .
  (* StdGlue state *)
     op ⟨_ | Acnt : _,Cust : _⟩ :  Id.StdGlue
       Id.Acnt  Id.Custm→ StdGlue.
  vars Z : Money ; H : Id.Cust .
  vars A : Id.Acnt, C: Id.Cust, Gl :  Id.StdGlue .
endContract . ♦

obj  VIP-Withdraw .
  extending object-state .
  using Id.Acnt  Id.Cust .
  subsort VIPglue  < object .
  (* VIPGlue state *)
     op CST_VIP : → Nat
     op ⟨_ | Acnt : _,Cust : _,Crd(_) : _⟩ :  Id.VIPGlue
       Id.Acnt  Id.Custm  Real → VIPGlue.
  vars Z : Money ; H : Id.Custm .
  vars A : Id.Acnt, C: Id.Cust, Ct :  Id.VIPGlue .
```

```
  eq CST_VIP = Nat_Value
endContract . ♦
```

5 CONCLUSIONS

For *reliably* developing complex concurrent and dynamically evolving information systems, we extended component-based Petri nets with ECA-compliant behavioral architectural connectors. We shown how to incrementally incorporate different state-full connectors those behaviors being extracted from externalized cross-organizational business rules. Both graphical animations and concurrent symbolic computations using soundly enriched rewriting logic are possible for validating the conceived evolving conceptual model.

This first step towards enhancing component-based formalisms with dynamic inter-component explicit and state-full interactions in true-concurrent distributed environments has to be further worked out for resulting in a complete methodology with adequate software tools supporting it.

ACKNOWLEDGEMENTS

The authors are grateful to the insightful comments and suggestions of the reviewers.

REFERENCES

Aoumeur, N. and Saake, G. (2002). A Component-Based Petri Net Model for Specifying and Validating Cooperative Information Systems. *Data and Knowledge Engineering*, 42(2):143–187.

Cheng, S. and Garlan, D. (2001). Mapping architectural concepts to uml-rt. In *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001)*.

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

Szyperski, C. (1998). *Component Software : Beyond Object-Oriented Programming*. Addision-Wesley.

Wan-Kadir, W. and Loucopoulos, P. (2003). Relating Evolving Business Rules to Software Design. *Journal of Systems Architecture*.