UML-driven Information Systems and their Formal Integration Validation and Distribution*

Nasreddine Aoumeur and Gunter Saake

Otto-von-Guericke-Universität Magdeburg
Institut für Technische und Betriebliche Informationssysteme
PF 4120, D–39016 Magdeburg, Germany

Abstract. Being the de-facto standard (object-oriented-OO) method(-logy) for software-intensive systems development, UML with its different diagrams and supporting tools represent nowadays the mostly adopted software-engineering means for information systems (IS). Nevertheless, due to this wide-acceptance by all organization stakeholders several enhancements at the modelling level are required before adventuring into further implementation phases. The coherence and complementarity between different diagrams have to tackled; On the basic of such endeavored coherent global view, the consistency and validation of the whole IS conceptual models are to undertaken; and last but not least as current information systems are mostly networked and concurrent, UML-driven have to cater for intrinsic distribution and concurrency.

To leverage UML-driven IS conceptual modelling towards these crucial enhancements, we propose a semi-automatic intermediate abstract phase before any implementation, we govern by a rigorous component-based operational and visual conceptual model. Referred to as CO-NETS, this specification/validation formalism is based on a tailored formal integration of most OO concepts and mechanisms enhanced by modularity principles into a variant of algebraic Petri Nets. For rapid-prototyping purposes, CO-NETS is semantically interpreted into rewriting logic. This UML-CO-NETS proposal for distributed IS rigorous development is illustrated through a non-trivial case-study for production systems.

1 Introduction

With the networking of most organizations into cross-organizational giants, where emerging collaborations and interactions and are the driving forces, information systems (IS) as the ”digitalized” accurate mirror of these new organizational interaction-driven realities are consequently under extreme pressure to keep in pace with these advances. For a reliable development of today’s IS integrated semi-formal and formal OO modelling frameworks have been widely adopted, providing powerful abstraction mechanisms for intrinsically integrating and building-on structural and behavioral features (e.g. OMTROLL [1], fOOSE [2]).

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

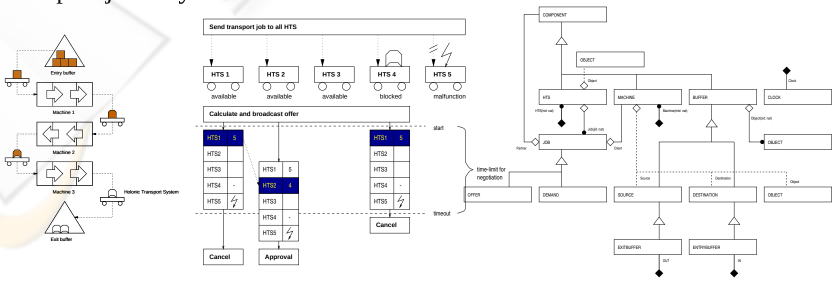
This contribution fits within these efforts, and it proposes to extend the semi-formal UML [3] method with more formality, global coherence, validation and distribution. We thus propose a validation based on a sound framework which allows us to shift from UML-driven IS conceptual modelling towards a fully distributed specification consisting of cooperative components. We hence extend UML-diagrams for fulfilling more advanced requirements including: (1) Intra- as well as inter-object concurrency; (2) synchronous and asynchronous communication; (3) specification of components as hierarchy of classes; (4) Explicit inter-component interactions without violating encapsulated part of each component; and (5) graphical animation accompanied by formal concurrent reasoning.

The proposed framework for this advanced specification/validation phase is a new form of component-based Petri nets model that we interpret in rewriting logic. Referred to as CO-NETS, this specification/validation formalism [4] is mainly characterized by the following key features: (1) To promote inter-communication and autonomy, CO-NETS explicitly distinguishes between local aspects and external ones in a given a component; (2) To enhance Behavior-centricity, we interact components through their explicit interfaces; (3) CO-NETS semantics is interpreted in rewriting logic[5] which is a *true-concurrent* operational semantics particularly allowing rapid-prototyping.

The rest of this paper is organized as follows. The second section informally introduces the case study through which we illustrate the different phases of our proposal. The third section presents UML class- object- and state-diagrams as well as OCL constraints in the form of pre- and post-conditions. In the third section we review the main CO-NETS features we focus on subsequently. In the main section, we present our ideas for shifting from these four diagrammatical views into a corresponding unique coherent CO-NETS specification. The fourth section deals with the validation phase by illustrating how transition rewrite rules are automatically derived. We finally close this paper with some concluding remarks and an outlook on future extensions.

2 The Holonic Transport Case Study

As depicted in the left-hand side of Figure below, we concerned with a part of a production system where specific work pieces (OBJECT) are processed by three different machines (M_x) in a specific order. The transport of the work pieces is carried out by so called ”Holonic Transport Systems” (HTS) which are mobile robots. Machines have to initiate JOBs to execute transports of work pieces. These HTS have to be *concurrent* and *self-organizing* in a way that they locally decide by competing offers which transport job they execute.



Additionally, there are two buffers in that scenario. The first one (IN) provides “fresh” (entirely unprocessed) work pieces, whereas completely processed work pieces are delivered into the second one (OUT). Every machine consists of local entry and exit buffers which may store unprocessed/processed work pieces. Each time an object is removed from the local entry buffer or inserted into the local exit buffer, the machine calls for a HTS to deliver a new or remove a processed work piece. This way, we can distinguish between demand- (DEMAND) and offer-jobs (OFFER).

A possible scenario is as like. A machine M_x generates a request and sends it together with a time-stamp via broadcast to all HTS. When a HTS receives the request, it first checks the current time and compares it with the requests time-stamp. If the elapsed time lies below a certain time-limit, the HTS may proceed the negotiation process. If a HTS is currently unable to perform the requested job for some reason it sends an *unable* message to all other HTS and aborts the negotiation. Otherwise the offer is calculated, sent to the other HTS and entered into an internal cost comparison table (CCT). Until the time-limit is reached, all HTS collect the offers of the other HTS and enter them into their CCT.

3 UML-Specification of the Case-study

We model here the *class-diagrams* to cover the static aspects and *state diagrams* description to represent the dynamic part of the specification [3].

3.1 Class and Object-diagrams

The right-hand side of the Figure above depicts the possible classes. The class COMPONENT, for instance, acts as a super-class for all active components (HTS, machines, buffers). HTS have to control the flow of work pieces in the scenario. The class JOB is an abstraction of a task a HTS must perform. Machines have local entry (SOURCE) and exit-buffers (DESTINATION). A BUFFER models an abstract super-class to insert, store, and release a (limited) number of work-pieces. Elements of class OBJECT model the work pieces. They are identified using the attribute oid.

3.2 State Diagrams

With the *behaviour* state-diagram, different states of the objects in the scenario with state transition rules depicting the necessary preconditions and events are described.
The Left-side of Figure 1 depicts the state diagram of the class HTS. By occurrence of the birth event start the HTS changes its state into ready. Receiving a job request from a machine in this state (receive_job) changes the HTS into the state received_J. In case a partner is required to perform the requested job, the HTS has to determine and contact all possible partners for the job (request_Partner). Otherwise (e. g. if the HTS already carries a requested work piece) it may directly calculate the offer by the event calc_Offer and make thereby a transition into the state calculated. In the contacted state, the HTS will have to wait until either all contacted partners have answered the request or a predefined time-limit (rlimit) has elapsed. The remaining HTS at this time may send the approval for the job (send_Approval).

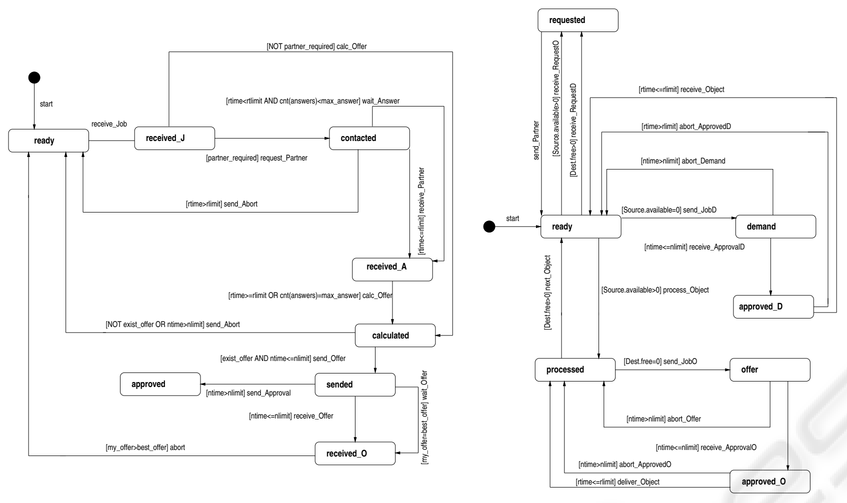

[NOT partner_required] calc_Offer
start
receive_Job
ready
received_J
[rtime<rtlimit AND cnt(answers)<max_answer] wait_Answer
contacted
[partner_required] request_Partner
[rtime>rlimit] send_Abort
[rtime<=rlimit] receive_Partner
received_A
[rtime>=rlimit OR cnt(answers)=max_answer] calc_Offer
[NOT exist_offer OR ntime>nlimit] send_Abort
calculated
[exist_offer AND ntime<=nlimit] send_Offer
approved
sended
[ntime>nlimit] send_Approval
[my_offer=best_offer] wait_Offer
[ntime<=nlimit] receive_Offer
[my_offer>best_offer] abort
received_O
requested
send_Partner
[Source.available>0] receive_RequestO
[Dest.free>0] receive_RequestD
[rtime<=rlimit] receive_Object
[rtime>rlimit] abort_ApprovedD
[ntime>nlimit] abort_Demand
start
ready
[Source.available=0] send_JobD
demand
[ntime<=nlimit] receive_ApprovalD
[Dest.free>0] next_Object
[Source.available>0] process_Object
approved_D
processed
[Dest.free=0] send_JobO
offer
[ntime>nlimit] abort_Offer
[ntime<=nlimit] receive_ApprovalO
[ntime>nlimit] abort_ApprovedO
[rtime<=rlimit] deliver_Object
approved_O

Fig. 1. State Diagram of HTS and Machine Classes.

3.3 OCL- pre- Post-Constraints

We further judge that relevant to the modelling phase belongs also the dynamic detail about different methods, namely their enabling pre-conditions and resulting post-conditions. With the modelling of such methods features at the modelling level, we result in more controlled implementation outputs, where the programmers have to be bounded by a minimal conceptual constraints in developing the body of each methods. Pre- and Post-constraints are further crucial for our subsequent phase, where they have to govern different transition input/output arc inscriptions and thereby the to-be-conceived nets.

Method Name	**Pre-Constraint**	**Post-Constraint**
receive_Job(j:JOB)	send_AbortA **or** send_AbortB	Jobs:=Jobs.**insert**(j), calc_OfferA, request_Partner
receive_Partner(m:MACHINE,costs:**nat**)	rtime≤rlimit , request_Partner	answers:=answers+, **mk-set**(**mk-tuple**(m,costs)), wait_Answer, calc_Offer
receive_Offer(h:HTS,costs:**nat**)	ntime≤nlimit	offers:=offers+**mk-set** (**mk-tuple**(h,costs)), wait_Offer, abort
request_Partner(j:JOB)	partner_required=**true**	m.receive_RequestD(**self**,j), m.receive_RequestO(**self**,j)
receive_Job(j:JOB)	send_AbortA **or** send_AbortB	Jobs:=Jobs.**insert**(j), calc_OfferA, request_Partner

With respect to the running case study, we depict in the following some of these pre- and post-conditions to be associated with different methods related different classes in the above discussed class-diagram. For instance, the `receive_job` method should preceded by the execution of the two methods `send_AbortA` and `send_AbortB`. The resulting output consists in incrementing the job list by the requested to-be-performed job and by the triggering of the two following methods `calc_OfferA` and `request_Partner`.

After achieving this analysis / modelling phase, we have now to leverage it to more cohesive view, where distribution, componentization, visual and symbolic validation are intrinsic. The corresponding specification/validation framework we are putting forwards is CO-NETS, a tailored form of integration of OO structuring mechanisms with high-level algebraic Petri nets that we soundly interpret in rewriting logic.

4 The CO-NETS Approach: An Overview

For the purpose of this paper and also due to space limitation, only some CO-Nets[1] aspects are reviewed in what follows. Reader is referred to [4] for more detail.

4.1 Component Structural Specification

States in CO-NETS are terms of the form $\langle Id | at_1 : v_1, ..., at_k : v_k, bs_1 : v'_1, ..., bs_{k'} : v'_s \rangle$; where Id is an object identity ; $at_1, .., at_k$ are local and $bs_1, ..., bs_s$ are observed from other components.We further allows 'splitting / recombining' this state at request. Similarly, we explicitly distinguish between internal, local messages and the external as imported/exported messages. Local messages allow for evolving the object states of a given class, while the external ones allow for communicating between different classes using exclusively their observed attributes.

4.2 Component Behavioral Specification

On the basis of this component signature, we define the notion of CO-NETS specification incrementally as follows. CO-NETS places are precisely defined by associating with each message generator (type) one place we call 'message' place. We also associate with each component sort one place that has to contain the current states within such component. CO-NETS transitions reflect the effect of messages on the targeted states. We further make distinction between local transitions that reflect state changes and the external ones modeling the interaction between different components.

4.3 CO-NETS: Semantical Aspects

As general behavioral pattern for transitions, we propose that the effect of CO-NETS transitions capture the following interaction-driven computation. The contact of state parts in a given componentn Cl, —namely $\langle I_1 | ats_1 \rangle$;..; $\langle I_k | ats_k \rangle$— with some messages $ms_{i1}, .., ms_{ip}, ms_{j1}, .., ms_{jq}$—declared as *local or imported* in this component— and under some conditions on the invoked attributes and message parameters results in the following effects: (1) Messages such as $ms_{i1}, .., ms_{ip}, ms_{j1}, .., ms_{iq}$ vanish; (2) State changes of some (parts of) states participating in this computation, namely $I_{s1}, .., I_{st}$; (3) Deletion of some states by sending explicitly delete messages; and new messages are sent to the component Cl state, namely $ms'_{h1}, .., ms'_{hr}, ms'_{j1}, .., ms'_{jq}$.

[1] An acronym for **C**oncurrent **O**bject oriented Petri **Net**.

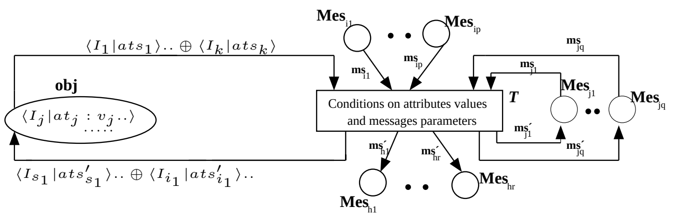

Mes$_{i1}$
Mes$_{ip}$
$\langle I_1|ats_1\rangle..\oplus\langle I_k|ats_k\rangle$
ms$_{i1}$
ms$_{ip}$
ms$_{jq}$
ms$_{j1}$
obj
$\langle I_j|at_j : v_j..\rangle$
Conditions on attributes values and messages parameters
T
Mes$_{j1}$
Mes$_{jq}$
ms´$_{j1}$
ms´$_{h1}$
ms´$_{hr}$
ms´$_{jq}$
$\langle I_{s_1}|ats'_{s_1}\rangle..\oplus\langle I_{i_1}|ats'_{i_1}\rangle..$
Mes$_{h1}$
Mes$_{hr}$

Fig. 2. The Intra-Component Computation Pattern.

Rewriting rules governing the CO-NETS behaviour Each CO-NETS transition is captured by an appropriate rewriting rule interpreted into rewrite logic[5]. Following the intra-component evolution pattern in figure 2, the general form of rewrite rules that we associate with it takes the form below. The operator $\oplus$ is defined as a multiset union and allows relating different place names with their current markings. Whereas the multiset operator $\otimes$ allows composing different couples place-inscription, so that we can define different input arcs and output arcs in a given transition. Moreover, we assume that $\otimes$ is distributive over $\oplus$ i.e. $(p, mt_1 \oplus mt_2) = (p, mt_1) \otimes (p, mt_2)$ with mt_1, mt_2 multiset of terms over $\oplus$ and p a place identifier.
$T : (Ms_{i_1}, ms_{i_1}) \otimes ..(Ms_{j_q}, ms_{j_q})\otimes\ (obj, \langle I_1|attrs_1\rangle \oplus .. \oplus \langle I_k|attrs_k\rangle)$
$\Rightarrow (Ms_{h_1}, ms'_{h_1}) \otimes ...(Ms'_{j_q}, ms'_{j_q}) \otimes (obj, \langle I_{s_1}|ats'_{s_1}\rangle.. \oplus \langle I_{i_r}|attrs'_{i_r}\rangle)$
$if\ Conditions$ and $M(Ad_{Cl}) = \emptyset$ and $M(Dl_{Cl}) = \emptyset$

We point out that more advanced component-based abstraction mechanisms have been conceived for capturing inheritance, aggregation and interaction between components [4].

5 From UML-driven IS to CO-NETS Components

After sketching the main concepts of the CO-NETS framework, we discuss in this main section how to incrementally and (semi-)automatically derive a coherent view of different UML diagrams and OCL constraints based on CO-NETS components. First, we present how structural features from class classes can be mapped to corresponding CO-NETS component templates. Further, we address the translation of different behavioral and dynamic diagrams and OCL constraints in the behavioral sides of the CO-NETS framework.

5.1 From UML Classes to CO-NETS Components Structure

This translation mainly concerns attribute and message descriptions, and it can be made precise through the following translating steps.

1. Different attributes associated with a given class-diagram are directly specified as component states–as tuple terms— by gathering them together and enriching them with the state identity part. Possibilities in restricting, initializing or fixing some attributes values have to be expressed as conditions in the creation/deletion component transitions.

2. In order to have just one and a uniform view, besides these stateless attributes that are *explicitly* defined in the class-diagrams, explicitly defined states and their changes from the state-diagram have also to be added to the tuple as *stateful extra-attributes*.
3. All event / message generators will be regarded as messages by enriching their arguments with identities of involved states. Moreover, by taking profit of the communication diagram, all events that are to be sent to other classes have to be defined as *exported* ones.
4. From the explicit effect of these external messages described in OCL- pre- and post-conditions we straightforwardly conceive the *observed* attributes part as those which are explicitly involved in such state changes.

The corresponding CO-NETS component signature for the HTS, for instance, is forwarded in what follows. First we have to specify the data-types that are used in this template signature for specifying attribute values and/or event parameters, such as natural, lists and so on (we are omitting here). The HTS component structure takes then the form:

```
obj HTS-signature is
  protecting Object-state HTS-DATA .
  subsort Id.HTS < OId .
  subsort Local_HTS  External_HTS < HTS .
  subsort SND_OFFR  RCV_OFFR  SND_ABR
          WAIT_ANSW  WAIT_OFFR
          CALC_OFFR  ABORT  APPROVAL < Local_HTS_Mes .

  subsort REQUEST_PRT < Exported_HTS_Mes .
  (* local attributes *)
   op ⟨_|answ : _, offr : _, rlimt : _, nlimt : _, nlimt : _, my_offr : _, exist_offr : _⟩ :
   Id.HTS List_answ List_offr nat nat nat nat bool → Local_HTS.
  (* observed attributes *)
   op ⟨_|StH : _, rtim : _, ntim : _, partn : _, job : _, mx_prt : _, Jobs : _⟩ :
      Id.HTS  StateH  nat  nat  Id.job Bool  nat  List_job → Ext_HTS .
  (* local messages *)
   op Snd_offr, Snd_abort, Wait_answ : Id.HTS → Local_HTS_Mes .
   op Rcv_offr, Wait_offr, Cal_offr: Id.HTS Real → Local_HTS_Mes .
   op Abort, Approval : Id.HTS  Job → Local_HTS_Mes .
  (* export messages *)
   op Requst_P(artner): Id.Job  Id:HTS  Id.Mach → Exp_HTS_Mes .
  (* Imported messages *)
   op Rcv_Job, Rcv_P : OId  Id.HTS Id.Job → Import_HTS_Mes
  vars H : Id.HTS ; J : Job ; S : List_answ.
  vars C : Real (cost) ; J : Job ; R, M, L : Nat; Q : Boolean.
endo.
```

5.2 From UML-behaviral Diagrams to CO-NETS

Following the CO-NETS approach, in addition to state places that have to contain the different current state instances, with each event (now a message) generator a corresponding place is associated. The behaviour of each local event is captured by an appropriate transition, where:

1. The place associated with this event is taken as input place, while the event itself labels the corresponding input arcs;

2. The OCL pre-condition is expressed either as conditions in this transition or as appropriate instantiations in the label of the input arc from the state place. If other messages are required in the pre-conditions, input
3. The OCL post-condition part clause is modeled as an appropriate inscription terms of an output arc that goes to the object place.
4. Finally the *calling* clause is captured by associating output arcs labelled by the corresponding called messages and destinated to their associated (message) places.

In the same spirit, for communicating different templates, external event behaviour is captured by transitions that make into relation only external attributes part and imported / exported events. We survey all these translating steps in the table below.

UML 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	conditions in creation/deletion net
state change in SM	additional attributes called State
events	messages with explicit identity of the invoked object
— OCL pre-	Transition condition
— OCL post-	Transition (output) effect

Example 1. By applying the above translating ideas to our running example, we result in two CO-NETS components reflecting respectively the behaviour associated with HTS and Machine CO-NETS template signatures (i.e. the HTS and the Machine). Moreover and because the interaction between these two components have to be achieved only through their explicit interfaces a further 'communicating' CO-NET is to be conceived for capturing their interaction.

For instance, by respecting the aforementioned translating steps, the CO-NET associated with the HTS template is depicted in figure 3. In this net, besides the state place **OBJ_HTS** that contains all HTS state instances, with each message generator a corresponding place is associated. Places reflecting imported / exported messages are represented in bold. For the Job component declared as a subcomponent (in the HTS TROLL text), we have just represented the places of imported messages namely **RCV_ApvD, RCV_ApvO, ADD_JOB** and the exported attributes **Type** of job (i.e. OFFER or DEMAND). Each transition reflect the effect of messages on the *just* concerned state parts. The transition **RCV_JOB**, where the message **Receive_J(J,M,H)** (with **J, M, H** denotes respectively job identity, machine identity and HTS identity) enters into with state part $\langle H|StateH : Ready, Job : Jb, Partner\_Rq : R\rangle$. This means that according to the HTS state diagram, the HTS state of **H** should be **Ready** and their is some (list of) jobs **Jb**. The effect of such contact depending on whether a partner is requested or not (i.e. the variable **R** is true or not) is as follows. In the first case (i.e. **Partner_Rq = True**) the invoked part of HTS object state is modified to $\langle H|StateH : Calculated, Job : Jb : J, Partner\_Rq : R\rangle$ with a simultaneous sending of the messages **Request_P(J,H,M)**, **Add_Job** (to the job subcomponent) and **Calc_Offer(J,H)**. In the second case, the HTS object part of state is modified to $\langle H|StateH : Contact\_J, Job : Jb : J, Partner\_Rq : R\rangle$ with a simultaneous sending of the messages **Add_Job** and **Calc_Offer(J,H)**.

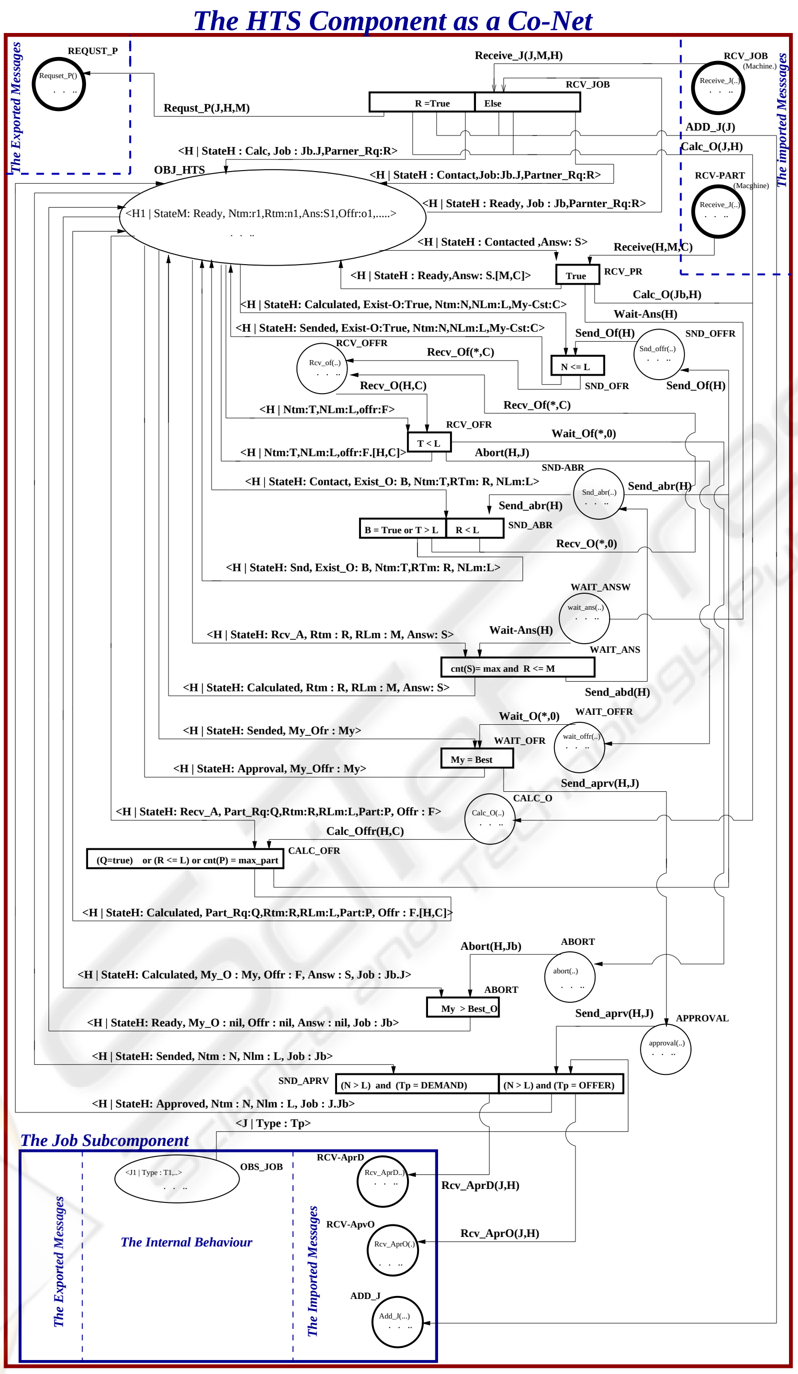

The HTS Component as a Co-Net
The Exported Messages
REQUST_P
Requset_P(..)
Requst_P(J,H,M)
Receive_J(J,M,H)
RCV_JOB
R =True
Else
RCV_JOB
(Machine.)
Receive_J(..)
The imported Messages
ADD_J(J)
Calc_O(J,H)
<H | StateH : Calc, Job : Jb.J,Parner_Rq:R>
OBJ_HTS
<H | StateH : Contact,Job:Jb.J,Partner_Rq:R>
RCV-PART
(Macghine)
Receive_J(..)
<H | StateH : Ready, Job : Jb,Parnter_Rq:R>
<H1 | StateM: Ready, Ntm:r1,Rtm:n1,Ans:S1,Offr:o1,.....>
<H | StateH : Contacted ,Answ: S>
Receive(H,M,C)
<H | StateH : Ready,Answ: S.[M,C]>
True
RCV_PR
Calc_O(Jb,H)
<H | StateH: Calculated, Exist-O:True, Ntm:N,NLm:L,My-Cst:C>
Wait-Ans(H)
<H | StateH: Sended, Exist-O:True, Ntm:N,NLm:L,My-Cst:C>
Send_Of(H)
SND_OFFR
RCV_OFFR
Rcv_of(..)
Recv_Of(*,C)
Snd_offr(..)
N <= L
SND_OFR
Send_Of(H)
Recv_O(H,C)
Recv_Of(*,C)
<H | Ntm:T,NLm:L,offr:F>
RCV_OFR
T < L
Wait_Of(*,0)
<H | Ntm:T,NLm:L,offr:F.[H,C]>
Abort(H,J)
SND-ABR
<H | StateH: Contact, Exist_O: B, Ntm:T,RTm: R, NLm:L>
Snd_abr(..)
Send_abr(H)
Send_abr(H)
B = True or T > L
R < L
SND_ABR
Recv_O(*,0)
<H | StateH: Snd, Exist_O: B, Ntm:T,RTm: R, NLm:L>
WAIT_ANSW
wait_ans(..)
<H | StateH: Rcv_A, Rtm : R, RLm : M, Answ: S>
Wait-Ans(H)
WAIT_ANS
cnt(S)= max and R <= M
<H | StateH: Calculated, Rtm : R, RLm : M, Answ: S>
Send_abd(H)
Wait_O(*,0)
WAIT_OFFR
<H | StateH: Sended, My_Ofr : My>
WAIT_OFR
wait_offr(..)
My = Best
<H | StateH: Approval, My_Offr : My>
Send_aprv(H,J)
CALC_O
<H | StateH: Recv_A, Part_Rq:Q,Rtm:R,RLm:L,Part:P, Offr : F>
Calc_O(..)
Calc_Offr(H,C)
CALC_OFR
(Q=true) or (R <= L) or cnt(P) = max_part
<H | StateH: Calculated, Part_Rq:Q,Rtm:R,RLm:L,Part:P, Offr : F.[H,C]>
Abort(H,Jb)
ABORT
<H | StateH: Calculated, My_O : My, Offr : F, Answ : S, Job : Jb.J>
abort(..)
ABORT
My > Best_O
Send_aprv(H,J)
APPROVAL
<H | StateH: Ready, My_O : nil, Offr : nil, Answ : nil, Job : Jb>
approval(..)
<H | StateH: Sended, Ntm : N, Nlm : L, Job : Jb>
SND_APRV
(N > L) and (Tp = DEMAND)
(N > L) and (Tp = OFFER)
<H | StateH: Approved, Ntm : N, Nlm : L, Job : J.Jb>
<J | Type : Tp>
The Job Subcomponent
<J1 | Type : T1,..>
OBS_JOB
RCV-AprD
Rcv_AprD..)
Rcv_AprD(J,H)
The Exported Messages
The Internal Behaviour
The Imported Messages
RCV-ApvO
Rcv_AprO(..)
Rcv_AprO(J,H)
ADD_J
Add_J(...)

Fig. 3. The HTS CO-NETS Specification.

6 Co-nets Specification Certification

The Co-nets approach with its rewriting logic based operational semantics allows not only for graphical visual animation of the intra-component computation and the inter-component cooperation but also for deriving symbolic proofs using the associated rewrite theory [4]. While validating the modelled component-based information systems, a strict explicit separation of concerns is adopted where:

- The behavioral certification of a given component (as a class in the simple case) by concurrent rewriting and simultaneous graphical animation from an initial component state is *completely independent* from any other component. The efficiency intervenes here by the limited number of manipulated transition rewriting rules compared to the case where the whole system (i.e. when following a usual UML monolithic process where all classes at stake at once) is analyzed.
- The certification of the communication and the effect of inter-component interactions on the whole system is also achieved independently, by taking into account only the interface of the concerned components.

Example 2. By applying the general forms of rewrite rules, it is not difficult to generate the rules governing the behaviour of both **HTS** and **Machine** components as well as those of their interactions. Hereafter, due to space limitation we just illustrate it through two rewrite rules of the HTS component.

RECV_JOB[2]: $(RCV\_JOB, Receive\_J(J, M, H)) \otimes (OBJ\_HTS, \langle H|StateH : Ready, Job : Jb, Partner\_Rq : R\rangle) \Rightarrow\ if\ (R = True)\ then\ (OBJ\_HTS, \langle H|StateH : Calculated, Job : Jb, Partner\_Rq : R\rangle) \otimes (REQUEST\_P, Request\_P(J, H, M)) \otimes (ADD\_JOB, Add\_job(J)) \otimes (CALC\_OFFR, Calc\_Of(J, H))\ else\ (OBJ\_HTS, \langle H|StateH : Contacted, Job : Jb, Partner\_Rq : R\rangle) \otimes (ADD\_JOB, Add\_job(J)) \otimes (CALC\_OFFR, Calc\_Of(J, H))$

RCV_PAR: $(RCV\_PART, Receive\_P(J, M, C)) \otimes (OBJ\_HTS, \langle H|StateH : Contact, Answ : S\rangle) \Rightarrow (OBJ\_HTS, \langle H|StateH : Ready, Answ : S.[M, C]\rangle) \otimes (WAIT\_ANS, Wait\_answ(H)) \otimes (CALC\_OFFR, Calc\_O(Jb, H))$

7 Conclusions

In this paper, we presented a stepwise proposal for semi-formally modelling and formally specifying and validating advanced distributed information systems. The proposal starts with widely accepted and practitioner-oriented UML methodology. More precisely, we model the information through its class- communication and statecharts diagrams plus OCL-pre- and post-conditions. For sake of global coherent view of these different diagrams, we propose to incrementally derive Co-nets components, where intra- computation is explicitly separated from inter-interactions and where rapid- prototyping are possible with graphical animation and rewriting computations. In order to emphasize the practicability and the capabilities for developing even complex information systems, we have applied the proposed methodology to a significant part of a realistic case study dealing the production cell problem.

Nevertheless, after achieving this very important and first steps towards developing advanced information systems, we are conscious that much a work remain ahead. First, we plan a deeper study for an appropriate and efficient translation of the generated rewriting rules into JAVA programs. Second, we are working for developing a complete software environment for the CO-NETS framework, including particularly an editor / simulator and relate it with current UML environment such as Rational and Posedion. For the rewriting sides we are adapting the current implementation of the MAUDE language. Last but not least, for coping with the runtime evolution due to frequent changes we are working on an appropriate reflective extension of the CO-NETS approach.

Acknowledgements

The authors want to warmly thank the three referees for their constructive comments, that significantly helped improving the present final version.

References

1. Jungclaus, R., Wieringa, R.J., Hartel, P., Saake, G., Hartmann, T.: Combining TROLL with the Object Modeling Technique. In Wolfinger, B., ed.: Innovationen bei Rechen- und Kommunikationssystemen. GI-Fachgespräch FG 1: Integration von semi-formalen und formalen Methoden für die Spezifikation von Software. Informatik aktuell, Springer (1994) 35–42
2. Wirsing, M., Knapp, A.: A Formal Approach to Object-Oriented Software Engineering. ENTCS **4** (1996)
3. Booch, G., Jacobson, I., Rumbaugh, J., eds.: Unified Modeling Language, Notation Guide, Version 1.0. Addison-Wesley (1998)
4. Aoumeur, N., Saake, G.: A Component-Based Petri Net Model for Specifying and Validating Cooperative Information Systems. Data and Knowledge Engineering **42** (2002) 143–187
5. Meseguer, J.: Solving the Inheritance Anomaly in Concurrent Object-Oriented Programming. In: ECOOP'93 - Object-Oriented Programming. Volume 707 of Lecture Notes in Computer Science., Springer (1993) 220–246