# Towards a Three-Level Methodology for Developing Cooperative Information Systems

Nasreddine Aoumeur[1] Soïen Balko Gunter Saake

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

March 2000

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

**Abstract**

The objective of this paper is to present our first results towards a three-level methodology for developing advanced information systems as fully distributed, autonomous yet cooperative components. The proposed methodology aims to deal with the main phases in developing such systems, namely analysis / specification phase, validation phase and implementation phase. For the analysis / specification level we follow the OMTROLL approach, that is, first different system classes are graphically modeled using the OMT object model with communication and state diagrams, and then structural and behavioural aspects of such components are specified using the TROLL language. At the validation level we propose an object oriented Petri net-based model named CO-NETS that we are developing. The two levels are intrinsically connected through a semi-automatic generation of CO-NET components from different OMTROLL descriptions. CO-NETS validation is achieved mainly through graphical animation accompanied by a concurrent computation based on its semantics expressed in rewriting logic with a full exhibition of intra- as well as inter-object concurrency. The proposed approach is illustrated using a significant and a realistic part of a production system case study.

# Contents

# Chapter 1

# Introduction

Due to multi-dimensional, non-standard requirements to cope with during their development, information systems are one of the vivid and complex application domains where different formal and semi-formal frameworks have to be put together. In fact, most of present-day information systems are regarded as consisting of fully distributed and cooperative components with large databases and application programs [PS98]. Henceforth, among the challenging characteristics they present there are particularly: very complex and multi-layered state-dependent data, complex behavioral aspects, intrinsic (geographical) distribution involving different forms of communication, runtime evolution, etc.

The great efforts undertaken in more than two decades agreed widely that the *object oriented* (OO) setting is one of the best and most common paradigm around which different formal and/or semi-formal frameworks can be easily and soundly integrated to overcome the aforementioned requirements. Confirmation of this claim is the recent success of object oriented approaches, dedicated particularly for the analysis/specification phases, integrating informal OO diagrammatic notations and formal OO specifications. Such approaches include especially OMTROLL [WJH+93, JWH+94] that integrates intuitive diagrammatic notations of the OMT object model [RBP+91] with the (highly) declarative OO specification TROLL [JSHS96] language. Similar integration are: formal OOSE [WK96] combining Jacobson's method OOSE with OO algebraic specification based on an adequate adaptation of Meseguer's rewriting logic; Z++ and VDM++ [Lan95].

The contribution of this paper fits in this vivid research direction, and it proposes to extend the analysis/specification phase of the OMTROLL approach for dealing appropriately with the further validation / implementation phases. However, for this aim we refrain from building 'ad-hoc' implementation-dependent animators as it has been hitherto done [HG94]. Such animators, although they give some insight on how the specified system works, they require very strong restrictions like absence of any forms of distribution, limitation of class number and so on.

As an appropriate alternative, we propose rather a validation based on a sound independent-machine framework allowing for shifting in a very natural and implicit way from the conceived as *object community* OMTROLL specification towards a *fully distributed, autonomous yet cooperative components* specification which is moreover ready for validation purposes. Thus, instead of restricting the OMTROLL specification, we intrinsically extend it for fulfilling more advanced requirements including:

- intra- as well as inter-object concurrency and synchronous as well as asynchronous communication,
- inter-component interaction without violating the encapsulated part of each component regarded as hierarchy of classes,

- graphical animation accompanied by formal concurrent reasoning.

The proposed framework for this advanced specification / validation phase is a new form of object oriented Petri nets model interpreted in rewriting logic [Mes92, Mes98]. Referred to as CO-NETS, this specification / validation model that we are developing [AS99a, AS99b] naturally extends the OMTROLL approach through the following key features:

- Like in OMTROLL, we propose a clean separation between *data* —specified algebraically and used for describing object attributes (values and identifiers) and message parameters— and *objects* as indivisible and uniquely identified units of structure and behaviour. Moreover, we propose to use ordered sorted algebras (OSA) [GD94] specification for the data level following an OBJ notations [GWM+92] particularly.

- For the object level, object states are conceived as tuples which may be *split* (resp. *re-combined*) at any time through a very flexible axiom. Different states of a given class are gathered into an appropriate (object) place, while for each message (also called event or method-invocation) generator a (message) place is conceived. The method body is reflected using appropriate transition respecting a very general evolution pattern that exhibits a full intra- as well as inter-object concurrency and synchronous as well as asynchronous communication.

- The modeling of simple and multiple inheritance, with associated polymorphism and dynamics binding with possibility of methods overriding, is achieved in a straightforward way; particularly by structurally using subsorts and behaviourally taking profit of the proposed 'splitting and recombination' axiom of the object state at a need. It also worth mentioning that besides being very simple, our modeling of different forms of inheritance coexist in harmony with the (intra- and inter-object) concurrency and hence does not suffer from the so-called inheritance anomaly.

- For promoting communication and autonomy, in the CO-NETS we distinguish in each component (i.e. hierachy of classes) between local structural and behavioural aspects and external ones. Structurally, we propose to distinguish between internal part of object state (i.e. attributes) which cannot be observed by other components and observed part which may be exchanged with other components. Parallely, we also distinguish between internal messages and external ones. Behaviourally, we propose for interacting different components through their explicit interfaces a general interaction pattern to be followed by transitions.

- The CO-Nets semantics is interpreted in rewriting logic that is a *true-concurrent* operational semantics allowing particularly rapid-prototyping using concurrent rewriting techniques in general and current implementation of the MAUDE language [Mes98] specifically.

From a methodological point of view, we argue that our three-level proposal methodology for developing information system is incremental. Indeed, each phase focus is to complete and refine in a natural way its preceding by fulfilling more requirements. More precisely, for developing reliable and fully distributed information, the proposed methodology aims to respect the following methodological guidelines and steps.

1. **The analysis / specification Level**

**The OMT analysis phase:** The success of the OMT object model as one of the easiest ways for eliciting user requirements and representing them as a hierarchy of classes is widely accepted[1].

**Communication and State Diagrams:** Experiences show that according to given application, it's very natural to extract different states through which a given entity or object may pass through. The same fact hold for determining which events or actions are communicating with each other.

TROLL **specification:** After achieving the two first steps where general views of the system are considered, the TROLL specification allows particularly for defining precisely the different attributes, events, conditions of events to act on attributes as well as the result of such effect.

2. **The advanced specification /validation Level**

CO-NETS **specification:** The main objective of this specification is to combine *together* the four above views. More precisely, while from the communication we extract which structural as well as behavioural part in a given class should local (resp. observed or external), from the state diagram with the corresponding TROLL specification we generate a uniform a view of the change in the system. The result of this phase is the generation of components that behave autonomously and interact with each other without violating the encapsulated part of each component.

CO-NETS **animation accompagnied by symbolic computation:** This step is crucial for detecting some design flaws and missing due to possible misunderstanding between application experts and designers. Moreover, due to our conception of the whole system as cooperating components, rapid-prototypes can be efficiently generated by animating each component separately.

3. **The implementation Level**

CO-NETS **Java implementation:** Although due to space limitation this phase will not be addressed in this paper we are planing to generate Java code from the rewriting rule governing the CO-NETS behaviour. The proposed translation is very close, but with significant adaptations, to the one proposed in [WK96].

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 the proposed methodology. The third section presents the object, communication and state diagrams as well as the TROLL specification associated with our running case study. The third section review the main features of the CO-NETS approach which are addressed in this paper. In the main section, we present our translating ideas for shifting from these four views into the corresponding CO-NETS specification. The fourth section deals with the validation phase. We finally close this paper with some remarks and future extensions.

[1]However, it is worth mentioning that human as well as organizational aspects [AF95] are out of the scope of the present work that focuses more on the functional requirements.

# Chapter 2

# The Holonic Transport case study

This scenario is used as a case study in a project[1] dealing with integration issues of specification techniques w.r.t. engineering applications.

As depicted in the left side of Figure 2.1, in its basic level, this case study deals with a part of a production system where specific work pieces (OBJECT) are processed by three different machines ($M_x$) in a specific order. Since there is no physical connection among these machines, 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. Since an important boundary condition is that there exists no central control for the transport, these HTS have to be self-organizing in a way that they locally decide by competing offers which transport job they execute.

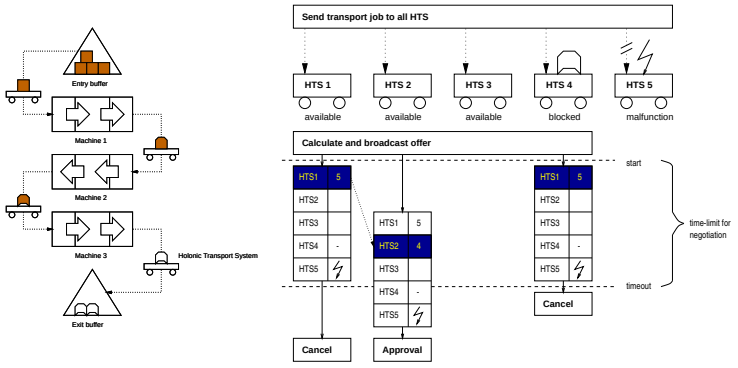


Figure 2.1: Scenario of the Holonic Transport and Job Negociation

Additionally, there are two buffers in that scenario. The first one (IN) provides "fresh" (entirely unprocessed) work pieces, wheres completely processed work pieces are delivered into the second one (OUT). In this basic case we will restrict the case study onto these physical components.

Every machine consists of local entry and exit buffers which may store unprocessed/processed work pieces. Eeach 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).

Due to space restrictions we can not formally specify every mentioned object (like central clock, local buffers etc.) but will give attributes and events of these objects with semantically

[1] which is referred to as "Integration of Software Specifications for Engineering Applications" and is sponsored by the Deutsche Forschungsgemeinschaft (DFG).

meaningful names instead. To ensure a termination of the job negotiation, we use *timeouts* to establish a time-limit for such processes. Any HTS participating in the negotiation has to determine its *cost-value* within this limit. Delayed HTS will not take part in the negotiation process. Right-side of Figure 2.1 depicts an informal representation of the job negotiation:

1. A machine $M_x$ generates a request and sends it together with a time-stamp via broadcast to all HTS.
2. All non faulty HTS can receive this request. Malfunctional HTS are known to the other HTS and do not take part in the negotiation process.
3. 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.
   - Otherwise the negotiation process is cancelled. Since the other HTS will also have their negotiation finished at this time, it is also not necessary to notify them.
4. 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.
5. Otherwise the offer is calculated, sent to the other HTS and entered into an internal cost comparison table (CCT).
6. Until the time-limit is reached, all HTS collect the offers of the other HTS and enter them into their CCT.
   - If the own value is below the offer of any other HTS, the first one aborts its negotiation process.
   - Otherwise, this cycle is continued until the time-limit is reached.
7. If there is one[2] HTS remaining after the time-limit, this HTS may send the approval for the request and therefore gets the job.

[2] Here we assume, that there can not be two HTS sending the same offer.

# Chapter 3

# OMTROLL-Specification

Based on the informal description in Section 2, we will now generate appropriate OMTROLL diagrams. An OMTROLL specification consists of *community diagrams* and *class declaration diagrams* to cover the static aspects and a *communication diagram* as well as *state diagrams* description to represent the dynamic part of the specification [GKFK+98].

## 3.1 Object Community Diagram

Beginnig with the strucural aspects of our specification we now present an object community diagram.

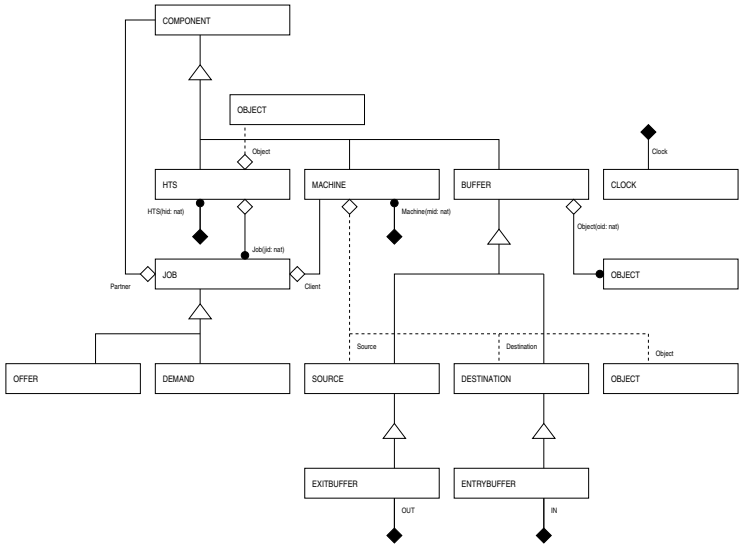


Figure 3.1: Object Community Diagram

In Figure 3.1 we present the structure of the objects in our example. Be aware that we omit any attributes or events notations in this diagram.

The class COMPONENT acts as a super-class for all active components (HTS, machines, buffers). Thus, it provides low-level broadcast communication facilities. 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. It consists of a client (the caller of the job) and (optionally) a partner (a second component needed to perform the job[1]). Depending on the clients status we distinguish between

[1] e.g. a exit-buffer as source for a work piece in a demand-job

OFFER and DEMAND. Machines have local extry (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 Class Declaration and Communication Diagrams

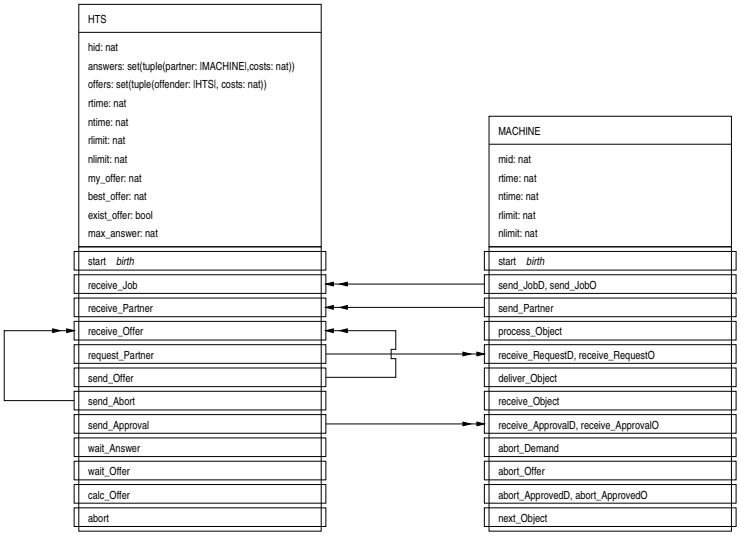


Figure 3.2: Communication Diagram

Figure 3.2 depicts the complete class declaration of the classes HTS and MACHINE with (a part of) their communication relationships.

**HTS** The HTS are identified using a unique natural number (hid). They contain timer-variables for the partner-request and job-negotiation process (rtime and ntime) and appendant time-limit-constants (rlimit and nlimit). Possible answers on a partner-request are maintained in the set answers whereas incoming offers of other HTS are stored in the offers set. In addition, there are a number of *derived* attributes. The attributes my_offer and best_offer extract the HTS own and the best offer from the offers set. The boolean attribute exist_offer marks if a offer could be calculated. The boolean attribute partner_required shows whether a partner is required to perform a certain job. Finally there is the natural constant max_offers which represents the maximum number of expected replies for a partner request.

**MACHINE** Likewise as HTS all machines are identified using a unique number called mid. Additionally, the attributes rtime, ntime, rlimit, and nlimit are also present in class MACHINE to provide endless waiting states.

In addition to the attributes and events, Figure 3.2 also depicts inter-object event calling by means of arrows. Thus, some events of class HTS cause an event in class MACHINE to occur and vice-versa. For example, the event HTS.request_Partner calls one of the events MACHINE.receive_RequestD or MACHINE.receive_RequestD depending on whether the requested partner is required to deliver or to take over a work piece. In our scenario, HTS have to communicate among each other as well.

## 3.3 State Diagrams

Following the structure and community diagrams, we now present the behavioural aspects. Using *behaviour diagrams*, different states of the objects in the scenario with state transition rules depicting the necessary preconditions and events are described. States are represented as circles or rounded rectangles, while state transitions are shown as arrows labeled with events connecting two states. Preconditions are optional and displayed as angular brackets in front of an event. Left-side of Figure 3.3 depicts the state diagram of the class HTS. By occurrence of the birth

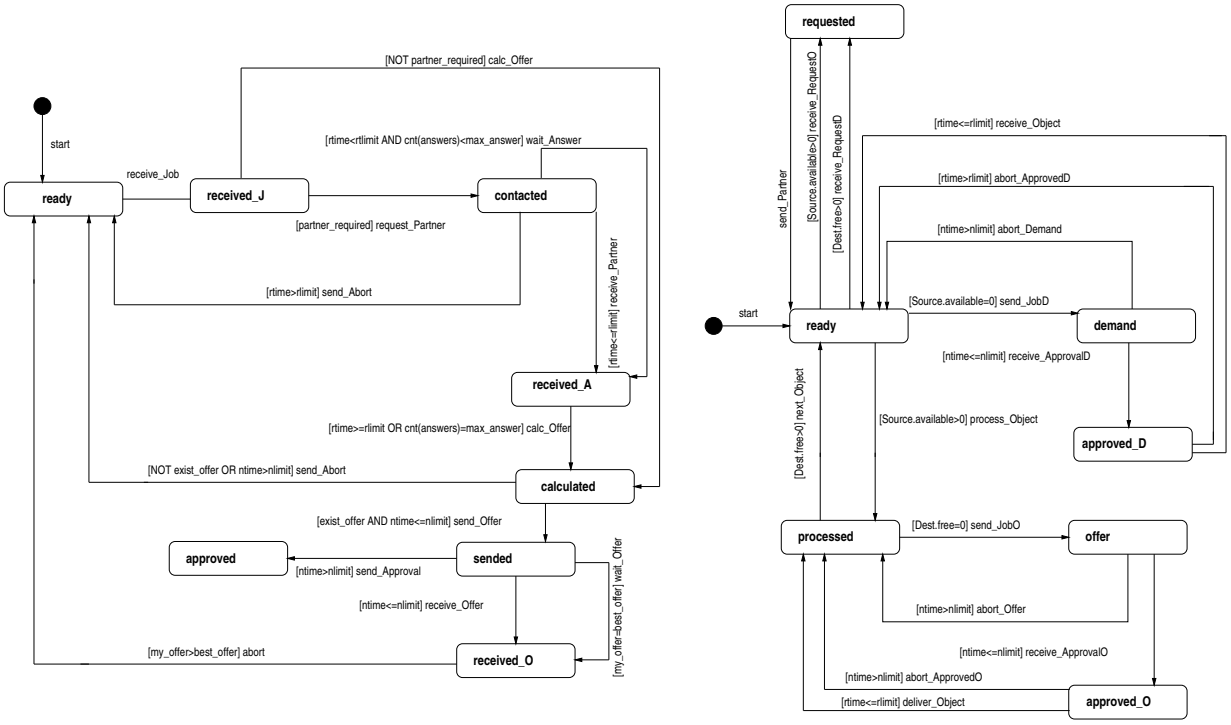


Figure 3.3: State Diagram of HTS and Machine Classes

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. If at least one answer has arrived a offer can be calculated.

Following the calculated state the current time (ntime) is compared with the negotiation time-limit (nlimit). Additionally, the calculated offer is checked on its existence. If both requirements are fulfilled, the offer can be sent to all other HTS (send_Offer) otherwise the HTS has to abort (send_Abort) the negotiation and notify the other HTS. When the offer was successfully sent, the HTS waits for competing offers until either a better offer arrives or the negotiation time has elapsed. The remaining HTS at this time may send the approval for the job (send_Approval).

Following the same guidelines, the right-side of Figure 3.3 depicts the Machine state diagram.

## 3.4 Troll-Specification

Based on the OMTROLL-diagrams in Section 3 we now derive a part of the TROLL-specification of the scenario. We will specify the relevant proportions of the classes HTS (and MACHINE in the appendix).

```
object class HTS
  identification
    ByHid:(hid)
  attributes
    hid:nat
      constant.
    rtime:nat
      initialized 0.
    ntime:nat
      initialized 0.
    answers:set(tuple(partner:|MACHINE|,costs:real)).
      hidden
      initialized emptyset.
    offers:set(tuple(offender:|HTS|,costs:real)).
      hidden
      initialized emptyset.
    partner:|MACHINE|
      initialized nil.
    job:|JOB|
      derived head(Jobs).ByHid().
    rlimit:nat
      constant
      initialized 20.
    nlimit:nat
      constant
      initialized 100.
    my_offer:nat
    best_offer:nat
    exist_offer:bool
      derived
        (my_offer≠0).
    partner_required:bool
      derived      max_partners:nat
      derived
        cnt(toSet(MACHINE)).
  components
    Jobs:list(JOB).
  events
    start
      birth
    receive_Job(j:JOB)
      enabled
        previous start or abort
        previous send_AbortA or send_AbortB
      changing
        Jobs:=Jobs.insert(j)
      calling
        calc_OfferA, request_Partner.
    receive_Partner(m:|MACHINE|,costs:nat)
      enabled
        rtime≤rlimit
        sometime request_Partner
        sincelast receive_Job
      changing
        answers:=answers+
          mk-set(mk-tuple(m,costs))
      calling
        wait_Answer, calc_Offer.
    receive_Offer(h:|HTS|,costs:nat)
      enabled
        ntime≤nlimit
      changing
        offers:=offers+mk-set(mk-tuple(h,costs))
      calling
        wait_Offer, abort.
  request_Partner(j:JOB)
    enabled
      partner_required=true
    calling
      foreach m:MACHINE
        do m.receive_RequestD(self,j) od.
      foreach m:MACHINE
        do m.receive_RequestO(self,j) od.
  send_Offer
    enabled
      exist_offer=true and ntime≤nlimit
    calling
      foreach h:HTS
        do h.receive_Offer(self,my_costs) od.
  send_AbortA
    enabled
      rtime>rlimit
    calling
      foreach h:HTS
        do h.receive_Offet(self,0) od.
  send_AbortB
    enabled
      exist_offer=false or ntime>nlimit
    calling
      foreach h:HTS
        do h.receive_Offer(self,0) od.
  send_Approval
    enabled
      ntime>nlimit
    calling
      {job.typ=DEMAND}
        job.Klient.receiveApprovalD
          (self,job.ByJid,my_offer)
      {job.typ=OFFER}
        job.Klient.receiveApprovalO
          (self,job.ByJid,my_offer)
  wait_Answer
    enabled
      rtime<rlimit and cnt(answers)<max_answer
      sometime request_Partner
      sincelast receive_Job
    calling
      send_AbortA.
  wait_Offer
    enabled
      my_offer=best_offer
    calling
      send_Approval.
  calc_OfferA
    enabled
      partner_required=false
    changing
      offers:=offers+mk-set(mk-tuple(self,costs))
```

```
    calling
      send_Offer,
      send_AbortB.
  calc_OfferB
    enabled
      rtime≤rlimit or
      cnt(partners)=max_partners
    changing
      offers:=offers+mk-set(mk-tuple(self,costs))
      /* not specified here */
    calling
      send_Offer, send_AbortB.
  abort
    enabled
      my_offer>best_offer
    changing
      offers:=emptyset
      answers:=emptyset
      Jobs:=Jobs−job.
end object class.
```

After achieving this analysis / specification phase, we have now a formal, high-level specification as input for the next phase.

# Chapter 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 [AS99b] for more detail.

## 4.1 Template Signature Specification

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

- The object states are terms of the form $\langle Id|atr_1 : val_1, ..., atr_k : val_k, at\_bs_1 : val'_1, ..., at\_bs_{k'} : val'_s\rangle$; where $Id$ is an observed object identity taking its values from an appropriate abstract data type $OId$; $atr_1, .., atr_k$ are the local, hidden from the outside, attribute identifiers having as actual values respectively $val_1, .., val_k$. The observed part of an object state is identified by $at\_bs_1, ..., at\_bs_s$ and their associated actual values are $val'_1, ..val'_s$.
- For exhibiting intra-object concurrency, we introduce a simple yet powerful axiom, called 'splitting / recombination' axiom permitting to split (resp. recombine) the object state out of necessity. As described in more detail later, this axiom, that can be described as follows: $\langle Id|attrs_1{}^2, attrs_2\rangle = \langle Id|attrs_1\rangle \oplus \langle Id|attrs_2\rangle$, allows us in particular, first, to exhibit intra-object concurrency[3]. Second, it provides a meaning to our notion of observed attributes by allowing separation between intra- and inter-component evolution (see later).
- In addition of conceiving messages as terms —that consist of message name, the identifiers of the objects the message is addressed to, and, possibly, parameters— we make a clear distinction between internal, local messages and the external as imported or exported messages. Local messages allow for evolving the object states of a given class, while the external ones allow for communicating between different classes using exclusively their observed attributes.

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

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

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

## 4.2 Template and Class Specification

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

- The places of the CO-Net are precisely defined by associating with each message generator one place that we called 'message' place. Henceforth, each message place has to contain message instances, of a specific form, addressed to objects (and not yet performed). In addition to these message places, we associate with the object sort one 'object' place that has to contain the current object states of this class. Note also that 'external' places associated with external messages will be drawn with bold circles.

- The CO-Net transitions reflect the effect of messages on the object states to which they are addressed. Also, we make distinction between local transitions that reflect the object states evolution and the external ones modeling the interaction between different classes. The requirements to be fulfilled for each transition form are given in the subsection below. The input arcs are annotated by the input conditions, while the output arcs are labelled by the created tokens. Both inscriptions are defined as multisets of terms respecting the type of their input and/or output places—the associated union operation is denoted by $\oplus$.

- Conditions may be associated with transitions. They involve attribute and/or message parameters variables.

## 4.3 CO-Nets : Semantical Aspects

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

### 4.3.1 Evolution of Object States in Classes

For the evolution of object states in a given class, we propose a general pattern that has to be respected in order to ensure the encapsulation property—in the sense that no object states or messages of other classes participate in this communication—as well as the preservation of the object identity uniqueness. Following such guidelines and in order to exhibit a maximal concurrency, this evolution schema is depicted in Figure 4.1, and it can be intuitively explained as follows: The contact of the just relevant parts of some object states of a given $Cl$, —namely $\langle I_1|attrs_1\rangle$[4] ;..; $\langle I_k|attrs_k\rangle$— with some messages $ms_{i1},..,ms_{ip},ms_{j1},..,ms_{jq}$—declared as *local or imported* in this class— and under some conditions on the invoked attributes and message parameters results in the following effects:

- The messages $ms_{i1},..,ms_{ip},ms_{j1},..,ms_{iq}$ vanish;

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

- The state change of some (parts of ) object states participating in the communication, namely $I_{s1}, .., I_{st}$. Such change is symbolized by $attrs'_{s1}, .., attrs'_{st}$ instead of $attrs_{s1}, .., attrs_{st}$. The other (unchanged part of) object states are denoted by $attrs_{i_1}, .., attrs_{i_r}$ so that $\{i_1, ...i_r\} \cup \{s_1, .., s_t\} = \{1, .., k\}$[5].

- Deletion of some objects by sending explicitly delete messages for such objects.

- New messages are sent to objects of $Cl$ , namely $ms'_{h1}, .., ms'_{hr}, ms'_{j1}, .., ms'_{jq}$.

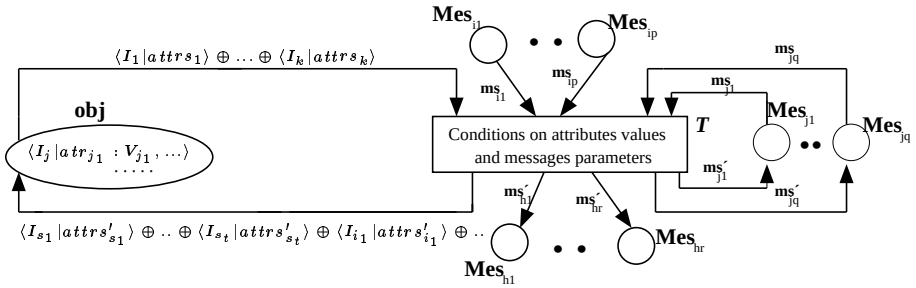


Figure 4.1: The Intra-Component Evolution Pattern

### 4.3.2 Rewriting rules governing the CO-Nets behaviour

Each CO-Net transition is captured by an appropriate rewriting rule interpreted into rewrite logic. Following the intra-component evolution pattern in figure 4.1, the general form of rewrite rules that we associate with it takes the following form:

$$T : (Ms_{i_1}, ms_{i_1}) \otimes .. \otimes (Ms_{i_p}, ms_{i_p}) \otimes (Ms_{j_1}, ms_{j_1}) \otimes .. \otimes (Ms_{j_q}, ms_{j_q}) \otimes$$
$$(obj, \langle I_1|attrs_1\rangle \oplus .. \oplus \langle I_k|attrs_k\rangle) \Rightarrow (Ms_{h_1}, ms'_{h_1}) \otimes ... \otimes (Ms'_{h_r}, ms'_{h_r}) \otimes$$
$$(Ms_{j_1}, ms'_{j_1}) \otimes .. \otimes (Ms'_{j_q}, ms'_{j_q}) \otimes (obj, \langle I_{s_1}|attrs'_{s_1}\rangle \oplus .. \oplus \langle I_{st}|attrs'_{st}\rangle \oplus \langle I_{i_1}|attrs'_{i_1}\rangle \oplus .. \oplus$$
$$\langle I_{i_r}|attrs'_{i_r}\rangle)$$
$$if\ Conditions \text{ and } M(Ad_{Cl}) = \emptyset \text{ and } M(Dl_{Cl}) = \emptyset$$

**Remark 4.3.1** *The operator $\otimes$ is defined as a multiset union and allows for relating different places identifiers with their current marking. Moreover, we assume that $\otimes$ is distributive over $\oplus$ i.e. $(p, mt_1 \oplus mt_2) = (p, mt_1) \otimes (p, mt_2)$ with $mt_1$, $mt_2$ multiset of terms over $\oplus$ and $p$ a place identifier. In the above rule, the condition $M(Ad_{Cl}) = \emptyset$ and $M(Dl_{Cl}) = \emptyset$ ensures that creation and deletion of objects have to be performed at first. In other words, before performing the above rewrite rule, the markings in the $Ad_{Cl}$ as well of the $Dl_{Cl}$ places have to be empty. This particularly allows to avoid inconsistency like the manipulation of an object that is already logically deleted (i.e. a corresponding delete message was sent) and not really or 'physically' deleted (i.e. no firing of the corresponding transition). Finally, please note that the selection of just the invoked parts of object states, in this evolution pattern, is quite possible because of the splitting /recombination axiom—that has to be performed before and in accordance with each invoked state evolution.*

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

## 4.4 CO-Nets : More Advanced Constructions

So far, we have presented only how the CO-Nets approach allows for conceiving independent classes. In what follows, we show how more complex systems can be constructed using advanced abstraction mechanisms, especially inheritance and interaction between classes. However, according to our running example, where no form of inheritance is used, only the component interaction will be presented.

### 4.4.1 Interaction between classes

Taking into account that object states evolution in classes is ensured by the intra-component pattern specified in figure 4.1, and in order to ensure the encapsulation property which stipulate that the internal part of an object state as well as the local messages have to be hidden from the outside, we propose an appropriate inter-component interaction pattern for communicating with these classes exclusively through their *explicitly* declared as observed attributes and external messages.

More precisely, as depicted in figure 4.2 such inter-component interaction may be made explicit as follows: The contact of some external parts of some object states namely $\langle I_1|attrs\_ob_1\rangle$, ..., $\langle I_k|attrs\_ob_k\rangle$, that may belong to different classes namely $C_1$, ..., $C_m$, with some external messages $ms_{i1}, .., ms_{ip}$, $ms_{j1}, .., ms_{jq}$ defined in such classes and under some conditions on attribute values and parameter 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 participating part of object states remain unchanged (i.e. deletion or creation of part of states is not allowed).
- new external messages (that may involve deletion/creation ones) are sent to objects of different classes, namely $ms'_{h1}, .., ms'_{hr}, ms'_{j1}, .., ms'_{jq}$.

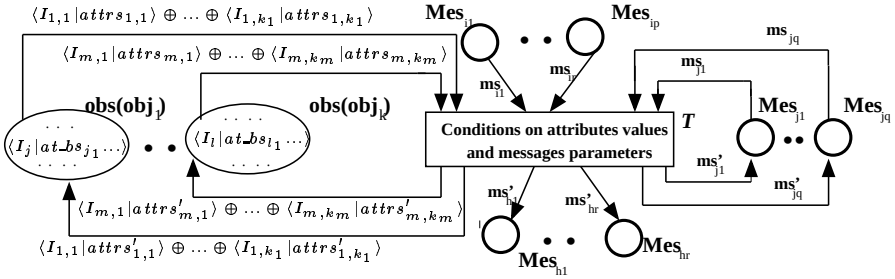


Figure 4.2: The Inter-Component Interaction pattern.

# Chapter 5

# Translating OMTROLL Specification into CO-NETS

Because of the object oriented setting on which both CO-Nets and OMTROLL approaches are based, (OO) systems specified using OMTROLL can be translated theoretically in straightforward way into the CO-Nets approach. Nevertheless, due to the different views in OMTROLL specification—particularly the communication diagram, the state diagram and the TROLL textual part—we have to integrate them judiciously and in coherent way so that we result in a CO-NETS specification as a fully distributed, autonomous and cooperative components. More precisely, given an OMTROLL specification, its main concepts (addressed in the example[1]) can be expressed in terms of those of CO-Nets by following two steps: OMTROLL signature translation and OMTROLL behaviour translation.

## 5.1 OMTROLL signature translation

This translation mainly concerns attribute and event descriptions, and it can be made precise as follows.

1. 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. In order to have just one and a uniform view, besides these attributes that are *explicitly* defined in the TROLL part, the state change within a given class which modeled using state diagram can be naturally conceived as an *additional attribute* (that we called state). This coherent view, that we have already applied to state-oriented Mondel specifications in [AS99a], allows us to integrate consistently the state machine specification with the TROLL specification part.

3. All event generators will be regarded as messages by adding to their arguments the identity sort of the invoked object. Moreover, by taking profit of the communication diagram, all events that are sent (i.e. represented by a corresponding arrow) to another class have to be defined as *exported* ones (and just reporting them as *imported* in the received class). Such external messages should include at least two object identities in their arguments (i.e. the sender and the receiver identities).

[1] Due to absence of subclasses, inheritance will not be addressed here.

4. From the explicit effect of these external messages described in TROLL specification part, it is quite possible to extract the *observed* attributes part as those which are explicitly involved (in the changing clause) in such effect.

**Example 5.1.1** *With respect to the described* CO-NETS *general form of template signature, the corresponding template signatures of the HTS and Machine components are described in what follows:*

**HTS template signature:** *First we have to specify the data types that are used in this template signature for specifying attribute values and/or event parameters. For the attributes we have:* **Nat** *for rtime, ntime, rlimit, nlimit, my_offer, best_offer, max_partner,* **Bool**, *List_answ, List_offr as list of tuple for the answers and offers attributes respectively, and the* **StateH** *for capturing the different state through which the HTS may transit (represented in the state machine diagram). For the event parameters the (same) sorts are used. All these data types will be referred to as HTS-DATA and specified using an OBJ notation as follows:*

```
obj HTS-DATA is
  protecting Id.Machine  Id.Job  Id.HTS  Bool  Nat  Real.
  sorts StateH.
  subsorts nil <answ < List_answ .
  subsorts nil < offer < List_offer .
  subsorts ni < Id.Job < List_job .
  subsorts answ  offer < Elt .
  subsorts List_answ  List_offer < List .
  op ready, received_j, contacted, received_A, calculated, sended, approved,
      received_O : → StateH .
  op  :  [_,_]:  Id.HTS  Real → offer .
  op  :  _._ :  offer  List_offr → List_offer [assoc.  Comm.  Id:nil] .
  op  :  <_,_>:  Id.Machine  Real → answ .
  op  :  _._ :  answ  List_answ → List_answ [assoc.  Comm.  Id:nil] .
  op  :  _._ :  Id.Job  List_Job → List_Job [assoc.  Comm.  Id:nil] .
  op  :  _in_:  Elt → Bool .
  var S : List .
  vars E, E' : Elt .
  eq E in nil = false .
  eq E in E'.S = if E == E' then true else E in S fi .
endfm.
```

*Using these data types and respecting the informal description given in the previous section, The corresponding HTS template signature is defined as follows:*

```
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 ⟨_|StateH : _, rtim : _, ntim : _, partn : _, job : _, part_requird : _, max_part : _, Jobs : _⟩ :
```

```
    Id.HTS StateH nat nat Id.Machine Id.job Bool nat List_job → External_HTS .
  (* local messages *)
  op Snd_offr, Snd_abort, Wait_answ : Id.HTS → Local_HTS_Mes .
  op Rcv_offr, Wait_offr, Calc_offr : Id.HTS Real (cost) → Local_HTS_Mes .
  op Abort, Approval : Id.HTS Job → Local_HTS_Mes .
  (* export messages *)
  op Request_P(artner): Id.Job Id:HTS Id.Machine → Exported_HTS_Mes .
  (* Imported messages *)
  op Rcv_Job, Rcv_P : OId Id.HTS Id.Job → Imported_HTS_Mes (* from the Machine Component *).
 vars H : Id.HTS ; J : Job ; S : List_answ.(* these variables are used in the corresponding net *).
 vars C : Real (cost) ; J : Job ; R, M, L : Nat ; Q : Boolean.
endo.
```

## 5.2 OMTROLL 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 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 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 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. All for all our translating ideas are sketched in the following table.

| OMTROLL **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 |
| state change in SM | additional attributes called State |
| events | messages with expliciting the identity of the invoked object |
| — enabled | Transition condition |
| — calling | messages sent |
| — changing | Transition effect |

**Example 5.2.1** *By applying the above translating ideas to our running example, we result in two* CO-NETS *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 interface another* CO-NET *is 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 5.1. In this net, besides the object place* **OBJ_HTS** *that contains all HTS object instances, with each message generator a corresponding place is associated. Places reflecting imported / exported messages are represented in bold. For the Job classes that is 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 part of the object state to which it is sent. Example of such effect is the transition* **RCV_JOB**, *where the message* **Receive_J(J,M,H)** *(with* **J, M, H** *denote respectively job identity, machine identity and HTS identity) enters into with the part of the object state* $\langle H|StateH : Ready, Job : Jb, Partner\_Rq : R\rangle$. *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)**.

## 5.3 Animating and Validating the CO-NETS Specification

One of the most advantage of the CO-NETS approach is its operational semantics expressed in rewriting logic; moreover, by introducing this splitting / recombination axiom there is a natural exhibition of intra-object concurrency. Besides that, the distinguishing advantage towards a more efficient rapid-prototyping is the resulting, from the OMTROLL specification, in autonomous yet cooperating CO-NETS components, where:

- 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.

However, because any real animation with symbolic computation necessitate more space, we limit ourselves to present the corresponding rewrite rules.

**Example 5.3.1** *By applying the general forms of rewrite rules, it is not difficult to generate the rewriting rules governing the behaviour of both* **HTS** *and* **Machine** *components as well as those of their interaction. Hereafter, we just present two rewrite rules of the HTS component; the remaining rules as well as those of the interaction between the HTS and Machine components are included in the appendix.*

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

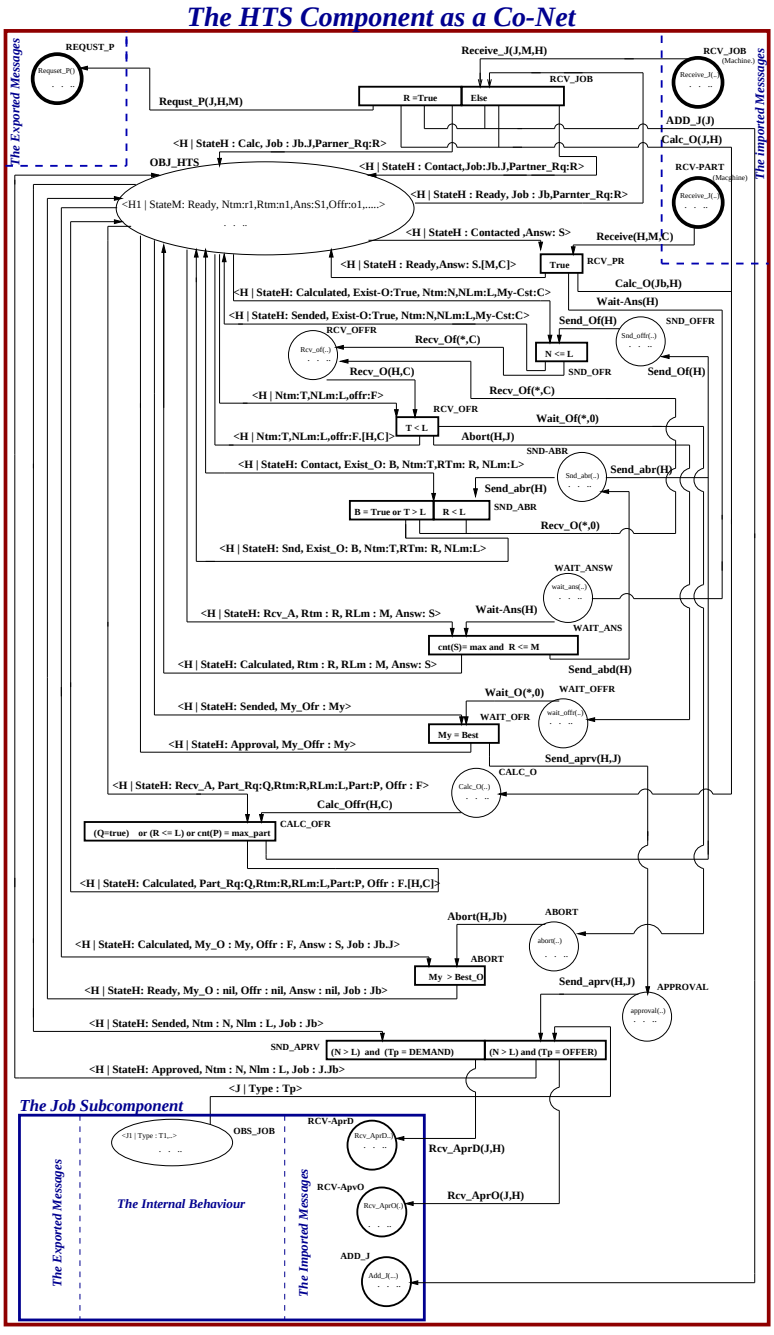


Figure 5.1: The HTS CO-NETS Specification

# Chapter 6

# Conclusion

In this paper, we present our first results towards an adequate methodology for developing reliable advanced information systems as fully distributed, autonomous yet cooperative components. The methodology is intended to deal with main phases in developing such systems, namely, the analysis, specification, validation and implementation phases. For the analysis / specification, we followed the widely accepted OMTROLL approach. For validation purpose as well as for coping with more advanced requirements—including intra- and inter-object concurrency, synchronous / asynchronous communication, component construction—, we have proposed the CO-NETS approach, a new form of object oriented Petri nets based model, that we are developing. For this aim, a very natural, semi-automatic translation from OMTROLL into CO-NETS has been introduced. This allow us particularly for shifting from the OMTROLL system specified as a community of objects towards a fully CO-NET cooperative components. Moreover, due to our CO-NETS interpretation in rewriting logic, rapid-prototypes may be generated using concurrent rewriting techniques accompagnied with a simultaneous graphical animation of the corresponding nets. Such prototypes may be efficiently generated due to the possibility of handling each component separately, besides the interaction of such components.

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.

However, after achieving this very important and first steps towards a suitable for developing advanced information systems, we are conscious that much a work remain ahead. First, we are planning a deeper study for an appropriate and efficient translation of the generated rewriting rules into JAVA programs. Second, we are also working for developing a complete software environment for the CO-NETS approach, including particularly an editor / simulator; for the rewriting techniques we are planing to use current implementation of the Maude language [Mes98]. Last but not least, for coping with the runtime evolution due to frequent change we are working on an appropriate reflective extension of the CO-NETS approach.

# Bibliography


[AF95] D.E. Avison and G. Fitzgerald. *Informations Systems Development : Methodologies, Techniques and Tools.* McGraw-Hill, 1995.

[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.

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

[GKFK+98] A. Grau, J. Küster Filipe, M. Kowsari, S. Eckstein, R. Pinger, and H.-D. Ehrich. The TROLL Approach to Conceptual Modelling: Syntax, Semantics and Tools. In T.W. Ling, S. Ram, and M.L. Lee, editors, *Proc. of the 17th Int. Conference on Conceptual Modeling (ER'98), Singapore*, Lecture Notes in Computer Science, Vol. 1507, 1998.

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

[HG94] R. Herzig and M. Gogolla. An Animator for the Object Specification Language TROLL *light*. In V.S. Alagar and R.Missaoui, editor, *Proc. Colloquium on Object Orientation in Dtabases and Software Engineering (COODBSE'94)*, pages 4–17, 1994.

[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.

[JWH+94] R. Jungclaus, R. J. Wieringa, P. Hartel, G. Saake, and T. Hartmann. Combining TROLL with the Object Modeling Technique. In B. Wolfinger, editor, *Innovationen bei Rechen- und Kommunikationssystemen. GI-Fachgespräch FG 1: Integration von semi-formalen und formalen Methoden für die Spezifikation von Software*, Informatik aktuell, pages 35–42, Springer-Verlag, 1994.

[Lan95] K. Lano. *Formal Object-Oriented Development.* Springer, London, 1995.

[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. In G. Agha, P. Wegner, and A. Yonezawa, editors, *Research Directions in Object-Based Concurrency*, pages 314–390, The MIT Press, 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.

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

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, 1991.

[WJH+93] R. Wieringa, R. Jungclaus, P. Hartel, T. Hartmann, and G. Saake. OMTROLL – Object Modeling in TROLL. In U. W. Lipeck and G. Koschorreck, editors, *Proc. of the Intern. Workshop on Information Systems – Correctness and Reusability (IS-CORE'93)*, Technical Report No. 1/93, pages 267–283, University of Hannover, 1993.

[WK96] M. Wirsing and A. Knapp. A Formal Approach to Object-Oriented Software Engineering. In J. Meseguer, editor, *Proc. of the First Inter. Workshop on Rewriting Logic*, Vol. 4, Electronic Notes in Theoretical Computer Science, 1996.

# Appendix A

# TROLL specification of the machine class

```
object class MACHINE
  identification
    ByMid:(mid)
  attributes
    mid:nat
      constant.
    rtime:nat
      initialized 0.
    ntime:nat
      initialized 0.
    rlimit:nat
      initialized 100.
    nlimit:nat
      constant
      initialized 20.
    job:|JOB|
      initialized nil.
    hts:  |HTS|
      initialized nil.
  components
    Source:  SOURCE
    Dest:  DESTINATION
    Object:  OBJECT
  events
    start
      birth
      calling
        {Source.available=0} send_JobD,
        {Source.available>0} process_Object.
    send_JobD
      enabled
        Source.available=0
      changing
        ntime:=0,
        job:=job.create(DEMAND,self)
      calling
        foreach h:HTS
           do h.receive_Job(job) od, abort_Demand.
        abort_Demand.
    abort_Demand
      enabled
        ntime>nlimit
      calling
        send_JobD, process_Object.
    receive_ApprovalD(h:|HTS|,j:|JOB|,offer:real)
      enabled
        sometime send_JobD
        sincelast receive_ApprovalD
        and j=job
      changing
        hts:=h,
        rtime:=0,
        rlimit:=offer
      calling
        abort_ApprovedD.
    abort_ApprovedD
      enabled
        rtime>rlimit
      calling
        send_JobD, process_Object.
    receive_Object(o:  |OBJECT|)
      enabled
        sometime receive_ApprovalD
        sincelast send_JobD
      calling
        send_JobD, process_Object.
    process_Object
      enabled
        Source.available>0
      calling
        next_Object, send_Job0.
    send_Job0
      enabled
        Dest.free=0
      changing
      ntime:=0,
        job:=job.create(OFFER,self)
      calling
        foreach h:HTS
           do h.receive_Job(job) od, abort_Offer.
    abort_Offer
      enabled
        ntime>nlimit
      calling
        send_Job0, next_Object.
    receive_Approval0(h:|HTS|,j:|JOB|,offer:real)
      enabled
        sometime send_Job0
        sincelast receive_Approval0
        and j=job
      changing
        hts:=h,
        rtime:=0,
        rlimit:=offer
      calling
        abort_Approved0.
    abort_Approved0
      enabled
        rtime>rlimit
      calling
        send_Job0,  next_Object.
    deliver_Object(!o:|OBJECT|)
      enabled
```

```
    sometime receive_Approval0
    sincelast send_Job0
   calling
    send_Job0, next_Object.
  next_Object
   enabled
    Dest.free>0
   calling
    process_Object, send_JobD.
  receive_RequestD(h:|HTS|,j:|JOB|)
   enabled
    Dest.free>0
   calling
    send_Partner(h,j).
  receive_Request0(h:|HTS|,j:|JOB|)
   enabled
    Source.available>0
   calling
    send_Partner(h,j).
  send_Partner(h:|HTS|,j:|JOB|)
   enabled
    previous receive_Request0 or
    previous receive_RequestD
   calling
    HTS(h).receive_Partner(self,...).
end object class
```

# Appendix B

# the Machine CO-NETS description

**Machine template signature:** The main data types associated with this machine template is the different states through which the machine may pass.

```
obj MACH-DATA is
  protecting Id.Machine  Id.Job  Bool  Nat  Real.
  sorts StateM.
  op requested, ready, demand, approved_D, offer, processed, approved_0,
    approved_D : → StateM .
endfm.
```

Using these data types and respecting the informal description given in the previous section, The corresponding MCH template signature is defined as follows:

```
obj Machine-signature is
  protecting Object-state HTS-DATA  MACH-DATA .
  subsort Id.Machine < OId .
  subsort Local_Machine  External_Machine < Machine .
  subsort RCV_OFFR  ABORT_DEMAND  PROCESS_OBJECT  NEXT_OBJECT  DLV_OBJECT
      ABORT_APPROVAL < Local_Machine_Mes .
  subsort SND_JOB  CREAT_JOB  SND_PARTNER< Exported_Machine_Mes .
  (* local attributes *)
  op ⟨_|Rtm : _, Ntm : _, Rlm : _⟩ :  Id.Machine  nat  nat  nat → Local_Machine.
  (* observed attributes *)
  op ⟨_|StateM : _, job : _, hts : _, source : _, dest : _, object : _⟩ :
     Id.Machine  StateM  Id.job  Id.HTS  Id.source  Id.dest  Id.object → External_Stock .
  (* local messages *)
  op Rcv_of, Abr_dmd, Proc_obj, Nxt_obj, Abr_aprv, Dilv_obj :  Id.Machine
      Id.Job → Local_Machine_Mes .
  (* export messages *)
```

```
  op Snd_job, Creat_job :  Id.Machine  OId → Exported_Machine_Mes .
  op Snd_part :  Id.Machine  OId → Exported_Machine_Mes .
 (* Imported messages *)
  op Rcv_RqustO, Rcv_RqustD : OId  Id.Machine → Imported_Machine_Mes .
 vars M : Id.Machine .(* these variables are used in the corresponding net *).
endo.
```

**Machine CO-NET component:** The net reflecting the machine behaviour is depicted in figure B.1. Example of this behaviour is the transition **RECV** the asynchronous and autonomous state change of the machine. More precisely, if the machine is in the state **Start** it may enter at any time the new **Ready**. Moreover, if the observed attribute in the corresponding **source** subcomponent class is valuated to 0 then there a simultaneous sending of the messages **Snd_job(J, M)** and **Process_job(J, M)** to their corresponding places.

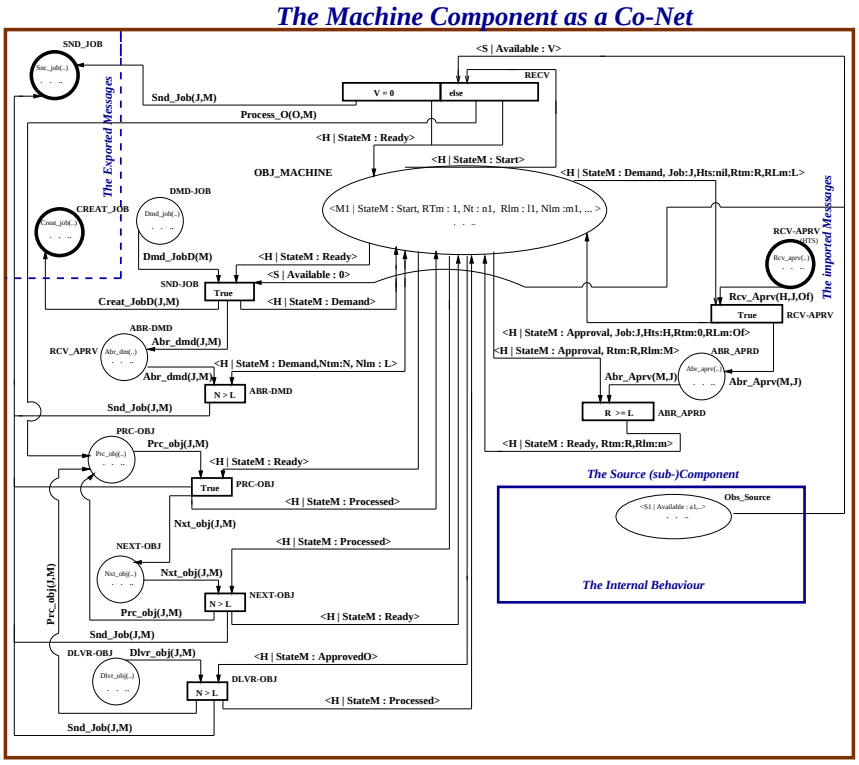


Figure B.1: The Machine CO-NETS Specification

**HTS CO-NET Rewriting rule:**

**SND_OFR:** $(SND\_OFFR, Send\_Of(H)) \otimes (OBJ - HTS, \langle H | StateH : Calculated, Exist\_Of : True, Ntm : N, Nlm : L, My\_Cost : C\rangle)$
$\Rightarrow (OBJ - HTS, \langle H | StateH : Sended, Exist\_Of : True, Ntm : N, Nlm : L, My\_Cost : C\rangle) \otimes (RCV\_OFFR, Recv\_Of(*^1, C))\ if\ N \leq L$

**RCV_OFR:** $(RCV\_OFFR, Recv\_Of(*, C)) \otimes (OBJ - HTS, \langle H | StateH : Sended, Ntm : T, Nlm : L, Offr : F\rangle) \Rightarrow (OBJ - HTS, \langle H | StateH : Received, Ntm : T, Nlm : L, Offr : F.[H, C]\rangle)$
$\otimes (WAIT\_OFR, Wait\_Of(*, 0)) \otimes (ABORT, abort(H, J))\ if\ T`L$

**SND_ABR:** $(SND\_ABR, Send\_abr(H)) \otimes (OBJ - HTS, \langle H | StateH : Contacted, Exist\_O : B, Ntm : T, Rtm : R, Nlm : L\rangle)$

[1] The notation '*' means that such a message is broadcasted to all HTS.

$\Rightarrow if\ (B = true\ or\ T > L)\ then\ (OBJ - HTS, \langle H|StateH : Ready, Ntm : T, Nlm : L, Rtm : R\rangle) \otimes (RCV\_OFR, Recv\_Of(*, 0))\ else\ if\ R < L\ then\ (OBJ - HTS, \langle H|StateH : Contacted, Ntm : T, Nlm : L, Rtm : R\rangle) \otimes (RCV\_OFR, Recv\_Of(*, 0))$

**WAIT_ANS:** $(WAIT\_ANSW, Wait\_answ(H)) \otimes (OBJ - HTS, \langle H|StateH : Received\_A, Rtm : R, RLm : M, Answ : S\rangle)$
$\Rightarrow if\ (cnt(S) = max\ or\ R \leq M)\ then\ (OBJ - HTS, \langle H|StateH : Calculated, Rtm : R, RLm : M, Answ : S\rangle) \otimes (SND\_ABR, Send\_abr(H))$

**WAIT_OFR:** $(WAIT\_OFR, Wait\_Of(*, 0)) \otimes (OBJ - HTS, \langle H|StateH : Sended, My\_Offr : My\rangle)$
$\Rightarrow if\ (My = best\ then\ (OBJ - HTS, \langle H|StateH : Approved, My\_Offr : My\rangle) \otimes (SND\_APRV, Send\_approv(H, J))$

**CALC_OFR:** $(CALC\_OFR, Calc\_Offr(H, C)) \otimes (OBJ - HTS, \langle H|StateH : Received\_A, Partner\_Rq : Q, Rtm : R, Rlm : L, Limit\_Part : P, Offr : F\rangle)$
$\Rightarrow if\ (Q = true\ or\ R \leq L\ or\ cnt(P) = max\_part)\ then\ (OBJ - HTS, \langle H|StateH : Calculated, Partner\_Rq : Q, Rtm : R, Rlm : L, Limit\_Part : P, Offr : F.[H, C]\rangle \otimes (SND\_OFFR, Send\_Of(H))$

**ABORT:** $(ABORT, Abort(H, Jb)) \otimes (OBJ - HTS, \langle H|StateH : Calculated, My\_Offr : My, Offr : F, Answ : S, Job : Jb.J\rangle)$
$\Rightarrow if\ (My > Best)\ then\ (OBJ - HTS, \langle H|StateH : Ready, My\_Offr : nil, Offr : nil, Answ : nil, Job : Jb\rangle)$

**SND_APRV:** $(SND\_APRV, Send\_aprv(H, J)) \otimes (OBJ - HTS, \langle H|StateH : Sended, Ntm : N, Nlm : L, Job : Jb\rangle) \otimes (OBJ_JOB, \langle H|Type : Tp\rangle)$
$\Rightarrow if\ (N > L\ and\ Tp = DEMAND)\ then\ (OBJ - HTS, \langle H|StateH : Approved, Ntm : N, Nlm : L, Job : Jb.J\rangle) \otimes (RCV\_APRV, Recv\_aprvD(J, H))\ else\ if\ N > L\ and\ Tp = OFFER)\ then\ (OBJ - HTS, \langle H|StateH : Approved, Ntm : N, Nlm : L, Job : Jb.J\rangle) \otimes (RCV\_APRV, Recv\_aprvO(J, H))$

**HTS-Machine interaction** CO-NET**:** Besides the already described proper behaviour of both Machine and HTS, both components interact with each other exclusively through their external attributes and imported / exported messages. This interaction depicted in the CO-NET in figure B.2, have to be captured from the communication diagramm and the correponding precise behaviour in TROLL part. First, remark that only the observed part of different object states and observed messages are concerned by such interaction. In other words, we abstract away from internal behaviour of each component. For instance, in the transition **RQ_P**, provided that the observed attribute **Partner_Rq** is valuated to *true*, the result of a sending of a message **Rq_partner(H,J)** from the HTS component is the appearance of two imported messages in the machine components, namely, **Rcv_RqO(H,*,J)** and **Rcv_RqD(H,*,J)**.

**HTS-Machine interaction** CO-NET **rewriting rules:**

**RQ_PART:** $(RQS\_PART, Rqs\_Partn(H, J)) \otimes (OBS - HTS, \langle H|StateH : Receive\_J, Partner\_rq : True\rangle)) \otimes (OBS - Machine, \langle M|StateM : Requested\rangle))$
$\Rightarrow (OBS - HTS, \langle H|StateH : Receive\_J, Partner\_rq : True\rangle)) \otimes (OBS - Machine, \langle M|StateM : Requested\rangle)) \otimes (RCV\_RQO, Rcv\_RqO(H, *, J)) \otimes (RCV\_RQD, Rcv\_RqD(H, *, J))$

**SND_JOB:** $(SND\_JOB, Send\_job(M, J)) \otimes (OBS - HTS, \langle H|StateH : Calculated, Partner\_rq : True\rangle)) \otimes (OBS - Machine, \langle M|StateM : Demand, Ntm : N\rangle)) \otimes (OBS - SOURCE, \langle S|Available : 0\rangle)) \Rightarrow (OBS - HTS, \langle H|StateH : Calculated\rangle)) \otimes (OBS - Machine, \langle H|StateM : Demand, Ntm : 0\rangle)) \otimes (OBS - SOURCE, \langle S|Available : 0\rangle)) \otimes (RCV\_JOB, Rcv\_job(M, J, *)) \otimes (CREAT\_JOB, Creat\_job(J))$

**SND_PART:** $(SND\_PART, Send\_part(M, J)) \Rightarrow (RCV\_PART, Recive\_P(M, *, J))$

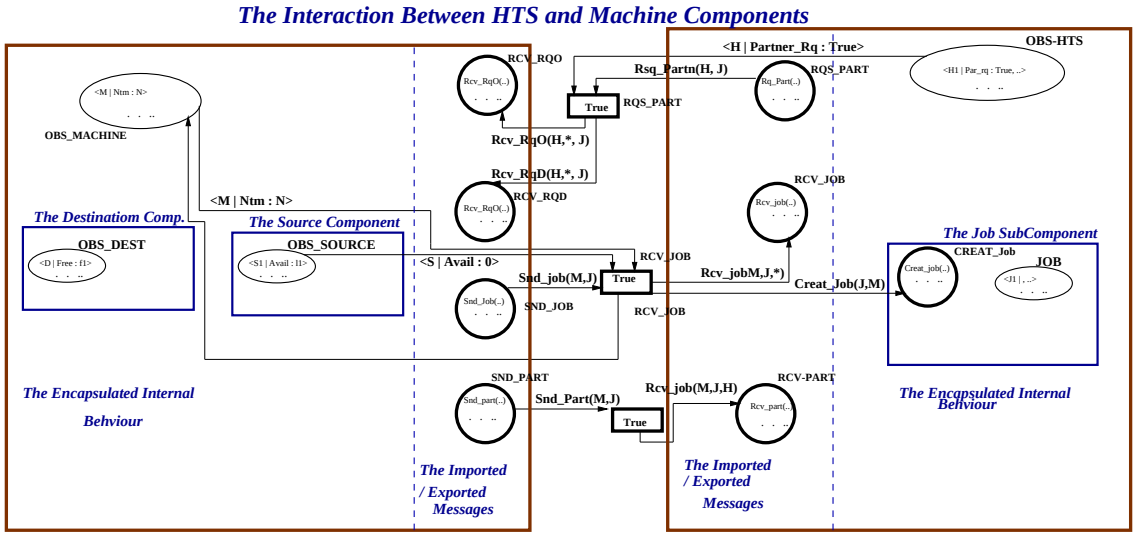


Figure B.2: The Interaction HTS-Machine CO-NETS