Distributed System Specification Using CO-OPN[*]

Didier BUCHS[†‡] & Nicolas GUELFI[†]

[†]**LRI**, URA 410 CNRS,
Bât. 490, University of PARIS XI, ORSAY
91405 ORSAY FRANCE
Tel: (33 1) 69 41 64 32 Fax: (33 1) 69 41 65 86
e-mail: GUELFI@lri.lri.fr

[‡]**CUI**, University of Geneva
12 rue du Lac,
1207 Genève, SWITZERLAND
Tel :(41 22) 787 65 80 Fax : (41 22) 735 39 05
e-mail: BUCHS@cui.unige.ch

Abstract

The CO-OPN specification formalism for large concurrent systems is presented. CO-OPN includes specification of data types and concurrency through the use of modular algebraic specification and structured algebraic nets. This structure follows the object oriented paradigms. Distributed system specification is studied with CO-OPN through a case study of communication components: the Transit-Node. The interesting specification features of CO-OPN -structuring, refinements and concurrency- are discussed.

1. Introduction

It is now accepted that real specification languages are needed for the development of systems. We believe that specification languages are based on a formal theory which gives a precise semantics. They must fully capture the problem to be described in an abstract way. Moreover, the specification formalism has to be able to formalize the implementation process and the proof techniques. In the case of distributed systems, concurrent and data type features have to be included in the specification language. For real problems, there exists a lack of structuring principles in actual specification languages. Specification formalisms including the formalization of data types and concurrency, as Lotos [14] or Algebraic Petri nets [22], [13], [10], [1], [19] cannot describe large systems.

CO-OPN (Concurrent Object Oriented Petri Nets) [4] has been designed to fulfill this requirement, that is, to obtain a complete specification formalism for complex concurrent systems and languages. This aim is reached by using a modular algebraic specification language (PLUSS [8], [9]) connected with a concurrency part in terms of nets. The algebraic specification language [11] is used to define the data structures needed in the specification of the system. Precisely, we use algebraic nets as modules of the specification, to describe the system in a structured way. CO-OPN can be seen as an extension of algebraic nets including modularity by means of a simple notion of objects. We adopt the software engineering viewpoint to formally characterize the object-oriented approach [12] by defining, in an object, methods for encapsulation and synchronization expressions for abstraction. An object has an internal behavior defined by the algebraic net. The external one is expressed using methods (parameterized transitions) for the object interface and synchronization expressions to describe the call of other methods. Different objects can be linked by "symbolic links" (distinct from "net arcs") between events to express synchronization. These symbolic links are oriented, representing the call of a method by another one or by an internal transition. Each link represents an atomic component of an expression built with the two binary synchronization operators: simultaneous and sequential. Each event associated to a synchronization expression corresponds to an abstraction of the events contained in its synchronization expression.

The semantics of our model [4] is defined by introducing a notion of distributed transition system (DTS). For a given CO-OPN specification, we build its DTS using inference rules on object markings. The concurrency expressed in the DTS corresponds totally with that of the modeled system: a transition of a DTS corresponds to a vector of multi-sets of events, one component for each object of the specification. The stepwise construction of the DTS by inferences follows the object hierarchy induced by the synchronization expressions. We also propose an incremental construction

[*] *This work has been supported by the "Fond National Suisse de la Recherche Scientifique" and by the "Esprit" Basic Research Action, "DEMON", n° 3148.*

of the DTS that fulfills the requirement that a modular semantics has to describe the behavior of an object (a net) as a function of its internal behavior and of the behavior of its inter-related objects.

We have defined a formal framework of specification refinement [4], in order to get an implementation from an abstract CO-OPN specification. A CO-OPN specification is equivalent to an other one if there exist a bisimulation [17] between them. In a given system, an object can be replaced by another one, if they are bisimilar. The bisimulation between two DTS is defined for the same algebra (the semantics of the algebraic specification) on both systems. A second refinement notion is based on observationaly equivalent algebras: this allows a change of the data implementation without changing the behavior of the system.

In this paper we present a case study of a reactive system specification using the specification formalism CO-OPN. Through this case study we give formal techniques for modeling any kind of system or languages. We show that CO-OPN can be considered, in software engineering for distributed systems, as a formal framework for system specification. We explain how to specify a communication component of a distributed system, called "Transit Node" (T-Node). Then we show how to specify the upper layer describing a set of T-Nodes modeling a communication network . We present the refinement features (studied in depth in the extended version of this paper [5]) of CO-OPN. They are used to improve each system specification by refining it step-by-step.

The adequacy of CO-OPN for system specification has also been shown in other papers dealing with programming languages such as Actors [3], OCCAM-2 languages and Open Distributed Systems [6].

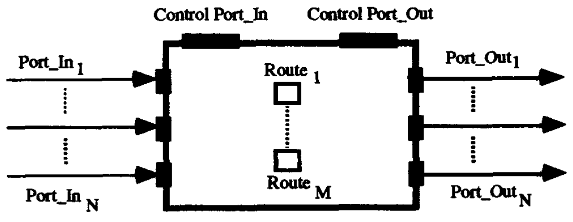
2. The specification of the T-Node

This case study was defined in the RACE Esprit project 2039 and is already specified without concurrency in [15] with the language PLUSS and in [16]. It consist of a simple T-Node where messages arrive, are routed, and leave the node. The informal specification reads as follows:

The system to be specified consists of a T-Node with :

- 1 control Port-In	- N Data Port-Out
- 1 control Port-Out	- M Routes Through

- N Data Port-In
(The limits of N and M are not specified.)



Transit node

Each port is serialized. All ports are concurrent to all others. The ports should be specified as separate, concurrent entities. Messages arrive from the environment only when a Port-In is able to treat them. The node is "fair". All messages are equally likely to be treated, when a selection must be made, and all messages will eventually pass through the node, or be placed in the collection of faulty messages. At the beginning, the T-Node can only receive messages on its Control Port-In to initialize the T-Node as described below; and the Control Port-Out is used to transmit the faulty messages.

The Control Port-In accepts and treats the following three messages :

- Add-Data-Port-In(n) & Add-Data-Port-Out(n) : gives the node knowledge of a new port-in "n" and a new port-out "n". The node starts to accept and treat messages sent to the port-in, as indicated below with Data Port-In.
- Add-Routes(m,n(i),n(j),...) : gives the node knowledge of a route, associating the route m with the output data Ports : n(i),n(j),....
- Send-Faults : routes all faulty messages saved, if any, to Control Port-Out. The order in which the faulty messages are transmitted is not specified.

A Data Port-In accepts and treats only messages of the type:

- <Route(m),Data> : The Port-in routes the messages, unchanged, to any one (non-determinate) of the Data-Ports-Out associated with route m. (Note that a Data Port-Out is serialized - the message has to be buffered until the Data Port-Out can process it). The message becomes a faulty message if its transit time through the node (from initial receipt by a Data Port-In to transmission by a Data Port-Out) is greater than a constant time T.

Data Ports-Out and Control Port-Out accept messages of any type and will transmit the message out of the node. Messages may leave the node in any order. All faulty messages are saved until a Send-Faults command message causes them to be routed to Control Port-Out. Faulty messages are messages on the Control Port-In that are not one of the three commands listed, messages on a Data Port-In that indicate an unknown route, or messages whose transit time through the node is greater than T. Messages that exceed the transit time of T become faulty as soon as the time T is exceeded. It is possible for a faulty message to be not routed to Control Port-Out (because, for example, it has just become faulty, but has not yet been placed in a faulty message collection), but all faulty messages must eventually be sent to Control Port-Out with a succession of Send-Faults commands.

It may be assumed that a source of time (which signals each time interval) is available in the environment and does not need particular specification.

This previous description of the T-Node behavior will be the basis of our specification. The specification presented describes a minimal T-Node which does not deal with other problems such as:

- adding of a route referring to unknown ports
- adding of empty routes or adding to existing ports
- fairness problem

Another specification presented in [5], tries to answer these problems by offering an adapted specification. This specification is obtained through a modification (refinement) of the first one.

As said before, a CO-OPN specification is made of two parts: the algebraic specification part describing the data structures used in the system; and a set of objects (algebraic nets) using the algebraic specification part, thus specifying the whole system. The first question which comes out is: "which part of the system may I specify using algebraic specifications and which part with the net ?" Of course, there is no answer to this question. We believe that the net is well-suited for the main actions with their execution order relations, while the algebraic specification part may be used to specify basic data structures with their associated operations. In fact a CO-OPN specification could be made of one object with one place and a transition computing an algebraic term (representing the system behavior). In this case all the specification is made with the algebraic part only. On the contrary, we could imagine a CO-OPN specification with an algebraic part reduced to the minimum and substituted by net constructions. Of course, these considerations are neither realistic nor true. They only give an idea of the decomposition problem of a specification. As algebraic nets merge algebraic specifications in nets, this problem is crucial and complex. In this paper we only give the general principles of specification construction.

To specify the T-Node using CO-OPN we have to define a set of objects and the specification of the algebraic abstract data type needed in our objects. We first give the algebraic specification using PLUSS language, followed by the CO-OPN objects.

2.1. The Algebraic Specification Part

We give a partial specification of the sorts used in our specification. The axioms not given (see [5]) which describe the properties of the operations associated to the specification are nevertheless useful for testing, refining and proving specifications [7]. We will first introduce the generic algebraic specification, and then, the instantiations of them.

The hierarchy of algebraic specification modules is built using the following method. We first define the fundamental sorts (here : the natural numbers 'nat', the booleans 'bool' and the messages 'msg'). Then the generic sorts (here: 'set', 'couple and 'triplet') and finally, their instantiated modules (here: route, port, time, routes, store_port, ctrl_msg, data_msg, store_route, store_data, and fault). This hierarchy is given in figure 2 and the specifications are described below.

Generic specification

A generic specification is an algebraic specification associated to a parameter list. Each parameter is the name of a "parameter specification". A "parameter specification" (Item) defines the sorts that can be used in the related generic specification (Set).

generic spec Set (ITEM) ;
sort : set ; ....; **end**;

par Item
sort Item ; **end**;

1) **spec** Set_Of_Nat **as** Set (ITEM ⇒ Nat **by** m)
2) **where** m : item ⇒ nat
3) **renaming** set **into** set_of_nat; **end**;

In the generic specifications the following points are specified:

1) To indicate the morphism m for the substitution of the specification Item by the specification Nat

2) The morphism gives the correspondence between the sorts
3) The new name for the instantiation of the generic specification

The generic specification of 'set' describes the sets with operations 'elemof' and 'choose'. The predicate 'elemof' indicates if a given element is in a given set and the function 'choose' gives a chosen element of a set. Another generic specification is CP3 (resp. CP2) which describes the Cartesian product of three (resp. two) sorts. The projections on a chosen component are given with the functions 'π1','π2','π3' (resp. 'π1','π2').

Parameter sorts

The parameter sorts for the generic specification are defined to give a model to the parameters.

Basic sorts

The basic sorts are used for the values of the ports, the routes and the time. The sort 'port' defines the port number in the interval [1...N]. This sort is used to designate the input and the output ports. The sort 'route' defines the route number in the interval [1...M]. The time is modeled by natural numbers, each tick of a clock increments unity by unity the current time. The messages arriving to the node are specified by the sort 'msg' (not specified).

```
spec : Port as Nat ;
sort port;
renaming:
        nat into port;
end;
```

```
spec:Route as Nat ;
sort route;
renaming:
        nat into route;
end;
```

```
spec : Time as Nat;
sort time;
renaming:
        nat into time;
end ;
```

```
spec : Msg ;
sort msg ;
......
end;
```

Instantiated sorts

The routes are modeled by sets of ports with the sort 'routes'. The final destination of a message for a given route is a port chosen into the set of the possible output ports.

```
spec : Routes as Set(ITEM => Port by m );
where m: item => port;
renaming : set into routes;
end ;
```

The following instantiated sorts (which are not given but shortly described) are used for : the structure of the datas inside the T-Node, describing the enable ports and routes. The specification "Store_Port" is defined to associate at each port p_i a value of sort store_port "<p_i , b_i>", where b_i is a boolean indicating if the (input or output) port p_i can be used.

The specification "Data_Msg" defines the input messages as a tuple <dt_i, p_i, r_i> with dt_i of sort msg and corresponds to the message itself, p_i is the input port where the message arrives and r_i is the associated route.
The specification "Store_Route" defines each route as a tuple <r_i , R_i , b_i> with r_i of sort route, R_i a set of ports of sort "routes" and b_i indicates if the route r_i is enable or not.

The specification "Store_Data" is another specification related to the messages passing in the T-Node. An element of sort store_data is a tuple <dt_i, p_i, t_i> where dt_i is of sort msg and corresponds to the message itself, p_i is the input port and t_i indicates at what time the message dt_i entered into the node. Moreover, we define the sort 'control_msg' to describe the message sent to the 'T-Node' control part. To treat the faulty messages, we define a 'fault' sort indicating the possible error message appearing in the T-Node.

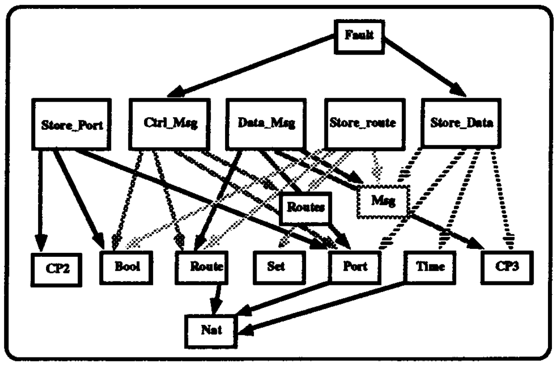


figure 2

2.2. The CO-OPN specification part

The first step of defining the CO-OPN system is to determine the set of concurrent objects which constitute the system. After structuring the system in independent parts, the methods of each object must be defined. In fact, the methods represent the interface of an object (they are the externally visible part of the object). The following step concerns the definition of the internal behavior of each object. We must stress that the concurrency of the system is modeled not only by the concurrency between the objects but also with the concurrency between events of each object. To complete the CO-OPN specification, the links between the objects must be explicitly given using 'synchronization expression' inside the events (internal transitions or methods) definitions. A synchronization expression can be associated to each event and is built using the two operators: "sequential" or "simultaneous". The possible operands are methods or synchronization expressions. Thus, a synchronization expression associated to an event represents distributed constraints over the CO-OPN object system. The T-Node can be divided into three parts (three CO-OPN objects). The first one concerns the treatment of the control messages arriving to the node. The second part corresponds to the reception of the data messages from the input ports and the routing of these messages through the T-Node to the output port. The thirds one concerns the time. We give a brief description of these objects but more details are given in [5]. The figure 3 corresponds to a partial (i.e: all information is not represented) graphical description of the three CO-OPN objects (the ellipses are algebraic nets) linked by the synchronization expressions (the stripped arcs).

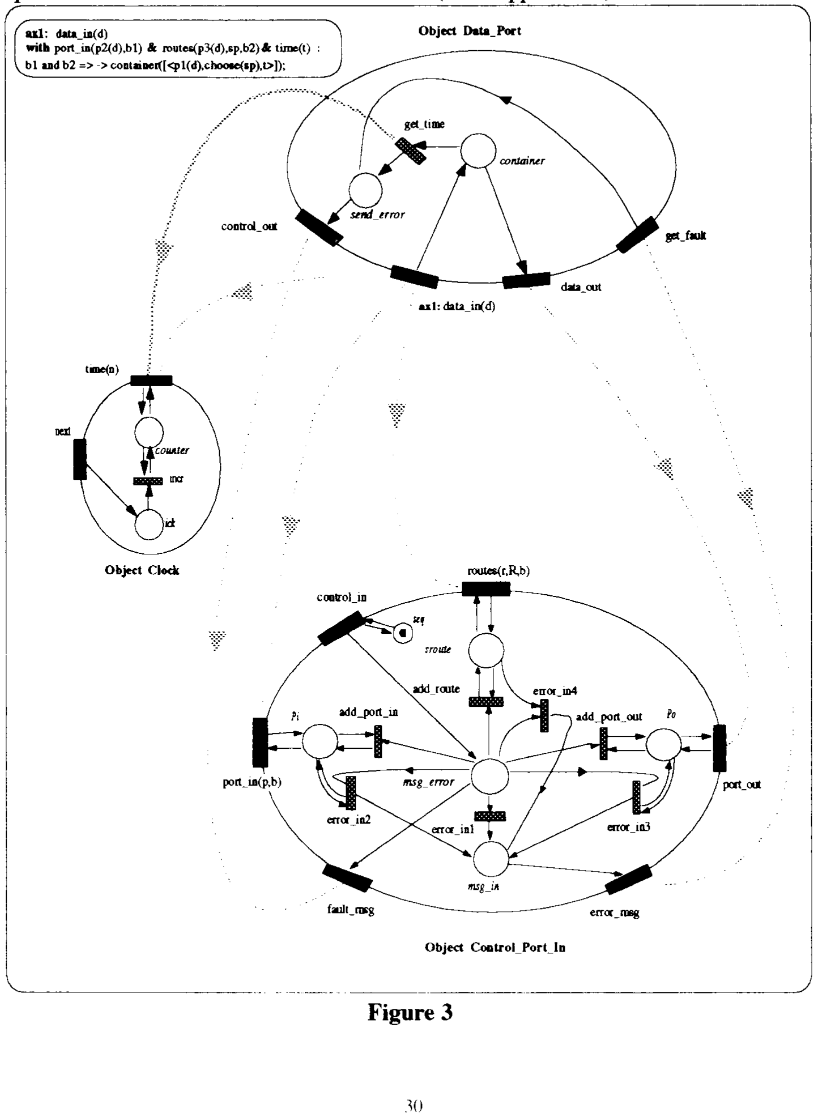


Figure 3

2.2.1. The Object Control_Port_In: The object Control_Port_In describes the behavior associated to the reception of input control messages as described in the informal presentation of the T-Node. The interface of this object is made of a set of methods (black transitions in figure 3) which specify the control part of the T-Node. For example, the method 'port-in' will be called by the method 'data-in' to ensure that the port receiving a data message has been enabled. The internal part of this object react in function of external calls of its methods to compute a new state of the control part of the T-Node.

2.2.2. The Object Data_Port: The object Data-Port models the transmission part of the T-Node. This object has two places, 'container' and 'send_error'. The place 'container' contains the messages in transit through the T-Node and the place 'send_error' contains the faulty messages available for the customer.

We have defined the following events :

1) The ***method*** 'data_in(d)' models the arriving of a data message 'd' with d=<msg,p,r> on the port p. The message 'msg' included in d is put in the place 'container' with a chosen route for the message. This event calls some methods of the objects 'Control_Port_In' and 'Clock' to ensure that p and r are enabled and to import the time and the routes associated to r.
2) The routing of a message msg contained in a data_msg dt=<msg,r,t> to an output port n, is done with the ***method*** 'data_out (n,π1(dt))' (π1(dt) = msg). For this, the method 'data_out' is synchronized with the method 'port_out(n,b)' of the object 'Control_Port_In' to ensure that the output port n is enable.
3) The 'control_out' ***method*** can be called by a customer to get an error message. It extracts a 'Send_fault' message from the 'msg_error' place using the 'fault_msg' method of the object 'Object_Control_Port'.
4) The ***internal transition*** 'get_time' is used to select the messages waiting for transmission that are too old.
5) The 'get_fault' ***internal transition*** is synchronized with the method 'error_msg' of the object 'Object_Control_Port' to extract the faulty messages of the place 'msg_error' and put them in the place 'send_error'.

We give below, the syntactic definition of this object using the CO-OPN language.

```
Object : Data_Port;
Specs used : Data_msg, Msg,Port, Route,Routes, Bool, Store_Data,Fault;
Objects used : Clock, Control_Port_In;
Methods :
        data_in :       data_msg;                       (* input data *)
        data_out :      nat * msg;                      (* output data *)
        control_out :   fault;                          (* timeout transmission*)
Internal :
        get_time; get_fault;                            (* select too old data *)
Places :
        container :     store_data;
        send_error :    fault;
Var :
        b,b1,b2 : bool; d : data_msg;
        dt : store_data; t : time; sp : routes; f: fault;
Axioms :
data_in(d) with port_in(π2(d),b1) & routes(π3(d),sp,b2) & time(t) :
                b1 and b2 =>    -> container([<π1(d),choose(sp),t>]);
get_time with time(t):
                (π3(dt)+delta)>t => container([dt]) -> send_error([fault(dt)]);
data_out(n,π1(dt)) with port_out(n,b) :
                b and n=π2(dt) => container([dt]) -> ;
control_out(f):
                send_error([f]) ->;
get_fault with error_message(f):
                -> send_error([f]);
Init_marking :
End ;
```

The syntax of the axioms given for the events (methods or internal transitions) is:

```
transition_name_with_parameters
with    synchronization_expression:
        boolean_condition
        =>input_relation->output_relation;
```

The simultaneous synchronization operator is symbolized by the character '&'. The multiset of terms 'a' and 'b' is represented by [a,b]. Methods are specific transitions which can be called by other events using synchronization expressions. The internal transitions cannot be called by other transitions; they represent the externally invisible modification of the state of the object. The methods are parameterized allowing to transmit values between objects.

2.2.3. The Object Clock: This object defines a current time for the T-Node, the method 'next' models a time base that increments the current time stored in the place 'counter'. An intermediate place 'counter' is used to manage the concurrent calls of the two methods 'time' and 'next'. It is the only way to achieve this because the 'counter' place is a single resource.

2.3. Refinement of the specification

In this section we give a summary of how to refine our specification on the object part and on the algebraic abstract data type part, using our results of component replacement (see [4], [5]). To do that, we must prove the "bisimulation equivalence" [18] between the old object and the new one, and the observational equivalence [21] between the old algebraic specification and the new one. For example, we could do a refinement of the specification to solve the problem of fairness of messages chosen into the object 'data_port' by using a fifo structure instead of the multi-set structure modeling the places. We recall here two results which give a way for refining this CO-OPN specification. For more explanations see [4].

Theorem (object replacement):

Let $SPEC_O$ a CO-OPN specification and $SPEC_{O'_j}$ a new specification of the sub-object o_j of O, then for a given algebra A, if $SPEC_{O_j}$ is bisimilar to $SPEC_{O'_j}$ then $SPEC_O$ is bisimilar to the specification $SPEC_{O'}$ in which o_j is replaced by o'_j.

Theorem (algebra replacement):

Let A1 and A2 two models of the algebraic specification, if A1 is observably equivalent to A2 and the CO-OPN specification is a free flow then $SPEC_O$ with the algebra A1 is bisimilar to $SPEC_O$ with the algebra A2.

2.4. Specification of a distributed set of T-Nodes

In our case, we want to specify a system constituted of a set of T-Nodes. Each T-Node is specified as described in the previous paragraphs, so we use the parametric objects construction of the CO-OPN language (not described in this paper) to specify as many instances of T-Node as needed. Thus the physical links between T-Nodes can be specified in an abstract way by defining a CO-OPN object with a method synchronizing the output message of one T-Node with the input message of an other T-Node. Concerning the concurrency expressed by this definition of the physical links: all physical links are concurrent (w.r.t the semantics of CO-OPN) and communications between the T-Nodes are synchronous. A next step in the specification will be to refine it by defining communication protocols between T-Nodes. To do this one can refer to the examples given for open distributed systems in [6].

To specify a distributed system we have to distinguish the computing part from the communication one. We first specify for each computing part its local behavior using a set of CO-OPN objects. Then the communication layer has to be specified representing the communication needed. It can be done using CO-OPN objects synchronizing the CO-OPN objects specifying the computing parts. This abstract system can be refined to describe several topologies and communication protocols [20]. These refinements will concern both the net and the algebraic parts.

3. Conclusion

In this paper we have presented an example of formal specification of distributed systems using the formalism CO-OPN. In this case study, CO-OPN features such as: algebraic specifications, modularity, concurrency and object synchronization have been presented. Through this work we show that the specification of distributed systems implies that the specification language has to be modular and concurrent. It also must formally treat the data structures and to include some formal refinement techniques. Parametric objects and formal semantics of CO-OPN have been quoted.

The works in progress or planned on CO-OPN are of two types:

- theoretical on the model and on the techniques that can be used for proving the equivalence of objects and the properties of the modeled systems.
- practical on the realization of tools for supporting the specification of systems (graphical editor, compiler, and prototyper on distributed systems and multiprocessor MIMD machine).

Acknowledgments: We are especially grateful to Elisabeth Pelz and Heather Sacco for their helpful suggestions on this paper.

BIBLIOGRAPHY

[1] E.Battiston, F.De Cindio, G.Mauri
"OBJSA a class of High level nets having objects as domains", LNCS n°340, pp 20-43

[2] D. Bertsekas, R. Gallager
"Data networks", Prentice Hall, 1987

[3] D. Buchs, N. Guelfi
"A semantic description of actor languages by structured algebraic Petri nets", research report n°639 LRI, UPS Orsay, 1991.

[4] D. Buchs, N. Guelfi
"CO-OPN : A Concurrent Object Oriented Petri Net Approach", 12th International Conference on theory and application of Petri Nets, Aahrus, 1991.

[5] D. Buchs, N. Guelfi
"System Specification Using CO-OPN", to appear as research report LRI, UPS Orsay, 1991.

[6] D. Buchs, N. Guelfi
"Open distributed programming using the object oriented specification formalism CO-OPN", research report n°700, LRI, UPS Orsay, 1991.

[7] G.Bernot, M.C. Gaudel, B. Marre
"Software Testing based on formal specifications : a theory and a tool", To appear in Software Engineering Journal, 1991.

[8] M. Bidoit
"The stratified loose approach : a generalization of initial and loose semantics", Recent Trends in Data Type Specification, selected papers of the 5th work-shop on specification of abstract data types, LNCS 322, 1987.

[9] M. Bidoit
"PLUSS, un language pour le developpement de spécification algébrique modulaires" Thèse de doctorat d'Etat, LRI UPS Orsay.,1989.

[10] D. Buchs
"EM2/specif : a method for specifying interactive application", Workshop on software engineering and its application, pp 540-556, Toulouse. 1988.

[11] H.Ehrig, B.Mahr
"fundamentals of algebraic specification 1: equations and initial semantics,Springer Verlag 1985."

[12] J.Fiadero, T.Maibaum
"Describing, structuring and implementing objects", in Foundations of objects oriented languages, LNCS 489, (1990) .

[13] B.Krämer
"Segras : a formal language combining Petri nets and abstract data types for specifying distributed systems" in proceedings of the 9th annual international conference on software engineering, pp 116-125, Monterey, 1987.

[14] L.Logrippo and al.
"An interpreter for Lotos : A specification language for distributed systems", in software practice and experience, vol. 18 n°4, 1988.

[15] A. Mauboussin, H. Perdrix, M.Bidoit, M.C. Gaudel, J. Hagelstein
"From an ERAE requirements specification to a PLUSS algebraic specification : a case study", in Algebraic methods II: Theory, Tools and Applications, J.A Bergstra, L.M.G. Feijs Eds, LNCS 490, 1991

[16] S.Mauw, F.Wiedijk
"Specification of the Transit Node in PSF_d", in Algebraic methods II: Theory, Tools and Applications, J.A Bergstra, L.M.G. Feijs Eds, LNCS 490, 1991

[17] R. Milner
"Communication and Concurrency". Prentice Hall, 1989.

[18] L. Pomello
"Some equivalence Notions for Concurrent Systems: An Overview".
In: Advances in Petri nets 1985 Springer Verlag, LNCS 222 pp. 381-400.

[19] W.Reisig
"Petri nets and Algebraic specifications", TCS.80, 1991

[20] D.Sidhu, A.Chung, T.P Blumer
"Experience with formal methods in protocol development, Computer communication review, vol. 21, n°2, 1991, ACM.

[21] D.Sannella, A. Tarlecki
"On observational equivalence and algebraic specification", TAPSOFT 85, LNCS 185 pp308-322., 1985.

[22] J.Vautherin
"Parallel systems specification with colored Petri nets and algebraic specification" LNCS 266, ICPN, 1987