Object-Oriented Modeling and Analysis Capabilities

CO-OPN to AMInets Task Force
(Didier Buchs[1], Matieu Buffo[1] and Fabrice Kordon[2], in alphabetical order)

[1] Software Engineering Laboratory, Swiss Federal Institute for Technology,
1015 Lausanne, Switzerland
{Didier.Buchs,Mathieu.Buffo}@epfl.ch
[2]SRC, LIP6, UMPC - CNRS
4 Place Jussieu, 75252 Paris Cedex 051015
Fabrice.Kordon@lip6.fr

Abstract. Usual object-oriented modeling notations have either high expressivity, or a high proof-potential. As a consequence, developers have to choose between an easy modeling or good analysis capability. This paper proposes to bridge the object-oriented modeling language CO-OPN with the Petri-net based formalism AMI-nets. Hence, user can manipulate easy-to-use, expressive CO-OPN models, while keeping high proof-potential thanks to AMI nets.

1 Introduction

Development and maintenance of industrial applications becomes more and more difficult. Systems' complexity increases [19], technologies evolve, requirements has to take care of «social factors» [12] and products' time to market reduces. Instead of «software crisis», people are now speaking of «software chronic crisis» [11]. Estimation costs of such a crisis have been estimated to $ 100 billion in 1996 [27].

One problem in implementing systems is its evaluation. Post implementation testing is not reliable because it is impossible to cover all possible executions of complex systems. Software modeling through Object Oriented approaches such as OMT or UML appears to be a solution to this problem. However, these description languages essentially focus on the description of a general solution. Then, the system still has to be analyzed.

Analysis could be supported by formal methods that are known to bring safe evaluation techniques, due to their mathematical foundation [20, 21]. However, these mathematical foundations also bring problems like:

- Formal notation is alien to most practicing programmers, who have little training in higher mathematics,
- Even if mathematical notations are very general, they require discrete types of methods for discrete application domains. Thus, the same notation can be used in many ways; this implies discrete evaluation and proof methods,

- It is difficult to integrate formal methods into industrial software processes because the notation used contain many mechanisms that are difficult to use,
- Many of the most popular formal methods do not scale up to practical-size problems.

Thus, it appears that formal methods cannot be directly used to validate large industrial-like systems. They have to be manipulated via high level representations.

This paper proposes to bridge the object-oriented Petri nets based specification language CO-OPN with the Petri net model of the CPN-AMI environment (AMI-Nets) to provide an approach dedicated to the analysis of concurrent systems. Hence, users can manipulate high-level representations in CO-OPN, while discrete evaluations and proof methods are likely to be performed on AMI-Nets using CPN-AMI. Section 2 introduces the various models and notations used in this work. Section 3 presents the current stage of the bridge between CO-OPN and AMI-Nets. Then, sections 4 and 5 illustrate our approach by means of two discrete examples focusing on two different aspects of concurrent systems. Finally, section 6 concludes the paper.

2 Software Modeling

It was mentioned in the introduction that one problem in implementing software reside in their evaluation. One way - promoted by Software Engineering - to meet this challenge is to develop software models before the definitive product. These software models can be used to validate user requirements, to derive early prototypes, and finally to evaluate the various possible solutions. Actually, analyzing software models is one of the key task of software development process.

Software models usually include descriptions of the main features of the proposed software, defined in a suitable modeling language or notation. Actually, we feel that good modeling languages have an ability to abstract - at least intuitively, for the developers - marginal but rather complex implementation details.

Among the various modeling notations used yet, we propose to have a look at three typical notations, namely UML, Petri nets and algebraic abstract data types.

- UML [25] (Unified Modeling Language), proposed by OMG (Object Management Group), is becoming the current leader among the modeling languages for object-oriented systems. UML models are composed of various graphs, defining classes, objects and messages composing the models.
- Petri nets are a convenient way to describe software models based on the intuitive notion of state machines. In short, a Petri net is a notation relying on places, containing the resources of the system, and transitions, reflecting the interactions among the resources. With regards to automata, Petri nets are much more easy to use, much more intuitive, because the coordination between the various parts of the system are explicitly represented.
- Algebraic abstract data types are a formal modeling notation based on a sound mathematical background. In short, such models, called algebras, are induced by a set of axioms, formalizing the desired properties.

2.1 Expressiveness Power versus Analysis Capabilities

Let us consider discrete approaches and notations family. We could class them using three criteria:

- *structuration potential*: this criterion requires the associated notation to provide high level structuration capabilities, like an object model (classes, inheritance etc.), a module approach, and so on;
- *verification potential*: this criterion requires strong foundation to provide execution of a specification, model checking and even structural analysis of a system model;
- *ease to verify*: this criterion requires that the verification potential is likely to be performed using simple techniques and tools

Fig. 1. shows how some usual notations can be classified according these three criteria. OMT or UML provide good design capabilities but are very poor in terms of verification that is mostly cross review by humans or simulation (usually ease to produce). On the other end, notations such as Petri nets or algebraic specifications have strong formal verification capabilities with average ease-to-verify; however these notations lack high-level structuring capabilities and it is difficult to handle complex specification such as the one of industrial applications. Some extensions to formal methods are often investigated like CO-OPN and other Object-Oriented Petri Nets [14, 18, 2]. The main problem of such formalisms is to catch up with formal properties: small modifications induce theoretical problems that are still to be solved. Thus, even if these formalisms have a good structuration potential and a good verification potential, they have a rather low ease to verify evaluation.

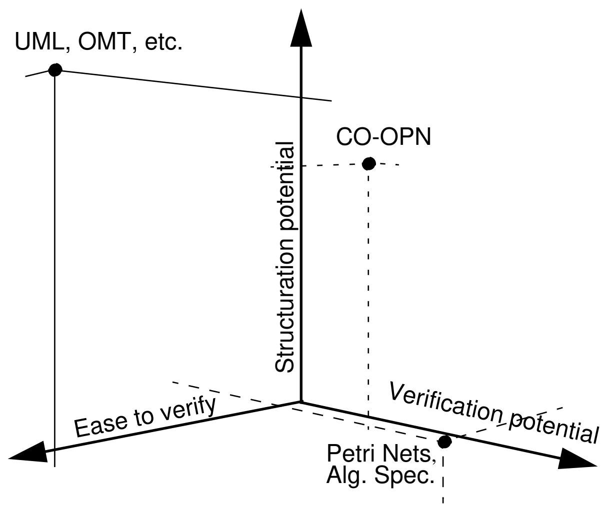

Fig. 1. A classification of specification notations.

2.2 Association of UML, CO-OPN and AMI-Nets

Our approach is based on several refinement steps. Each one is dedicated to a main goal and relies on an appropriate formalism. Fig. 2. illustrates our process. First, modeling is performed using a suitable modeling language like UML [25]. This is done according to a given method like FUSION [8]. Then, the second step resides in the *derivation* into a high level formal notation for object oriented modeling: CO-OPN. Finally, the last steps consist in *transforming* CO-OPN into AMI-Net: a formal specification suitable for supporting the proof process. AMI-Net is a Petri net dialect having the expression power of Well-formed nets [6].

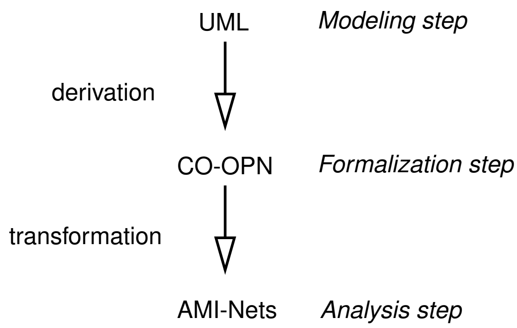

Fig. 2. Chaining formalisms over the proposed approach.

Fig. 2. provides a more synthetic view the proposed procedure. Our aim is to safely derive distributed programs from UML models. Both formalization and analyses steps allows the exploitation of formal methods. This enables the validation of the UML model and verification its formalization (using Petri nets). It also allows us to take into consideration formal properties to optimize the resulting program.

Fig. 2. also shows the path from a specification level to another one. One CO-OPN specification is derived from an UML specification. On the contrary, several Petri Net specifications are derived from CO-OPN. Each one represents one particular aspect of the CO-OPN model and is dedicated to the verification of a given property. Implementation of CO-OPN modules into programs takes benefits from this analysis.

The first step now is to briefly describe the basics of CO-OPN and AMI-Nets as well as the features of CPN-AMI. Then we have to present the translation strategy between both formalisms we use. Finally - and archetypally - we illustrate our approach by means of two paradigmatic case studies. These case studies are very simple but cover a substantial part of the main concepts of CO-OPN.

The goal of the first case study is the modeling of a simple communication protocol. It allows the analysis of the various components involved in the system, the analysis of their instantiation, their composition, their synchronization. The second case study deals with the modeling of an accumulator component. It focus on the deployment of complex object synchronizations (in particular, sequential recursive method calls) and the management of algebraic data types.

2.3 CO-OPN

CO-OPN is an object-oriented modeling language, based on Algebraic Data Types (ADT), Petri nets, and IWIM coordination models [5]. Hence, CO-OPN concrete specifications are collections of ADT, class and coordination *modules* [3, 4]. Syntactically, each module has the same overall structure; it includes an *interface section* defining all elements accessible from the outside, and a *body section* including the local aspects private to the module. Moreover, class and context modules have convenient graphical representations, showing their underlying Petri net model. Low-level mechanisms and other features dealing specifically with object-orientation, such as genericity, sub-classing and sub-typing are out of the scope of this paper, can be found in [3].

ADT Modules. CO-OPN ADT modules define data types by means of algebraic specifications. Each module describes one or more sorts (i.e. names of data types), along with generators and operations on these sorts. The exact definition of the operations is given in the body of the module, by means of equationnal axioms. For instance, Figure 3 describes a (very simple) ADT defining one sort (the booleans) and one operation on this sort (the negation).

```
ADT SimpleBooleans;
Interface
  Sort booleans;
  Generators true, false : ->boolean;
  Operation not_ : boolean->boolean;
Body
  Axioms
    not (true) = false;
    not (false) = true;
End SimpleBooleans;
```

Fig. 3. ADT SimpleBooleans

Class Modules. CO-OPN classes are described by means of modular algebraic Petri nets with particular, parameterised, external transitions, the *methods* of the class. The behaviour of transitions are defined by so-called *behavioural axioms*, corresponding to the axioms in ADT. A method call is achieved by synchronizing external transitions, according to the fusion of transitions technique.

Below is the code and the associated Petri net graphics of a class modeling an unusual storage system; it stores boolean values, but delivers the negated ones. The interface defines two methods, for the storage and the retrieving of values. The body is actually a textual representation of the associated Petri net. Free variables may be defined and used in the behavioural axioms.

```
Class StrangeStorageSystem;
Interface
  Use SimpleBooleans;
  Methods put _ ,  get _ : boolean;
Body
  Place container _ : boolean;
  Axioms
      put b :: -> container b;
      get b :: container not(b) -> ;
  Where b : boolean;
End StrangeStorageSystem;
```

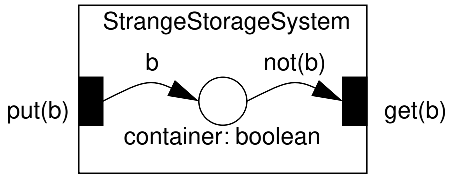

Fig. 4. Class StrangeStorageSystem

Coordination Modules. A third kind of modules is present in CO-OPN, the *context* modules [5], which share the same overall structure with ADT and class modules. Basically, context modules allow the modeling of distributed systems, by means of suitable coordination mechanisms, more complex than the fusion of transitions seen above. As context modules are clearly specific to the coordination theory, they are not illustrated here.

2.4 AMI-Nets

AMI-Nets are a Petri Net dialect having an expression strength equivalent to the one of Well-formed nets [6]. They include, besides the graphical features of a Place/Transition Petri net (places, transitions and arcs) textual information like:

- place and transition domains, and transition guards,
- an enriched syntax for arc labels and place markings,

The behavior of an AMI net is controlled by the same set of rules used for general colored nets:

- A domain is associated with each place and transition of the model. Elements of these domains are called colors.
- When firing, a transition is binded by an element of its domain.
- Each token in a place is colored by an element of the place domain (several tokens may have the same color). The marking of a place is thus a multiset of colors - a set in which an element may occur several times.
- For a binded transition to be enabled, each input place of the transition must contain a sufficient (possibly null) number of tokens for every color of the place domain. These tokens will be taken from the place when the transition fires. Similarly, the firing will produce colored tokens in the output places of the transition.

Like in Ordinary Petri nets, the label attached to the arcs determines the number of tokens to be taken or produced. However, this label is now a color function that associates a multiset of colors of the place domain with each binding of the transition.

- Independently from the evaluation of the color functions, a transition may not be enabled if its binding does not satisfy some predicate. This predicate is called the guard of the transition.

2.5 The CPN-AMI environment

CPN-AMI [22] is a collection of tools federated in FrameKit [16], a generic CASE environment offering both integration capabilities and an enhanced development environment. As all CASE environments generated from FrameKit, CP-AMI offers a user-friendly access to Petri net services through a unique user interface: Macao [23].

This architecture is one of the strongest points in CPN-AMI. It enables an enrichment process taking benefits of other developments to propose a unified Petri net based environment. Enrichment of the successive versions of CPN-AMI was done at a relatively low development cost.

The current version of CPN-AMI offers numerous services such as:

- Modeling tools:
  - *Syntactic verifier*: checks the AMI-Net syntax and transform the Petri net into an internal representation.
  - *Modular Petri net assembling*: this tool is built to help designer to assemble modules communicating either by means of places or by means of transitions. The users select a group of objets and then, merge them to one equivalent object if it is possible (for example, color domains are the same for places).
  - *Pretty Petri Nets*: this service aims to rearrange "spaghetti" Petri nets. This service has been made to be exploited by other Petri net services (like CPN-Unfolder, Prefix or reachability graph display). However, it can be directly invoked by a user. This service relies on DOT [17].
  - *Suppression of 0-bounded places and non-firable transitions*: uses the bound of place service to suppress 0-bounded places and transitions with those places as precondition. Mainly used with structural analysis in order to limit the study to the useful part of the net.
- Simulation and debugging:
  - *Colored Petri net simulator*: in this tool, we have attempted to keep, as more as possible, the analogy with programming language debuggers. To achieve this goal, the user may use different execution modes, break point possibilities, data extractions during the execution and external treatments associated to transition. Standard debugging functions are also available like intermediate state management (including load and save operations) and configuration management (a configuration is a set of simulation parameters: scripts definitions, observation

net, intermediate state).

- Structural analysis:
  - *Boolean formula on reachability graph*: this tool computes a set of markings containing the reachability set. In this set, places are just considered as "marked" or "unmarked". The results are displayed as properties over the net. This service uses BDDs [24] to compute the marking set.
  - *Bounds of places:* this tool computes lower and upper bounds. The calculus is based on the state equation and uses linear programming techniques. As a consequence, the computed bounds (higher and lower) may not be the best ones, but this tool may be useful to quickly highlight some major problems in the model. For colored models, this tool can be accessed via P/T unfolding. This service is based on lp_solve (ftp://ftp.es.ele.tue.nl/pub/lp_solve).
  - *Place invariants:* computation of P-SemiFlows using a service from GreatSPN [7].
  - *Colored place invariants:* this tool computes invariants using a adaptaed version of the general algorithm [9]. It is one of the very few implemented ones.
  - *Transition invariants*: computation of T-SemiFlows using a service from GreatSPN.
  - *Siphon and deadlocks*: they can be computed using a service from GreatSPN or using a BDD based implementation.
  - *Liveness computation*: computes if the net is live (from any reachable state and for any transition it is possible to reach a state from which the transition is firable).
  - *Linear properties characterization*: the aim of this tool is to compute a linear characterization of the reachability set. When the resulting linear constraints system exactly describes the reachability set, a message warns the user.
  - *Colored Petri net unfolding*: transforms a colored Petri net into a Place/Transition Petri net. The resulting net is a new model that can be displayed and analyzed. An option allows to suppress 0-bounded places and non-firable transitions. This option uses a heuristic to compute those places (it is not based, like the "suppression of 0-bounded places and non-firable transitions" service, on linear programming). Another option allows to compute a pretty layout of the resulting net.
  - *McMillan unfolding*: this service computes an unfolding for a safe net (safety is not verified by the tool). This software has been developed by S. Römer (from Technische Universität München) and implements the algorithm defined by J. Esparza, S. Römer & W. Vogler in [10]. This tool is also part of PEP [13].
- Model checking:
  - *Generation of the reachability graph, CTL and LTL queries evaluation*: this tool is based on PROD [28].
  - *Generation of the symbolic reachability graph*: This service is based on a service in GreatSPN. A Symbolic Reachability Graph (SRG) is a highly condensed representation of the reachability graph built automatically from a specification of system in terms of Well-formed net. The building of such graph profits from the presence of object symmetries to aggregate either states or actions within

symbolic representatives (equivalence classes). The equivalence relation between states is based on structural symmetries that are directly read off from the types of objects defined in the system specification. By defining convenient types of actions for these types of objects, it can be ensured that states that are equivalent let the future behavior of the system unchanged

3 Translation Strategies and Rules

One of the key aspect of our work is to be able to translate a CO-OPN specification into a model more suitable for the analysis of properties. Hence, it is a way to extract properties of CO-OPN specification without paying the price of analyzing the original CO-OPN model. Then, properties computed on the «analysis model» can be interpreted in the CO-OPN model. Petri nets are suitable as the analysis model. To support the analysis of specification, we have chosen AMI-Net. Thus, the Petri net dialect we have selected is AMI-Net.

Fig. 5. shows a typical translation scheme. The translation process has to cope with the two aspects of CO-OPN (Petri nets and Algebraic data types, respectively noted PN and ADT on the Figure). Please note that, due to the expressiveness of CO-OPN, we consider several translation schemes. Each one is dedicated to the verification of particular properties.

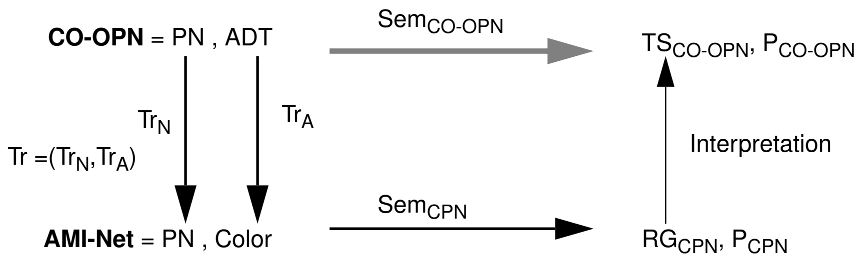

Fig. 5. The translation scheme.

The gray arrow in Fig. 5. corresponds to the difficult way of proving a CO-OPN specification. Hence, we exploit the correspondence between CO-OPN and AMI-Nets. For some properties $P_{valid} \subseteq P_{CO-OPN}$ we aim to prove, we can find a translation Tr such that the interpretation of AMI-Net properties is included in P_{valid} (i.e. $Interpretation(P_{CPN}) \subseteq P_{valid}$), where P_{CPN} is such that $Sem_{CPN}(Tr(CO-OPN)) = (RG_{CPN}, P_{CPN})$.

Therefore, our goal is to find suitable transformations Tr between CO-OPN and AMI-Nets. We decompose this transformation in two discrete steps. The first one is devoted to the translation of CO-OPN systems into standard (i.e. un-synchronized) algebraic Petri nets (APN) [29, 26], while the second one is dedicated to the translation of algebras into colors. This situation is suggested in Fig. 6.

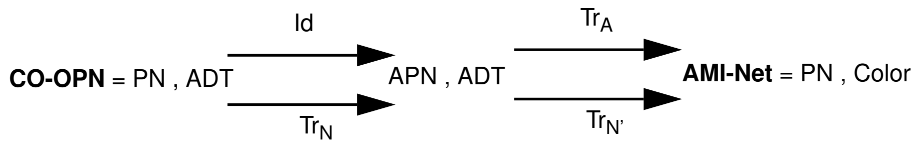

Fig. 6. Detail of the CO-OPN to AMI-Nets translation.

The following sections present the key problems of the transformation process from CO-OPN to AMI-Nets, as well as its limitations. We now provide translation rules for the synchronization operations, as well as those dealing with axioms (recursive definitions, classes and algebraic values, etc.).

The detail of the translation between CO-OPN and algebraic net is given in Fig. 7. The translation is divided into two parts, the first is the construction of the computation of the transaction by sub-nets. Each sub-net being connected to each other by fusion of transition, the second step is the interpretation of the fusion operators in order to produce the resulting algebraic net..

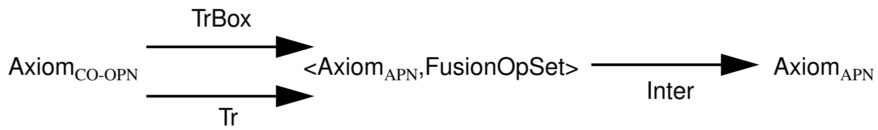

Fig. 7. Detail of the CO-OPN to Algebraic net translation Tr_N.

First of all, we must provide a simplified but formal description of the CO-OPN notations, needed for the description of the translation rules. Given an algebraic specification ADT, given a set of places P, a set of transitions T and a set of behavioural axioms $Axioms_{COOPN}$ denoted by t with $sync : pre \rightarrow post$ where $t \in T$ is a transition, $sync \in Sync$ is a synchronization built over transitions by the simultaneous, sequential and alternative operators:

$$t \in T \Rightarrow sync \in Sync \qquad (1)$$

$$s, s' \in Sync \Rightarrow s\ //\ s' \in Sync \qquad (2)$$

$$s, s' \in Sync \Rightarrow s + s' \in Sync \qquad (3)$$

$$s, s' \in Sync \Rightarrow s \ldots s' \in Sync \qquad (4)$$

If pre and $post$ are markings over the set of places P, a CO-OPN specification is:

$$Spec_{COOPN} = \langle P, T, Axioms_{COOPN}, ADT \rangle \qquad (5)$$

Given axioms $Axiom_{APN}$ of form $t : pre \rightarrow post$, we define an algebraic net as:

$$Spec_{APN} = \langle P, T, Axioms_{APN}, ADT \rangle \tag{6}$$

Before going deeper into details about transformations $TrBox$, Tr and $Inter$, we briefly introduces now the concepts of transactions - how they are seen in CO-OPN - and transition fusion.

3.1 Synchronizations as Transactions

CO-OPN synchronizations can be considered as nested transactions. Therefore, we first describe how a single synchronization can be translated into a transaction using standard Petri net features. Fig. 8. shows an example of synchronization, where transitions t and $serv$ are fired simultaneously, as a transaction.

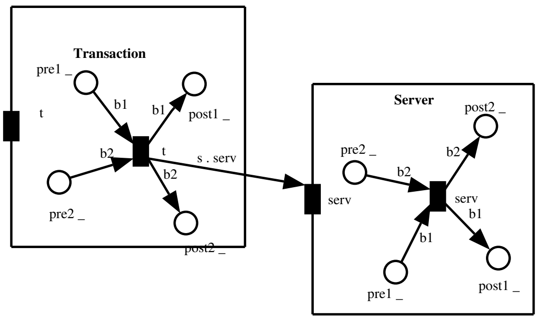

Fig. 8. Example of CO-OPN synchronization.

Fig. 9. shows the result of our translation procedure. we split transition t into two transitions: $Tstart$ and $Tend$, representing respectively the beginning and the end of the transaction. Both transitions are fired sequentially. A similar transformation is applied to transition *serv*. To respect the transaction concept, $Tstart$ is associated with $ServStart$ and $Tend$ is associated with $ServEnd$. Both associations are accomplished by applying the transition fusion principle [1].

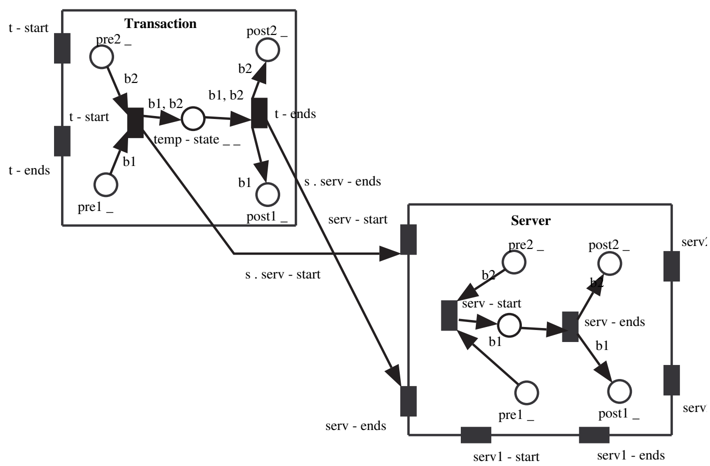

Fig. 9. Translated synchronization.

3.2 Transition Fusion

In order to collect the set of all the necessary fusion of transitions in T, we define the syntactic fusion operator, denoted by $FusionOpSet$, with the following profile:

$$name \# name_1 ... name_n \in T^{n+1} \tag{7}$$

The semantic of the fusion of transition consists in the union of all the sets representing pre- and post-conditions of transitions $name_i$ involved in the fusion operation. This obviously produce a new pre- and post-condition, associated to the new transition denoted by $name$.

We are now able to express the transition fusion that is needed when decomposing CO-OPN synchronizations using our syntactic operator. The interpretation of this syntactic operator on a set of axiom is given by the $Fusion$ operation, defined as follows:

$$Fusion : \wp(Axiom_{APN}) \times FusionOpSet \rightarrow \wp(Axiom_{APN}) \tag{8}$$

$$Fusion(Ax, name \# name_1 ... name_n) = Ax \cup \{name : pre \rightarrow post\} \tag{9}$$

where $pre = \bigcup_{0 < i < n+1} Pre(Ax, name\dot{i})$ and $post = \bigcup_{0 < i < n+1} Post(Ax, name\dot{i})$.

This operator is the basis of the translation of CO-OPN into algebraic nets. Nevertheless all problems are not solved yet, as the following remaining questions should be stated:

- How to manage algebraic terms? (variables must convey the values)
- How to deal with the multiplicity of the axioms? (it is mainly a combinatorial expansion of the fusion)

These questions will be illustrated through paradigmatic examples.

3.3 Translation of Synchronization Expressions: $TrBox$

We define $TrBox$ the translation operation acting on synchronization expressions:

$$TrBox : Sync \to Axiom_{APN} \times \wp(FusionOpSet) \times T \times T \tag{10}$$

This operation is defined with the rule $Trans$ dealing with simple coercion, and recursively by the rules Sim, Seq and Alt dealing with simultaneity, sequentiallity and non-determinism respectively. The translation produces axioms, operators of fusion, and transitions. Actually, the translated elements can be seen as a kind of box, with two transitions generically called $Tstart$ and $Tend$ (The last two elements in the operation profile) acting as connectors. The semantics of these boxes must be interpreted as follows: the synchronization is fire-able if there exist - in the semantics of the translated net - a fire-able sequence of transitions, the first element of which is $Tstart$ and the last element $Tend$.

The rule for a simple coercion t, called $Trans$ is defined as follow (we assume that transitions $Tstart$ and $Tend$ are new for each application of the rule):

$$\frac{}{TrBox(t) = \langle \emptyset, \emptyset, Tstart, Tend\rangle} \quad Trans \tag{11}$$

The rule for the simultaneity of synchronizations s and s', called Sim, is defined as follow (we assume that each application of the rules produces new items $Tstart$, $Tend$, $temp$, $Sseq_1$ and $Sseq_2$):

$$\frac{TrBox(s) = \langle Ax, Fus, Sstart, Send\rangle,\ TrBox(s') = \langle Ax', Fus', S'start, S'end\rangle}{TrBox(s\ //\ s') = \langle newAx, newFus, Tstart, Tend\rangle} \quad Sim \tag{12}$$

Where:

$$newAx = Ax \cup Ax' \cup \{Sseq_1 : \ \to temp\} \cup \{Sseq_2 : temp \to \ \}$$

and

$$newFus = Fus \cup Fus' \cup \{Tstart\ \#\ Sstart\ S'start\ Sseq_1\} \cup \{Tend\ \#\ Send\ S'end\ Sseq_2\}$$

The rule for the sequentiality of synchronizations s and s', called Seq, is defined as follow (we assume that ε is the generic name for un-named transitions, and that each application of the rules produces new items $Tstart$, $Tend$, $temp_1$, $temp\bar{2}$, $temp_3$, $Sseq_1$, $Sseq_2$, $S'seq_1$ and $S'seq_2$):

$$\frac{TrBox(s) = \langle Ax, Fus, Sstart, Send\rangle,\ TrBox(s') = \langle Ax', Fus', S'start, S'end\rangle}{TrBox(s \ldots s') = \langle newAx, newFus, Tstart, Tend\rangle} \quad Seq \tag{13}$$

Where:

$$newAx = Ax \cup Ax' \cup \{Sseq_1 : \ \to temp_1\} \cup \{Sseq_2 : temp_1 \to temp_2\}$$
$$\cup \{S'seq_2 : temp_2 \to temp_3\} \cup \{S'seq_2 : temp_3 \to \ \}$$

and

$$newFus = Fus \cup Fus' \cup \{Tstart \# Sstart\ Sseq_1\} \cup \{\varepsilon \# Sseq_2\ Send\}$$
$$\cup \{\varepsilon \# S'start\ S'seq_1\} \cup \{Tend \# S'end\ S'seq_2\}$$

The rule for the non-deterministic choice of synchronizations s and s', called Alt, is defined as follow (we assume that ε is the generic name for un-named transitions, and that each application of the rules produces new items $Tstart$ and $Tend$):

$$\frac{rBox(s) = \langle Ax, Fus, Sstart, Send\rangle, TrBox(s') = \langle Ax', Fus', S'start, S'end\rangle}{TrBox(s + s') = \langle Ax \cup Ax', newFus, Tstart, Tend\rangle} \quad Se \qquad (14)$$

Where:

$$newFus = Fus \cup Fus' \cup \{Tstart \# Sstart\} \cup \{Tstart \# S'start\}$$
$$\cup \{Tend \# Send\} \cup \{Tend \# S'end\}$$

It must be noted that recursive definitions are not covered by $TrBox$. Hence, we propose to handle recursivity through net expansion; each level of recursion implies the adding of new transitions in the translated net. For instance, consider the example depicted in Fig. 10. showing iterators on naturals. The simple net is translated and expanded for two levels of recursion.

```
        t: Naturals;
        t(succ(n)) with t(n): pren -> postn;
        t(0) : pre0 -> post0;
```
Produces, for two iterations:
```
        t(succ(n)): n= succ(m) & m = 0 =>
              pren/1,pren/2,pre0 -> postn/1,postn/2,post0;
        t(succ(n)): n= 0 => pren/1,pre0 -> postn/1,post0;
        t(0) : pre0 -> post0;
```
Which can be reduced by replacing equal by equal:
```
        t(succ(succ(0))):
              pren/1,pren/2,pre0 -> postn/1,postn/2,post0;
        t(succ(0)): pren/1,pre0 -> postn/1,post0;
        t(0) : pre0 -> post0;
```
Produces for 2 iterations and transition splitting:
```
        t-start(succ(n)) : n= succ(m) & m = 0 =>
              pren/1,pren/2,pre0 -> temp1 n, temp2 m, temp0;
        t-ends(succ(n)):
              temp1 n ,temp2 m, temp0-> postn/1,postn/2,post0;
        t-start(succ(n)) : n = 0 =>
              pren/1, pre0 -> temp1 n, temp0;
        t-ends(succ(n)): temp1 n, temp0 -> postn/1, post0;
        t-start(0) : pre0 -> temp0;
        t-ends(0) : temp0 -> post0;
```
Which can be reduced by replacing equal by equal:
```
        t-start(succ(succ(0))) :
              pren/1,pren/2,pre0 ->
              temp1 succ(0), temp2 0, temp0;
        t-ends(succ(succ(0))): temp1 succ(0) ,temp2 0, temp0 ->
              postn/1,postn/2,post0;
        t-start(succ(0)) : pren/1, pre0 -> temp1 0, temp0;
        t-ends(succ(0)): temp1 0, temp0 -> postn/1, post0;
        t-start(0) : pre0 -> temp0;
        t-ends(0) : temp0 -> post0;
```

Fig. 10. Iterator on Naturals

3.4 Translation of Synchronizations: Tr

CO-OPN introduces synchronization between transitions by means of the abstraction operator called $With$. This operator links an event (in other words, a local transition) and a synchronization expression, the translation of which is given above.

We define Tr the translation operation acting on axioms, including the synchronizations:

$$Tr : Axiom_{COOPN} \rightarrow Axiom_{APN} \times \wp(FusionOpSet) \tag{15}$$

This operation is defined with the rule $With$ dealing with axioms including synchronization, and with the rule $WithNoSync$, taking care of un-synchronized axioms.

The rule $With$ is defined by the following rule (we assume that each application of the rules produces new items $temp$):

$$\frac{Tr(Ax) = \langle Ax_1, Fus_1\rangle,\ TrBox(s) = \langle Ax_2, Fus_2, Sstart, Send\rangle}{Tr(Ax \cup \{t\ withs : cond \Rightarrow pre \rightarrow post\}) = \langle newAx, newFus\rangle}\quad With \tag{16}$$

Where:

$$newAx\ =\ Ax_1 \cup Ax_2 \cup \{Tpre : cond \Rightarrow pre \rightarrow temp\} \cup \{Tpre : temp \rightarrow post\}$$

and

$$newFus\ =\ Fus_1 \cup Fus_2 \cup \{Tstart\ \#\ Tpre\ Sstart\} \cup \{Tend\ \#\ Tpost\ Send\}$$

The rule $WithNoSync$ is defined by the following rule (we assume that each application of the rules produces new items $temp$):

$$\frac{Tr(Ax) = \langle Ax_1, Fus_1\rangle}{Tr(Ax \cup \{t : cond \Rightarrow pre \rightarrow post\}) = \langle newAx, newFus\rangle}\quad WithNoSync \tag{17}$$

Where:

$$newAx\ =\ Ax_1 \cup \{Tpre : cond \Rightarrow pre \rightarrow temp\} \cup \{Tpre : temp \rightarrow post\}$$

and

$$newFus\ =\ Fus_1 \cup \{Tstart\ \#\ Tpre\ Sstart\} \cup \{Tend\ \#\ Tpost\ Send\}$$

3.5 Interpretation of the fusion: $Inter$

The translation function Tr does not ensure the computation of all axioms. It is necessary to perform an interpretation of the transition fusion, as collected in the second component of the result of Tr. This is done by the interpretation operation $Inter$ which is based on the interpretation of the operator $Fusion$ seen above, acting on axioms. Formally, we define the interpretation operation:

$$Inter : Axiom_{APN} \times \wp(FusionOpSet) \rightarrow Axiom_{APN} \tag{18}$$

$$Inter(\langle Ax, Fus\rangle) = \begin{cases} Inter(\langle Ax \cup \{f\}, Fus - \{f\}\rangle) & \text{if } \exists f \in Fus,\ f = \{t\ \#\ t_1\ \ldots\ t_n\} \\ Ax & \text{else} \end{cases}$$

3.6 Recursive Definitions

The interpretation operation $Inter$, as seen above, do not cover recursive definitions. We propose, for now, to manage recursivity by admitting a bound to the execution of the recursive synchronizations, and by adopting ad-hoc techniques to avoid conflicts between the partially computed data resulting from the various level of recursion.

Hence, we should obtain an approximation of the original semantics, with a static limit in the re-application of recursive synchronizations. For a set of axioms Ax, we denote by Ax_{PART} the new set of axioms reflecting this process of recursion reduction. Many work must still be done in this area of investigations; we just can cite now the following conjecture:

Conjecture of Buchs: Ax has no cycles $\Rightarrow Semantics(Ax_{PART}) = Semantics(Ax)$

3.7 Templates

Class information are modelled by cartesian products of object references and values. Methods include an additional parameter representing the object on which the method is applied.

3.8 Taking into account Algebraic value in the Translation

The rule presented before must include algebraic value management, this is necessary for keeping the same value for the variable in the *t-start* and *t-end* variables. Intermediate place must be a cartesian product of the values of all the variables used in both *t-start* and *t-end* axioms.

3.9 Unfolding of Behavioural Axioms

In the axioms it is possible to unfold the conditions and parameter passing according to the algebraic definitions. This process splits behavioural axioms in various cases corresponding to the case of the algebraic definitions of the operators. This unfolding is generally infinite; bounds must be fixed depending on the interest of this decomposition. Unfolding can also be applied on selected operators depending on the goal of this unfolding.

4 Case study 1: Communication Protocol

The first case study deals with a communication protocol problem and illustrates the efficiency of both CO-OPN and Petri nets to extract accurate information suitable for a final implementation.

We would like to design a safe channel based on a single cable line. The usual problem with a unique cable is that electric signals coming from various origins may provoke collisions (message is lost). To ensure a safe communication on the channel, we propose the architecture of Fig. 11.

The channel relates two *interlocutors* that communicate together. It is composed of a *control cable* and a *controller* that manages shared access to the channel *main* *cable* (128 bits width). The controller is connected two each interlocutor with a discrete control cable (3 bits width). There is one control cable per interlocutor. Interlocutors cannot send a signal at the same time : they must ask first the line to the controller that accepts or refuse (according to an implemented strategy).

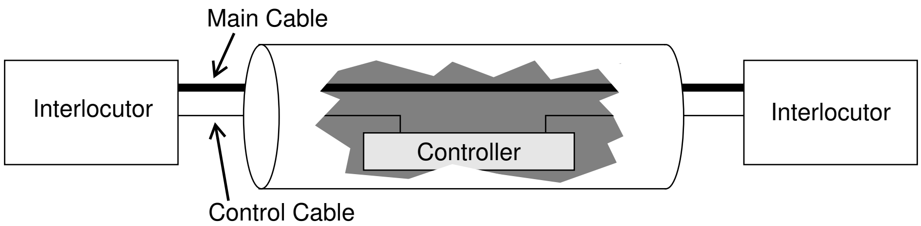

Fig. 11. Structure of the line that composes a safe channel.

Interlocutors have to respect the following protocol:

(1) the default state for an interlocutor is listening to the main cable,
(2) when it wants to emit a signal, the interlocutor asks for the main cable,
(3) if the controller provides the main line, then, the interlocutor sends its message and waits for an acknowledge,
(4) if the controller refuses the main line, then the interlocutor cannot get the line and should retry and retries later on,
(5) interlocutors only send one message at a time,
(6) when an interlocutor gets its acknowledge, it frees the line for another use,
(7) Only messages passing through the main cable are acknowledged,
(8) The control cable is secure,
(9) Signal on the main cable can be lost; however, we assume that connection between the two interlocutors cannot be cut (message loss is bounded).

The table above provides the identification of signals passing through the cables.

Signal name	**Signification**	**Signal direction**		
		Interlocutor		**Controller**
AMC	Ask for main cable		®	
RMC	Refuse main cable		√	
PMC	Provide main cable		√	
MSG	Message	√ ®		
ACK	Acknowledge	√ ®		
FMC	Free the main cable		®	

Fig. 12.

A typical execution scenarios is provided hereafter to illustrate the expected behavior of a interlocutor according to specific situations.

Fig. 13. illustrates the behavior of a interlocutor that initiates a communication when the controller provides the main cable. Then, the answer to AMC (demand) is PMC. The interlocutor (here, 1) then sends the message to the other interlocutor (here, 2) and waits for an acknowledge. When it gets the acknowledge, it releases the main cable (FMC).

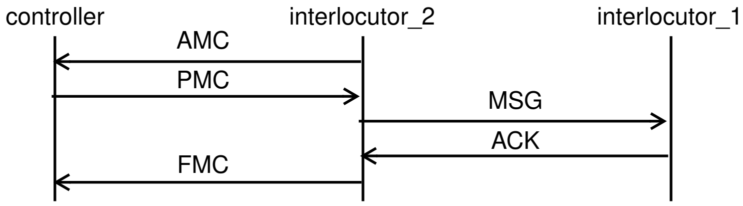

Fig. 13. UML-like sequence diagram of an accepted connection

Let us state some properties we would like to verify on this system:

(i) the controller can never provide the line to more than one interlocutor,
(ii) when an interlocutor decides to send a message, the other one always get it soon or later.

4.1 The CO-OPN model

The CO-OPN model is composed of two classes, representing the controller and the interlocutors. The controller class is rather simple, as shown in Fig. 14. It includes two places denoting an idle and a busy state. Receptions of AMC and FMC trigger the switch from a state to another. In addition, a reception of AMC in busy state triggers the emission of a RMC.

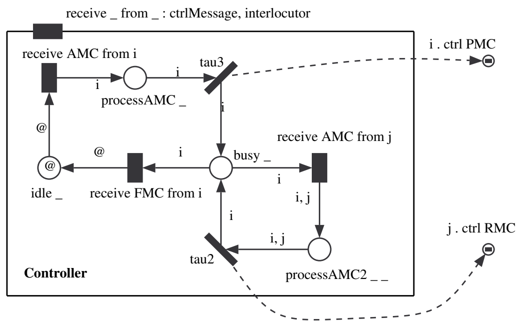

Fig. 14. CO-OPN description of the controller.

The case of the interlocutors is more complex, as shown in Fig. 15. In essence, an interlocutor is composed of two parallel processes; the first process cares about the reception of messages, while the second one cares about the emission.
The reception process, shown on the bottom of the figure, is a simple sequence of message's reception, repetions of message's receptions (due to signal lost, an re-emis-

sions), and finally an ACK emission.
The emission process starts by asking the controller for the main cable, followed either by RMC an retries, or by PMC and the message's emission. At this time, the reception process is stopped (the token in place "listen" is consumed) and new emissions cannot be proceed (a resource in place "prio" is consumed). Then after a bounded number of re-emissions (controlled trough the place "maxTmo") and the reception of an ACK, the process ends and the interlocutor is ready to emit a new message, or to receieve messages again (a token is produced into "listen"). In order to decsribe the class accurately, we had to fix upper-bounds for free values in the system; in particular, we decided to have at most two interlocutors (denoted by +i1" and "i2"), and three message's re-emissions.

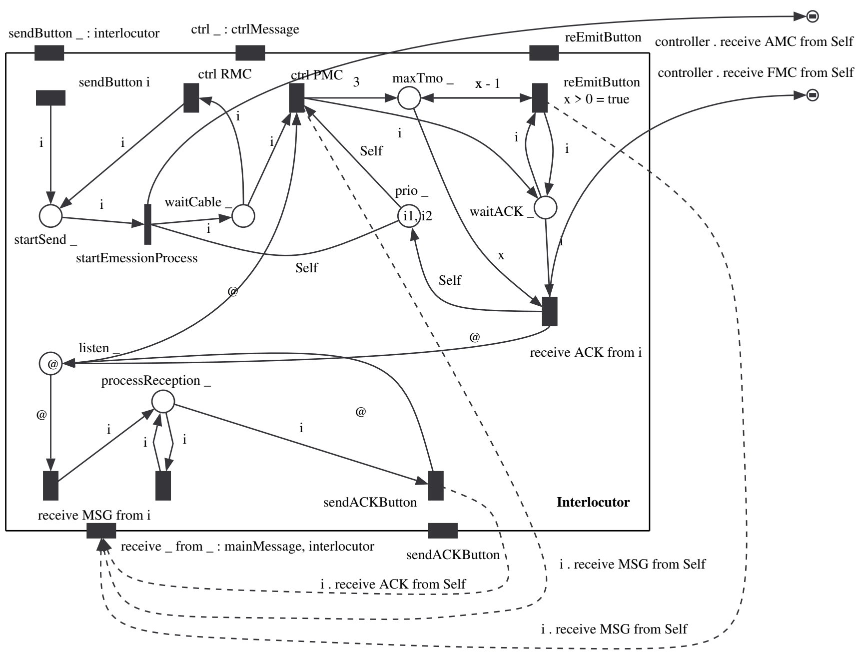

Fig. 15. CO-OPN description of the interlocutor

4.2 Checking template model: the skeleton

Petri net synthesis. The generation of the skeleton of both classes is straightforward, as shown in Fig. 16. and in Fig. 17.

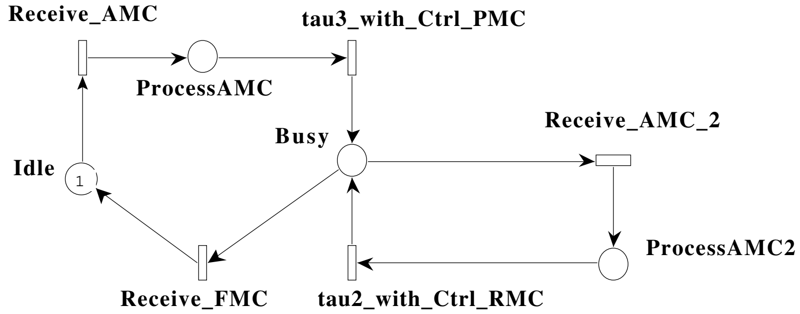

Fig. 16. Skeleton of the controller.

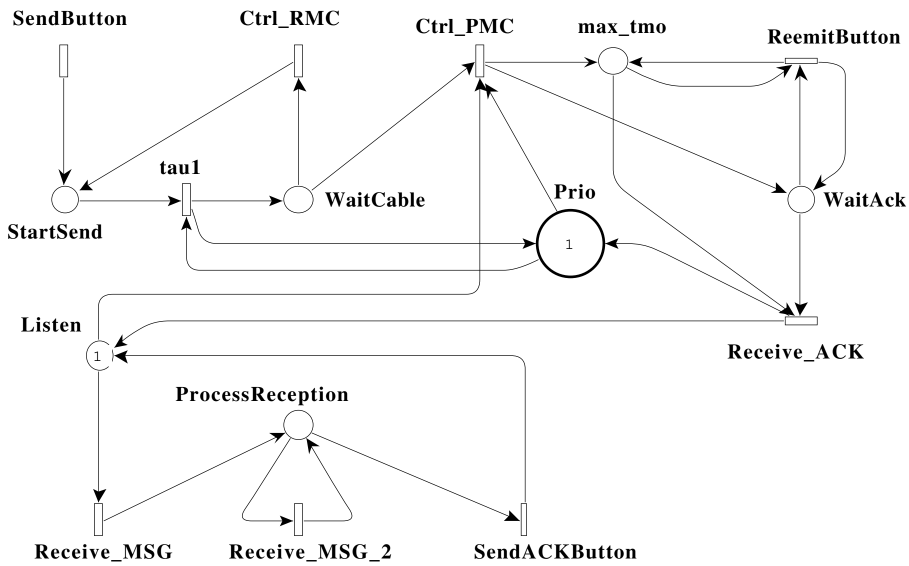

Fig. 17. Skeletton of an interlocutor

The skeleton itself is produced by collapsing both class skeletons, by means of fusion of transition. In spite of the fact that the result is complex, as depicted in Fig. 18., the translation process itself is rather simple. The classes of the system exhibit seven simple synchronizations, i.e. cohercions. Each of these cohercions is represented by a new transition, resulting from the fusion of both the source and the target of the cohercion.

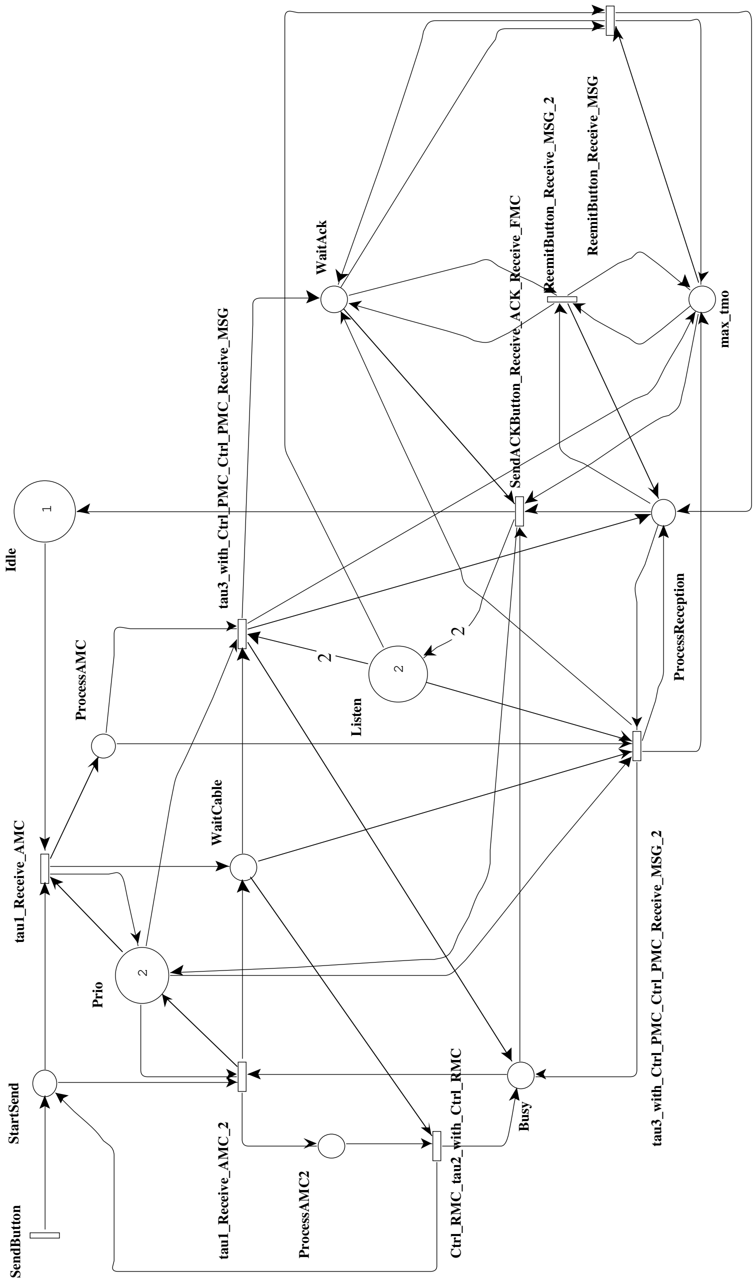

Fig. 18. The skeleton.

Petri net analysis. The check of property (i) is easy to translate in Petri net theory. To verify it, we have to prove that place Busy is 1-bounded (a token in this place represents an interlocutor using the main cable). We can thus expect that some structural invariant help us to prove this assertion. There are two interesting place invariants involving place Busy provided by GreatSPN in CPN-AMI:

- Invariant 1: Idle+ Busy+WaitCable= 1
- Invariant 2: ProcessAMC2 + Busy + ProcessReception + Listen = 1

They both prove that place "busy" is structurally 1-bounded. Then, a token cannot be in more than one of places support of these invariants. We can notice that the second invariant corresponds to the sequential automaton of the controller.

The computation of structural bounds can also be used to check this assertion. in CPN-AMI, the corresponding tool provides us with:

- Prio : [1 ... 2]
- max_tmo : [0 ... 1]
- Listen : [0 ... 2]
- ProcessReception : [0 ... 2]
- WaitAck : [0 ... 1]
- WaitCable : [0 ... 1]
- StartSend : [0 ... $+\infty$]
- Busy : [0 ... 1]
- Idle : [0 ... 1]
- ProcessAMC : [0 ... 1]
- ProcessAMC2 : [0 ... 1]

We also verify the fact that Busy marking cannot exceed one token. We can also deduce that the reachability graph of this Petri net is infinite, due to place StartSend. This is normal: transition SendButton can be fired as many times as possible: it is the interface with some external user.

Thus generation of the reachability graph is useless when we need temporal logic to verify property (ii). An «injection mechanism» has to be introduced in the model in order to roughly simulate a «normal» user that do not stacks SendButton event quicker that the system can afford. To take benefits of the information carried in tokens (essentially, identity of interlocutors), let us perform this operation on the skeleton+ in the next section.

4.3 Checking instance model: the skeleton+

Petri net synthesis.

In essence, the skeleton+ is the skeleton augmented with information regarding the objet identifiers. In our case, we have one controller and an indefinite number of interlocutors. The skeleton+ of our system, depicted in Fig. 19., puts in evidence this similarity; it is almots equals to the skeleton, the arcs of which are decorated with object identifiers.

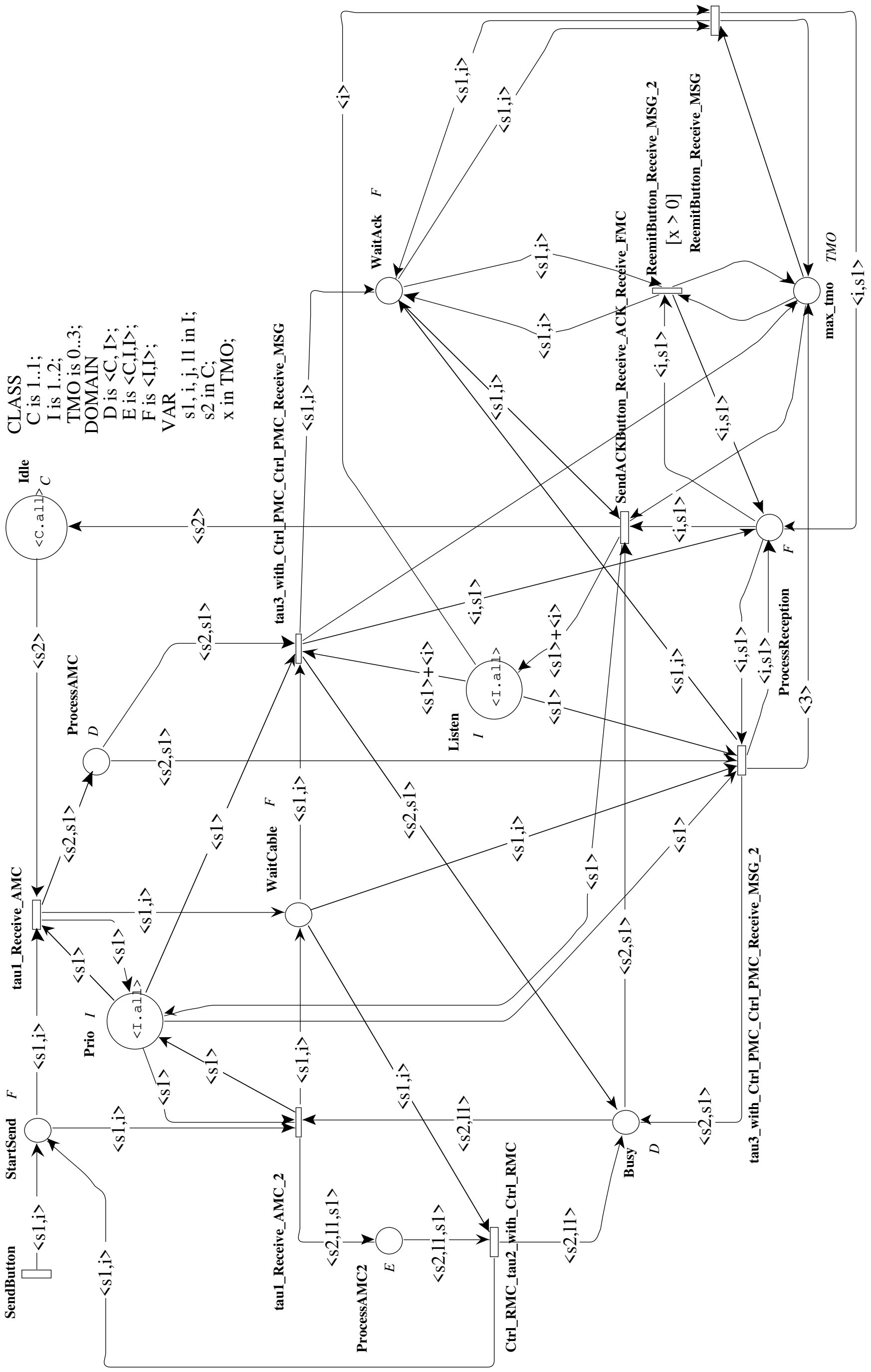

Fig. 19. The skeleton+.

Petri net analysis. As mentioned in the previous section, we have to enrich the Petri net model with some environmental behavior. This is an easy operation because the environment modeling is quite simple there. To bound the number of SendButton event, we introduce two places (Fig. 20.):

- *Emitter_list*: it contains all potential emitters (i.e. all interlocutor instances). It is a precondition of the *SendButton* transition and a postcondition of the *SendACKButton_Receive_ACK_Receive_FMC* transition that corresponds to the end of a communication session (i.e. the line becomes available again),
- *Receiver_list*: it contains all potential receiver. This place provides a value too variable s1 in the Petri net model. Of course, we force in *SendButton*'s guard that $i \neq s1$ (i.e. no interlocutor uses the line to send a message to himself)[1].

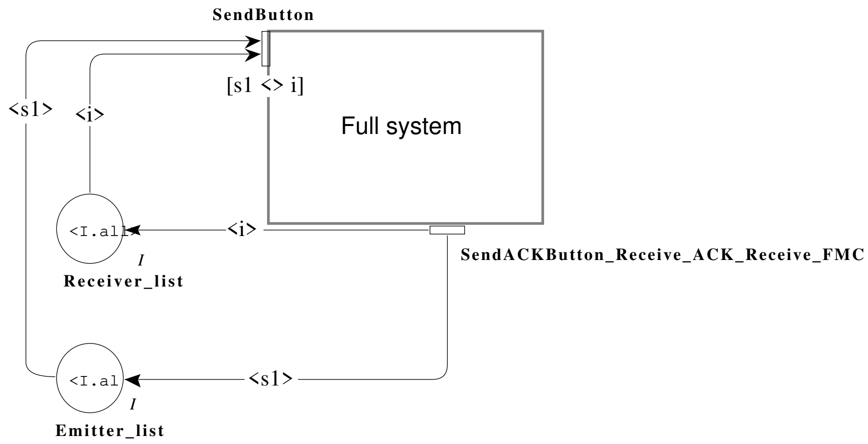

Fig. 20. A first injection mechanism.

Checking of property (ii) can be done by means of two symmetrical CTL queries corresponding to all possibilities in the system (1 send a message to 2 and 2 sends a message to 1). Using PROD, the Petri-Net based model checker integrated in CPN-AMI, they can be expressed as follow:

query verbose AG(IfThen (StartSend ==<.1,2.> == 1,
AF (Emitter_list == <.2.>))) (19)

query verbose AG(IfThen (StartSend ==<.2,1.> == 1,
AF (Emitter_list == <.1.>))) (20)

Textual interpretation of formula (19) is: «when interlocutor 1 decides to send a message to interlocutor 2 (i.e. Send Button as been fired with the corresponding binding), all path in the future lead to a state where the message is acknowledged and the line released (i.e. transition *SendACKButton_Receive_ACK_Receive_FMC* has been

1.If this guard is not set, the reachability graph contains two deadlocks that trivially correspond to self emission of a message.

fired)».

Generation of the reachability graph provides the following information: 14 nodes, 30 arrows and 0 terminal nodes. These queries are not satisfied.

Let us consider query (21), which is (19) with a «EG» (at least one future leads to) instead of a «AF» (meaning any future lead to).

query verbose AG(IfThen (StartSend ==<.1,2.> == 1,
AF (Emitter_list == <.2.>))) *(21)*

Query (21) is verified. Thus, apparently, the problem comes from the reemission mechanism that allows for example the infinite firing of transition *ReemitButton_Receive_MSG_2*. The problem comes from that fact that skeleton+ dos not contain sufficient information about local variables in the CO-OPN specification. Here, such local variables are used to bound message loss. Thus, to check this hypothesis, we have to work on the valued model.

4.4 Checking valued model: the complete description

Petri net synthesis. Due to the fact that the exemple does not include algebraic values, with the exception of the kind of messages, which are already handled in the skeleton, the valued model is equals to the skeleton+, as shown in Fig. 26.

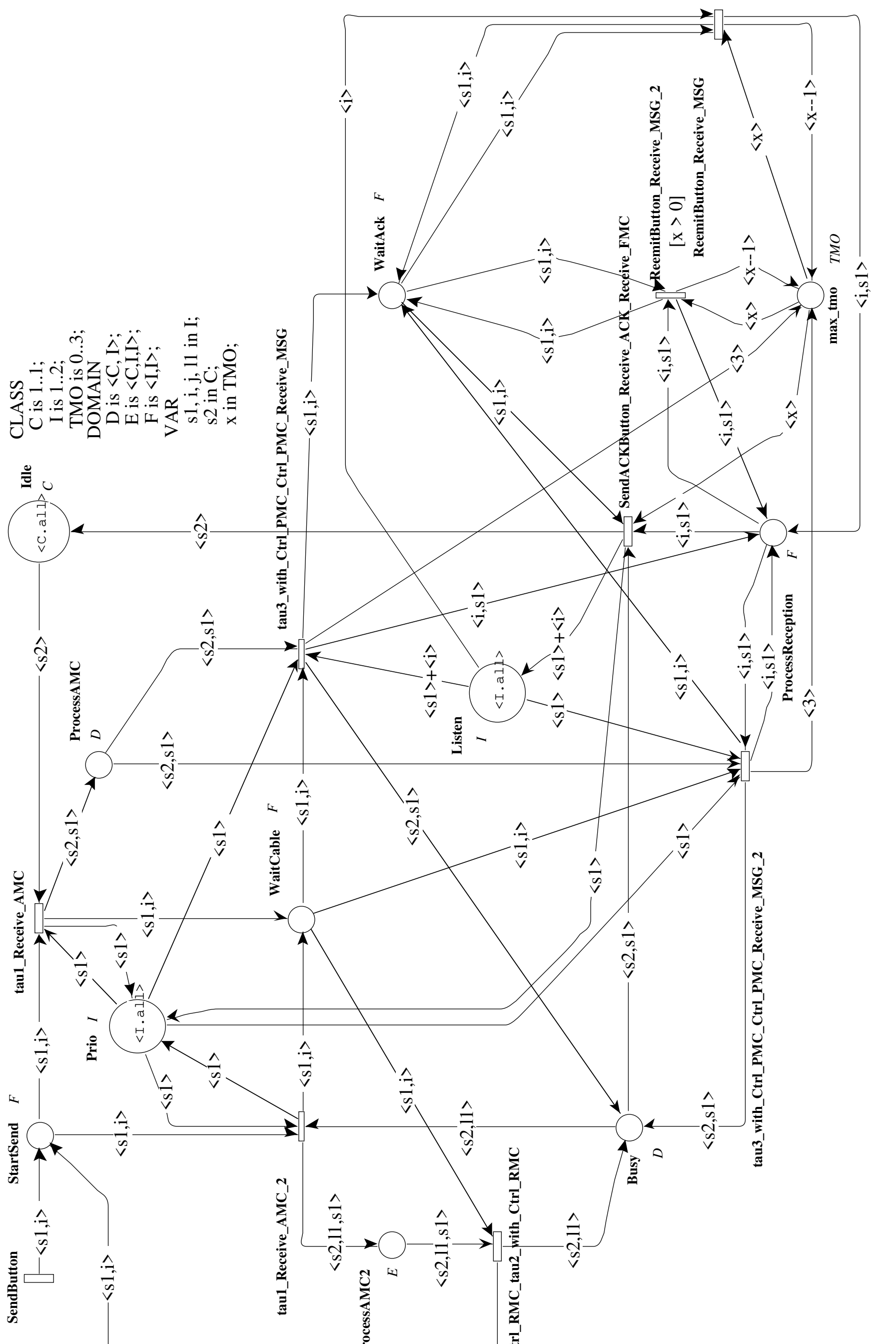

Fig. 21. The valued model.

Petri net analysis.

Let us now apply queries (19) and (20) to the valued model (Fig. 21.) plus the injection mechanism presented in the previous section (Fig. 20.). Generation of the reachability graph provides the following information: 32 nodes, 72 arrows and 0 terminal nodes. Queries (19) and (20) are still not verified, (21) is. The problem was not only due to the reemission mechanism.

Checking for loops in the reachability graph bring us to the following observation: We can infinitely fire *tau1_Receive_AMC_2* and then *Ctrl_RMC_tau2_with_Ctrl_RMC*. It means that an interlocutor may focus on getting the line without listening for an incoming message. Then, if the other interlocutor has the line and waits for an acknowledge to release it, there is a livelock. Such a livelock could be avoided if there is a way for a given interlocutor to know when it can send a message. That could be solved by introducing new constraints in a communication protocol between the two interlocutors *on top of the protocol we are studying*.

Let us verify this hypothesis by changing the modeling of environmental behavior (Fig. 22.). In this new one, we consider a simple deterministic strategy: round robin. Place Emmiter_list only contains one token (here *<1>*). Then, the successor of this token will be produced when the message is sent (this is value *<2>*) and so on. This ping pong mechanism should never stop.

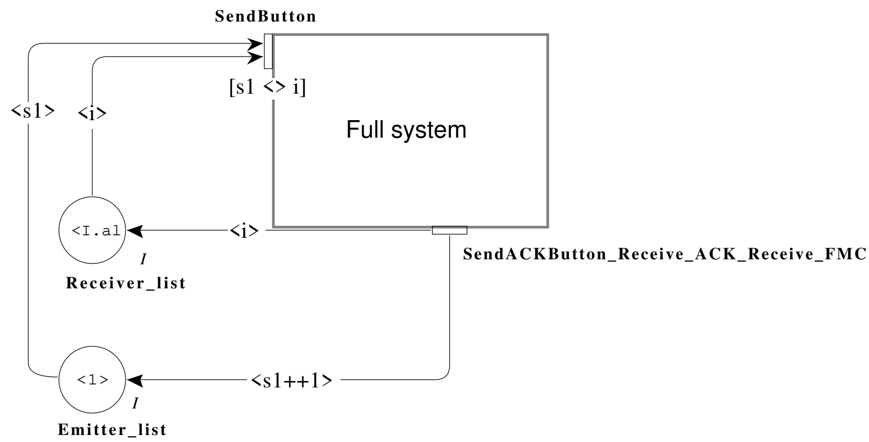

Fig. 22. The elaborated injection mechanism.

Generation of the reachability graph for this new model provides the following information: 14 nodes, 20 arrows and 0 terminal nodes. Queries (19) and (20) are verified.

4.5 Conlusion from the analysis

The conclusion of this verification procedure is that we have to change the protocol description and introduce a new hypothesis in point (10):

(10) There must be a deterministic mechanism that allow processes to know when they can decide not to read from the line or a preemptive mechanism that periodically force an interlocutor to read from the main cable.

According to the verification we have done, this extra point should insure a proper execution of the protocol.

5 Case study 2: Accumulator

The second case study deals with the modeling of a single component, acting as a provider of computing resource. This component must be able to accept a number, perform a pre-defined operation, and finally deliver the result. Obviously, this component should then be able to process a new computing cycle. With regards to the first case study, this example cares about two new complex concepts of CO-OPN, namely the sequential synchronization and the recursive synchronization. Moreover, this example uses an algebraic data type with a complex operation, namely the addition.

More precisely, in this case study, we want to model a component obeying to the following contract:

- the component has a port “start” accepting a natural number and starting a computation;
- the component has a port “result” delivering the result as a natural number;
- the result is defined as the sum of the natural numbers less or equal than the parameter;
- the component is designed to accept sequences of “start” followed by “result”.

We would like to check that this algorithm is convergent.

5.1 The CO-OPN model

The CO-OPN modeling of the accumulator consists in a class, with two methods corresponding to the two ports mentioned above. The first port accepts values to be computed, and put the result in a dedicated place, namely “r”. The second port takes the result from this place and delivers it. Looking at the class, we see that the first port, “start”, has two behaviours:

- for a positive number “succ n”, after a transit through an apposite place “i”, a recursion is performed to compute the result for the value “n”, which is used to put the current result in place “r”;
- for a null number, i.e. the end of the recursion, the value zero (i.e. “zero”) is put in place “r”.

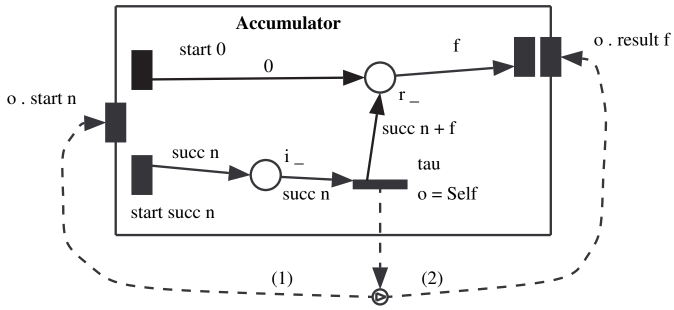

Fig. 23. The "Cumulator" Class

It seems interesting to present now a CO-OPN model of the typical environment for this accumulator class, as this model may serve as the basis for the description of the injection mechanism used during the formal analysis.

The typical environment is the class called "AccumulatorEnvironment" depicted in Fig. 24. The instances of this class repeat continuously sequences of method calls on "start" and "result". The model includes a place acting as a reservoir of possible parameters for the computation.

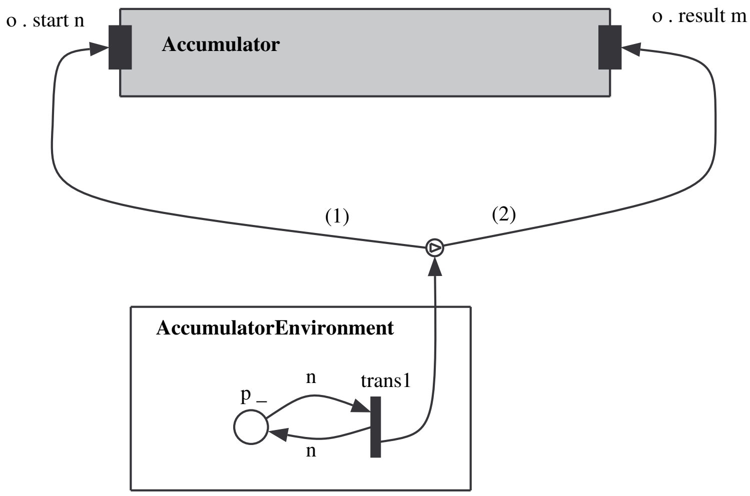

Fig. 24. The Accumulator Environment

5.2 Checking template model: the skeleton

Petri net synthesis. We first look on the template model, based on the skeleton of the accumulator class. The skeleton is obtained by transition expansion and fusion.

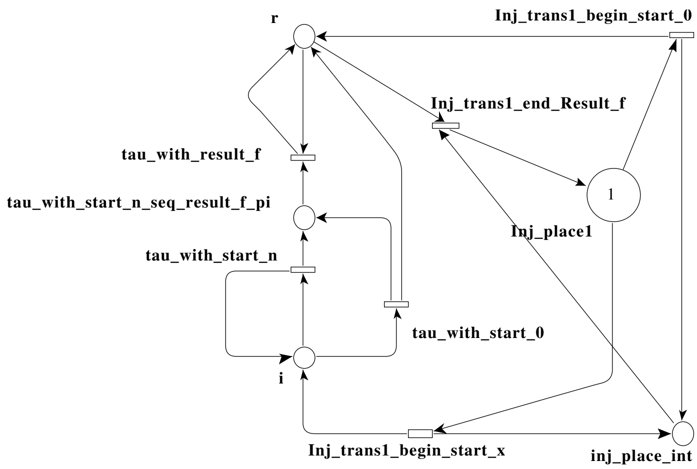

Fig. 25. The skeleton.

The first step in the process of skeleton generation consists in the expansion of the sequential synchronization. We achieve this goal by cutting "tau" in two transitions connected through a place "tau_pi", reflecting the two steps of the sequence. Each of these transitions are then fusionned with the apposite transition, reflecting the actual synchronization at each step; this generates the transitions "tau with start n" and "tau with result f". Fig. 25. shows the resulting skeleton.

Petri net analysis. The convergence of the algorithm can be verified if place *tau_with_start_n_seq_result_f_pi* is bounded. Let us apply the structural boun tool that says it is not the case.

We can easily understand that a recursivity is not structurally bounded. Moreover, the end of a recursion can be decided using values of parameters. These values are not expressed in the skeleton model. Thus, the property is not decidable.

Let us note that GreatSPN provides us wit two place invariants:

- *Inj_place1+inj_place_int* corresponds to the terminal case,
- *i+r+Inj_place1* corresponds to the general recursive case.

5.3 Checking instance model: the skeleton+

Petri net synthesis.

As mentioned before, this case study deals with the modeling of a single component. Accordingly, the resulting CO-OPN system is composed of a single instance of the class "Accumulor".

Hence, the skeleton+ and the skeleton are equals, except that each place and transition of the skeleton+ are overloaded with a constant object identifier, with regards to the skeleton. In other words, the skeleton+ and the skeleton are equivalent.

Petri net analysis. The analysis of the skeleton+ does not bring any more information than the one of the skeleton. This is due to the fact that local variables of the cumulator carry out to much information; thus, the skeleton+ is almost as empty as the skeleton.

5.4 Checking valued model: the complete description

Petri net synthesis.

According to the translation strategy presented in Section 3, the valued model integrates the unfolding of algebraic data types, as shown in Fig. 26. In addition, we adopted here an ad-hoc strategy to associate partially computed values with their recursive level (i.e. to simulate the recursive stack).

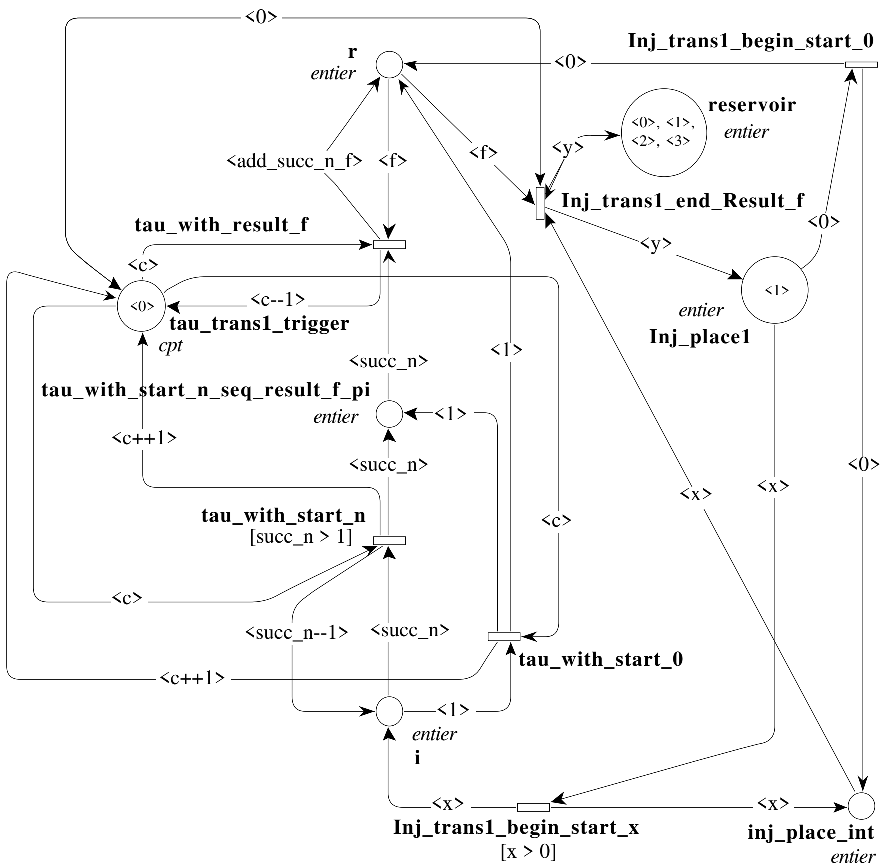

Fig. 26. The valued model.

Petri net analysis. Once again, computation place bounds (after unfolding of the colored Petri net) shows that place *tau_with_start_n_seq_result_f_pi_entier* is unbounded. There is thus no structural bound but it is of interest to check if this structural bound if reached (structural bounds are sometimes larger than effective bounds).

Another way to check if the computation is convergent is to evaluate the following CTL query:

$$\text{query verbose AG (IfThen (card(inj\_place\_int) == 1, AF (card (Inj\_place1) == 1)))} \tag{22}$$

Textual interpretation of formula (22) is: «when a computation is started (e.g. a token is dropped in place *inj_place_int*, all future lead to a state where a new computation can be performed (i.e. for this model that performs only one computation at a time, there is a token in place *Inj_place1*)».

Generation of the reachability graph for this new model provides the following information: 91 nodes, 679 arrows and 0 terminal nodes. Query (22) is satisfied.

5.5 Conlusion from the analysis

We have been able to perform some verification on the Petri net model generated for the second case study. However, conclusions are less optimistic than the ones of the first case study.

Basically, the main reason is the complexity of the model. While structural bounds cannot be used to prove the property we are expecting, we have to explore a reachability graph having a bad complexity increase. Thus, it is almost impossible to check the system for large values (for example 1000). Moreover, the computation of structural properties on the corresponding P/T net (obtained by unfolding of the Colored Petri net) is almost impossible because of the combinatorial explosion of the resulted P/T net. This combinatorial explosion is illustrated by the table below.

Max value of N	**Reachability Graph**		**Unfolded P/T model**		
	nodes	**arcs**	**places**	**Transitions**	**Arcs**
3	91	679	60	1746	9402
4	135	1543	93	8047	42875
5	190	3118	134	28778	152050
6	256	5734	183	85353	448017
7	333	9787	240	219754	1147490

Fig. 27.

This is a typical illustration of a lack in Petri nets: they are not suitable for the management of numeric variables and computations. Thus, analysis potential is very limited in this area.

Another point is the difficulty to check that the algorithm produces a good result. This means that the arithmetic «+» has to be implemented, which is not easy, especially for a large interval. Thus, this part of the work was dropped.

6 Conclusions

We have proposed in this paper a prototyping approach based on CO-OPN and AMI-Nets (colored Petri Nets). CO-OPN, the entry point of our approach proposes a high level description language based on semantic construction dedicated to the management of concurrent systems. It has the following nice features :

- it relies on a sound formal semantics that enables definition of expected properties;
- it proposes most of the main nice structuring capabilities expected from a object oriented language.

These characteristics enable the use of Petri-nets as a verification formalism. These Petri nets can be used transparently. Thus, our approach can be used without having to be an expert in formal methods.

Our approach requires various formally defined translations from CO-OPN constructors into Petri nets elements. The resulting nets are likely to be analyzed using a dedicated tool. In this paper, our examples have been analyzed with CPN-AMI [22].

Our experience shows that such an approach is of interest for some kinds of systems. However, some other lead to Petri net models that are too complex to be handled efficiently automatically. In this last case, a deep knowledge of the Petri net formalism is required to expect manual simplifications. In particular, we experienced that, the more complex algebraic types are, the more difficult the analysis is. On the contrary, our approach remains very suitable to manage the control aspects of concurrent systems. This appears to be a lack observed in Petri nets (poor management of computational aspects).

We already started the implementation of a prototype tool allowing a semi-automatic translation of CO-OPN models into Petri nets. We must enhance now the theoretical aspects, the implementational aspects, as well as the methodological aspects of our method. This will provide the system designer a guide to have an appropriate use of such an approach.

References

[1] H. Bachatène & J.M. Couvreur, "A Reference Model for Modular Colored Petri Nets", in proceedings of IEEE/System, Man and Cybernetics International Conference, Le Touquet, France, October 1993

[2] R. Bastide, "Approaches in unifying Petri nets and the Object-oriented Approach",1st Workshop on Object-oriented Programming and Models of Concurrency, Torino, Italy, 1995

[3] O.Biberstein & D. Buchs, "Structured Algebraic Nets with Object-Orientation", Proceedings of the first international workshop on Object-Oriented Programming and Models of of Concurrency, in the 16th International Conference on Application and Theory of Petri Nets, Torino, Italy, June 1995

[4] O. Biberstein, D. Buchs & N. Guelfi, "Object-Oriented Nets with Algebraic Specifications: The CO-OPN/2 Formalism", To appear in Advances in Petri Nets on Object-Orientation, G. Agha and F. De Cindio Eds, LNCS, 1999

[5] M. Buffo, "Experiences in Coordination Programming", Proceedings of the workshops of DEXA'98 (International Conference on Data base and Expert Systems Applications), IEEE Computer Society, Vienna, Austria, Aug 1998

[6] G. Chiola, C. Dutheillet, G. Franceschini & S. Haddad, "On Well-Formed Coloured Nets and their Symbolic Reachability Graph", High Level Petri Nets. Theory and Application. Edited by K. Jensen G.Rozenberg, Springer Verlag 1991

[7] G. Chiola, G. Franceschinis, R. Gaeta & M. Ribaudo, " GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets", in Performance Evaluation, special issue on Performance Modeling Tools, 24(1&2), pp47-68, November 1995

[8] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes & P. Jeremaes "Object-Oriented Development: the Fusion Method", Prenctice-Hall, 1994

[9] J.M. Couvreur, "The general computation of flows for coloured nets", 11th International Conference on Application and Theory of Petri Nets, pp204-223, Paris, France, June 1990

[10] J. Esparza, S. Römer & W. Vogler, "An Improvement of McMillan's Unfolding Algorithm", in proceedings of Tools and Algorithms for the Construction and Analysis of Systems, LNCS 1055, pp 87-106, Springer Verlag, March 1996

[11] W. Gibbs, "Software's Chronic Crisis," Scientific American, Sep. 1994, pp. 86-95

[12] J. Goguen, "Requirements Engineering as the Reconciliation of Social and Technical Issues," in Requirements Engineering: Social and Technical Issues, M. Jirotka and J. Goguen eds., Academic Press, pp. 165-200, London, 1994

[13] B. Grahlmann, “The State of PEP”, in the Proceedings of AMAST’98 (Algebraic Methodology and Software Technology), LNCS 1548, Springer Verlag 1999

[14] V. Janousek, "PNtalk: Object Orientation in Petri nets", in proccedings of European Simulation Multiconference ESM'95, Prague, pp 196-200, June 1995

[15] J.Kerr & R. Hunter, "Inside RAD", McGraw Hill, 1995

[16] F.Kordon & J-L. Mounier, "FrameKit, an Ada Framework for a Fast Implementation of CASE Environments", in proceedings of the ACM/SIGAda ASSET'98 symposium, pp 42-51, Monterey, USA, July 1998

[17] E. Koutsofios & S.C. North, "Drawing graphs with dot", Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1993

[18] C. Lakos & C.D. Keen, "An Open Software Engineering Environment Based on Object Petri Nets", Technical Report TR95-6, Computer Science Department,University of Tasmania, Australia, 1995

[19] N. Leveson, "Software Engineering: Stretching the Limits of Complexity", Communications of the ACM, Vol 40(2), pp 129-131, February 1997

[20] Luqi & J. Goguen, "Some Suggestions for Progress in Software Analysis, Synthesis and Certification," in proceedings of the 6th International Conference on Software Engineering and Knowledge Engineering, Knowledge Systems Institute, pp. 501-507, Skokie, USA, 1994

[21] Luqi & J. Goguen, "Formal Methods: Promises and Problems", IEEE Software, Vol 14, N°1, pp 75-85, January 1997

[22] MARS-Team. CPN-AMI Home page. http://www-src.lip6.fr/cpn-ami.

[23] MARS-Team. Macao Home page. http://www-src.lip6.fr/macao

[24] E. Pastor, O. Roig, J. Cortadella & R.M. Badia, "Petri Net Analysis Using Boolean Manipulation", in LNCS 815, Springer Verlag, R. Valette (ed.), Proceedings 15th International Conference on Application and Theory of Petri Nets, Zaragoza, Spain, 1994

[25] T. Quatrani, "Visual Modeling with Rational Rose and UML", Addison-Wesley, ISBN: 0-201-31016-3, 1998

[26] W. Reisig, "Petri nets and algebraic specifications", In Theoretical Computer Science, volume 80, pages 1–34. Elsevier, 1991.

[27] Standish Group International, "Chaos 97 technical report", Internal report, available on <http://www.standishgroup.com/chaos.html>, 1995

[28] K. Varpaaniemi, J. Halme, K.Hiekkanen & T.Pyssysalo, "PROD reference manual", Technical Report B13, Helsinki University of Technology, Digital Systems Laboratory, Espoo, Finland, August 1995

[29] J. Vautherin, "Un modèle algébrique, basé sur les réseaux de Petri, pour l'étude des systèmes parallèles", PhD thesis, Université de Paris-Sud, LRI, 1985