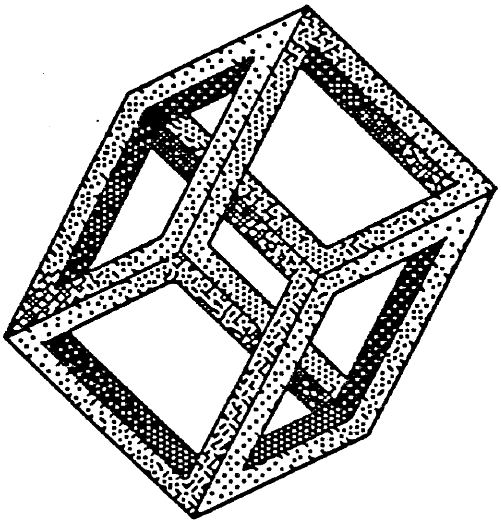
OCTOBRE 1988 N° 256



Laboratoire
METHODOLOGIE
&
ARCHITECTURE
DES SYSTEMES
INFORMATIQUES

GENERATION OF ADA CODE FROM PETRI NETS MODELS

Bernard COUSIN, Pascal ESTRAILLIER

UNIVERSITE P. ET M. CURIE CNRS
Laboratoire MASI UA 818
4 Place Jussieu, 75252 PARIS cedex 05 FRANCE

Generation of ADA code from Petri Nets models

Bernard COUSIN Pascal ESTRAILLIER

Laboratoire MASI - CNRS UA 818
Université Pierre et Marie CURIE
4, place Jussieu
75252 PARIS cedex 05 - FRANCE
téléphone : (1) 43.36.25.25 - poste. 3192

Article présenté au "3rd International Symposium
on Computer and Information Sciences",
du 29 octobre au 2 Novembre 1988,
à Izmir - TURQUIE

Résumé

Nous présentons une méthode automatique de **génération de code** efficace. Le code **ADA** est obtenu à partir d'un modèle formel décrit en **réseaux de Petri** exprimant le schéma de contrôle de l'application parallèle.

La méthodologie repose sur la décomposition de l'implémentation en 3 phases permettant l'intégration progressive des contraintes liées à l'application, à son environnement et au langage de programmation.

Nous introduisons une technique originale pour tendre vers une implantation **optimale**, minimisant le nombre de tâches supportant l'exécution du l'application tout en conservant le degré de parallélisme potentiel du modèle. L'idée majeure consiste à réutiliser les invariants produits par l'étape de validation du modèle pour déterminer le nombre minimal de tâches. Cette technique permet la caractérisation des types d'objets utilisés par l'implantation.

Nous appliquons la méthode proposée à un modèle conçu spécialement pour couvrir l'ensemble des problèmes soulevés, et nécessitant l'ensemble des diffèrents types d'objets.

Abstract

We propose a automatic method for the **generation** of efficient code. The **ADA code** is produced from a formal model using **Petri nets**. The formal model describes the control scheme of the parallel application.

Our methodology organizes the implementation into 3 steps that allow the progressive integration of the constraints of the application, of the environment, and of the programming language.

We introduce a new technique to tend towards an **optimal implementation** that minimizes the number of processes. This set of processes, that support the execution of the application, must keep the intrisic parallelism of the model. The major idea is to reuse the linear invariants given by the validation of the model. These invariants allow us to define the optimal process number. This technique characterizes the objects used by the implementation steps.

We apply our method on a model which has been especially built to cover many technical problems and to include all the characterized object types.

1. INTRODUCTION

The parallel systems and their complexity have brought to the fore the advantages obtained during the conceiving stage by **formal models**. Supported by strict mathematical results, this kind of model enables the characteristics and the parameters of the system to be studied before its implementation.

Petri Nets, as a formal model, has already demonstrated its appropriateness to solve the problems raised by the study of distributed systems [Ayache 85]. But code generation is a very important point to allow Petri nets to be used in industrial environments. Activities in this area have a tradition in the context of Programmable Logic Controllers, for centralized logic automation applications.

Code generation in software applications has recently become most interesting because of the emergence of parallel architectures and new programming paradigms and objectives. Imperative programming is chosen most of the time, for example : **ADA** [Colom 86], CHILL, OCCAM [Steinmetz 86] and others [Nelson 83]. For specially tailored classes of nets, an object-oriented programming paradigm is being explored for Petri net model prototype implementation [Bruno 86]. Special characteristics and architecture of some distributed applications (for example telecommunications) allow the use of even more direct implementation.

The objectives of the project **PN_TAGADA** (Petri Net _ **T**ranslation, **A**nalysis and **G**eneration of **ADA** code) are to design and implement tools performing code generation in a context of distributed architecture. So we need to design and implement a tool for producing semi-automatically efficient parallel code from a validated Petri net model.

In our project, the code is generated from a validated model. There are many advantages :

. The model is a formal description containing the engineering requirements.
. The model has been verified and no design errors remain; The early detection of errors is justified from an economic point of view.
. We can use the structural and behavioural properties, issued from the validation of the model, to optimize code generation.

In a Petri nets model, each object (place and token) can be seen in terms of an elementary task that synchronizes with other objects by means of transitions. The basic idea for an efficient implementation is to group together several non-concurrent elementary tasks in one sequential process, to minimize the synchronization.

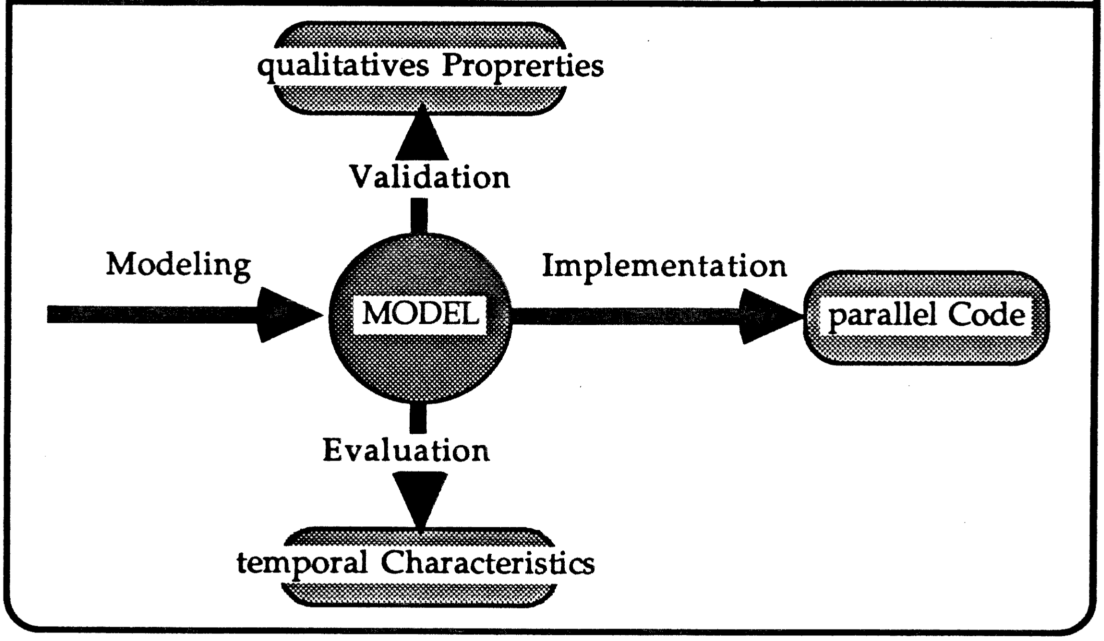
Among the high-level concurrent languages (ADA, OCCAM, CHILL, etc...), ADA has been chosen because it provides advanced tasking and structuring mechanisms, but we are only just beginning to study generation of OCCAM code for a multi-transputer system.

2. METHODOLOGY

2.1 The phases

From the specification of the system the **modeling** phase builds a model that describes all synchronizations (the precedence between tasks, the control access to global variables, communications, etc...) (Figure 1).

- The **validation** phase enables the qualitative properties from the model to be obtained. This phase is useful to check the conception of the system [Cousin 87].
- The **evaluation** phase constitues the quantitative phase of the exploitation of the model. It checks the temporal caracteristics of the system with the requirements.
- The **implementation** phase is a natural continuation of the work done by the previous phases. The model (verified by the validation and evaluation phases) is used as a basic control scheme to generate the synchronisation code of the parallel system.



Formal model environment

Figure 1

2.2 The steps

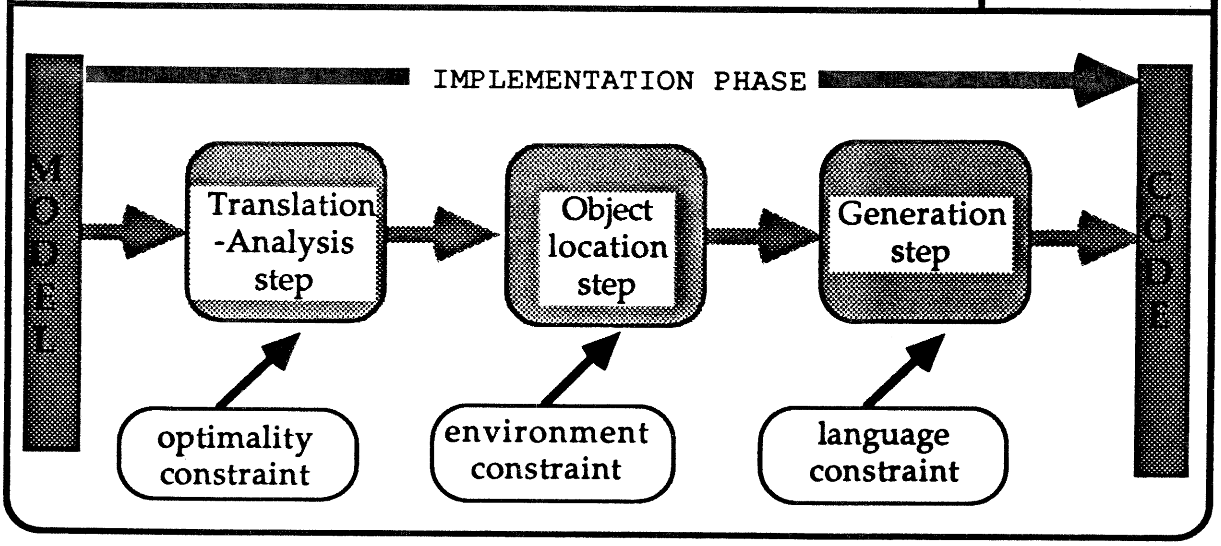
Our methodology organizes the implementation into 3 steps (translation and analysis, location, generation). These steps allow the progressive integration of the constraints respectively of the application, of the environment, and of the programming language, to meet the objectives (quality and performance criteria) (Figure 2).

Each step defines a set of new **objects** taking into account the constraints, using the objects defined by the previous step :

- The first step is concerned with semantics and the internal synchronizations of the model. It performs the **Translation** and the **Analysis** of the validated model. The model is decomposed into a set of cooperating sequential processes which may be obtained from the analysis of the control structure of the model. Problems concerning conflicts need a careful consideration (deadlock, management of shared resources, etc...). This step produces a collection of objects needed for the next step.

- The second step defines the object **location** for a specific distributed architecture. It manages the allocation of the hardware and software resources.

- The third step **generates** the code.The semantic rules of Petri nets behaviour have to be expressed within the programming language. Moreover, new constraints may be added by the language itself.



Translation and Analysis - Location - Generation

Figure 2

For the Translation and Analysis step, we presented in a previous paper [Cousin 88], a technique tending towards an **optimal implementation** that minimizes the number of processes. The reduction minimizes the complexity of the synchronization needed to control the parallel system. Nevertheless this set of processes, supporting the execution of the parallel system, maintains the intrisic parallelism of the model.

Rather than analyse the formal model to detect the optimal set, we reuse the **linear invariants** obtained from the validation phase of the model. An invariant is a conservative flow of the model which can be interpreted as the sequential states of a process. These invariants allow us to define the process distribution over the model, and then the optimal process number.

This Analysis of the model and the Location constraints of the second step enable the objects used by the generation step to be easily characterized. We determine in the following paragraph the types of the objects involved in the code generation.

3. APPLICATION

3.1 The Objects

The following object types must be defined for the code generation. **State** objects are derived from the places of the Petri net model, while **Action** objects are derived from the transitions of the Petri net model. The distinction into subtypes is done by Analysis and Location steps, on the basis of model and architecture constraints: **process** or **resource** for the state objects, **process** or **synchronization** for action objetcs.

State

- **Process** : models a possible state of a process.
  - Simple : only one action is possible from this state.
    (at most one transition is potentially fireable from the place)
  - Alternative : many actions are possible from this state.
    (more than one transition is potentially fireable from the place)

- **Resource** : models a resource (data, message, record, ...)
  - Private : exclusively accessed by a process.
  - Shared : can be accessed by many processes.

Action

- **Process** : models an action to be performed by one process.
  - Simple : no guard is associated with the execution of the action.
    (a guard is the condition for the existence of one or more resources)
  - Guarded: a guard may prevent the execution of the action.
- **Synchronization** : models an action which must be performed by cooperating processes in a synchronous way.
  - Simple : no guard is associated with the action.
  - Guarded: a guard may prevent the execution of the action.

3.2 The Model

We apply our method on a model which has been especially built to cover many technical problems and to include all the object types (Figure 3).

The model specifies three distributed processes {x, a, b}. We briefly describe their behaviour :

The process x, when all the conditions are satisfied (depending on marking of places **xPRL** and **PRG0**) prepares (transition **xTP3**) a message (Place **PRG7**) and informs both processes a and b.
The processes a and b have the same symetric structure. So they compete in treating the message: one becomes the sender (transition **TP1**), the other becomes the receiver (transition **TP3**). Only one of them is allowed to send the message (transition **TP2**) to the other process which receives it (transition **TP4**).
Then, both processes synchronously (transition **abTS**) return to their home states (respectively places **aPP1** and **bPP1**).Then the same execution can be done again.

The figure 4 exhibits the type of each object obtained after the analysis and location steps. The places and transitions prefixed by a, b, or x belong to the associated process. The PRGi places are ressource objects shared by several processes.

3.3 The Generation

The problematic of the generation phase is to use judiciously the target langage and its specific objects. The similarity between the Petri net places and transitions and the ADA task and rendez-vous concepts make code production easier. Nevertheless a systematic translation from places and transitions to tasks and rendez-vous renders the code inefficient.

A close study of transition firing shows that it can be split into three operations :
- the precondition requires all the processes and the resources needed to execute the action.
- the action associated with the transition executes the processes on the ressources .
- the postcondition releases all the processes and the resources used during the execution.

Our purpose is to reduce synchronization involved in the precondition and the postcondition of each transition. For example :
- a transition, linking two successive states of a same process, needs no precondition because the execution of an ADA task is obviously sequential.
- structures like "if or case" are simpler and more efficient than "rendez-vous" to synchronize elementary tasks of the same process.

Our code generator is written in ADA, has about 2800 lines of code, and uses packages intensively to facilitate the generic production of target code. The figure 5 shows the code produced from the application model by the generator. We can distinguish the code of the three ADA task associated with the three model processes.

4. CONCLUSION

The objectives of the project **PN_TAGADA** are to study the specific problems caused by the code generation from Petri net models, and the design of an automatic code generator.

The technical problems solved are :
- the dissolving of the distinctive control of the Petri Nets model in ADA code,
- the reduction of the processes number,
- the resolution of distributed conflict,

to obtain an efficient code generator.

Code generation may be useful in rapid prototyping for two different purposes. The first is to show preliminary system properties to a customer without spending too much effort in the design process. The second is the evaluation of specific system properties in order to guarantee a suitable system design. In neither case can it be expected that a product system be the result of this process. However, in parallel with the prototype system parts of the system specification for the final product can be realized.

Acknowledgments
Our grateful thanks to G.Headland for his helpful rereading.

References

[Ayache 85] J.M.Ayache, J.P.Courtiat, M.Diaz, G.Juanole. *Utilisation des réseaux de Petri pour la modélisation et la validation de protocole.* TSI vol 4 n°1 p51, janvier 1985.

[Bruno 86] G. Bruno and A. Balsamo. *Petri nets-based object-oriented modelling of distributed systems.* OOPSLA 86 proceedings, p 284, 1986.

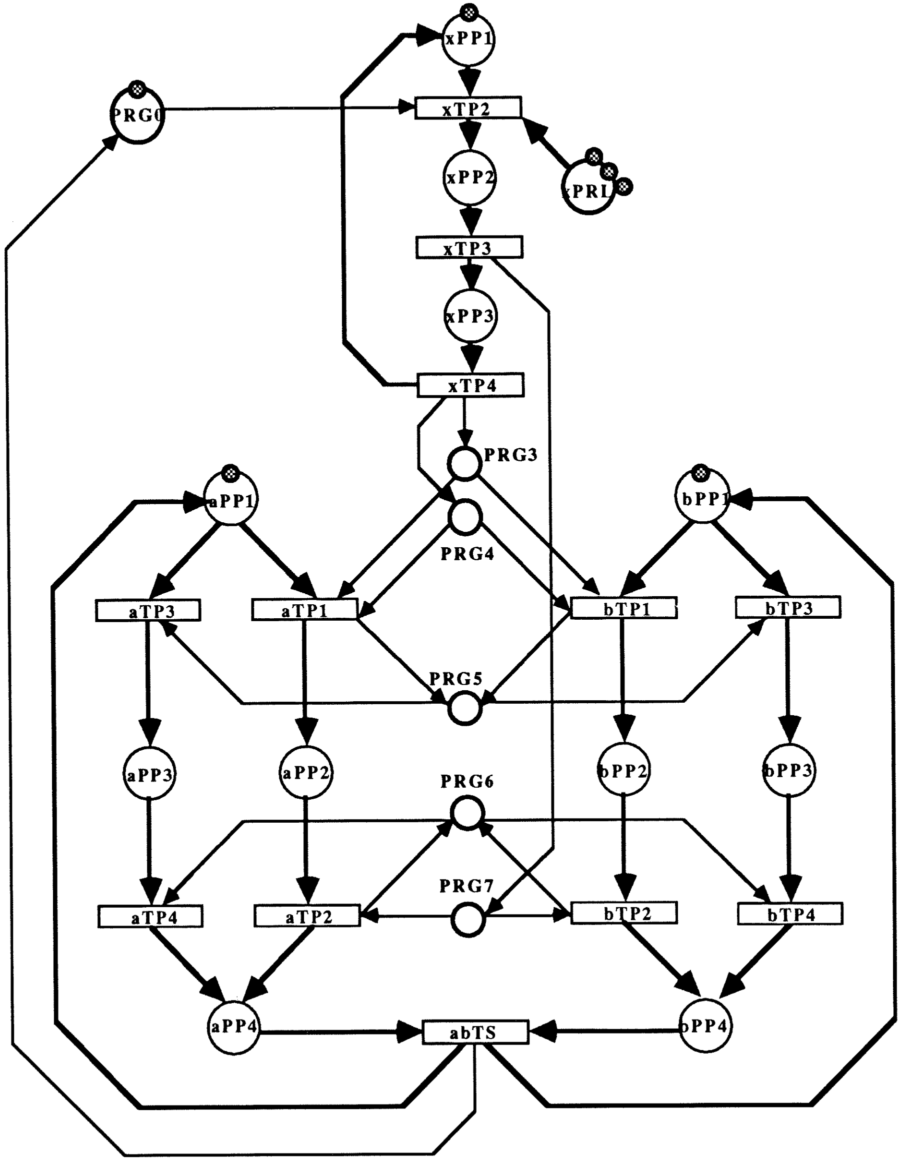
[Colom 86] J.M. Colom, M. Silva, J.L. Villarroel. *On software implementation of Petri Nets and colored Petri Nets using high-level concurrent languages.* 7th European Workshop on Petri net Applications and Theory, Oxford-England, July 1986.

[Cousin 87] B.Cousin. *Méthodologie de validation des systèmes structurés en couches.* Thèse de l'univeristé de PARIS 6, Avril 1987.

[Cousin 88] B.Cousin, P.Estraillier. *Traduction et analyse d'un modèle en vue d'une implémentation optimale.* Rapport MASI n° 219, janvier 88.

[Nelson 83] Nelson R.A, L. Haibt, P.B. Sheridan. *Casting Petri nets into programs.* IEEE Trans. on Software Engineering vol.9 n°5, p 590-602, September 1983.

[Steinmetz 86] R.Steinmetz. *Relationships between Petri nets and the synchronization mechanisms of the concurrent languages CHILL and OCCAM.* 7th European Workshop on Petri nets Applications and Theory, Oxford-England, July 1986.



- The Model - Figure 3

S T A T E	Processus	Simple	xPP1, xPP2, xPP3 aPP2, aPP3 bPP2, bPP3
		Alternative	aPP1, aPP4 bPP1, bPP4
	Resource	Private	xPRL
		Shared	PRG0, PRG3, PRG4 PRG5, PRG6, PRG7
A C T I O N	Processus	Simple	xTP3, xTP4
		Guarded	xTP2 aTP1, aTP2, aTP3, aTP4 bTP1, bTP2, bTP3, bTP4
	Synchro-nization	Simple	
		Guarded	abTS

The places (xPP1, aPP1, bPP1) contain one token, they are the initial states of respectively the x, a, b processes. The place (xPRL) contains three local resources, whereas place (PRG0) contains one global resource.

- Characterization of the Objects of the model - Figure 4

```
with aleatoire, transition, ress_loc, rg, geslist, struct_res, text_io;
use aleatoire, transition, struct_res, text_io;
procedure PARTIR is
package iio is new integer_io (integer);
use iio;
        package ressource_g is new RG (ress_glob);
        use ressource_g;
        package ptr_rg is new geslist (ress_glob);
        use ptr_rg;
        task processus_B is                                                       -- declaration de la tache processus_B
                entry marq_init (init :in natural);
        end processus_B;
        task processus_A is                                                       -- declaration de la tache processus_A
                entry marq_init (init :in natural);
        end processus_A;
        task processus_X is                                                       -- declaration de la tache processus_X
                entry marq_init (init :in natural);
        end processus_X;
        task T_SYNC is                                                            -- declaration de la tache T_SYNC
                entry pret (t_precond :in tplace; reponse :out boolean);
                entry accuse (t_precond :in tplace; reponse :out boolean);
        end T_SYNC;
        max_att : natural;
task body processus_B is                                                          -- I/processus: processus_B
        chx_B_PP1: tab_etat (1..2) := (1 => processus_B_B_TP3, 2 => processus_B_B_TP1);
        etat : natural;continue :boolean := true;precondition: boolean;reponse: boolean;bidon : boolean;
        cpt : natural;g_lst_loc : ptr_rg.pt_el;
        begin
                accept marq_init (init :in natural) do etat := init;end marq_init;
                while continue loop
                        case etat is
                                when processus_B_T_SYNC =>                -- T_SYNC (I/Transition de Synchronisation Simple)
                                        reponse := false;
                                        while not (reponse) loop
                                                T_SYNC.pret (IPPS, reponse);
                                        end loop;
                                        reponse := false;
                                        while not (reponse) loop
                                                T_SYNC.accuse (IPPS, reponse);
                                        end loop;
                                        etat := processus_B_B_PP1;
                                when processus_B_B_PP1 =>                 -- B_PP1 (IPlace Processus Alternative)
                                        etat := tirage (chx_B_PP1);
                                when processus_B_B_TP1 =>                 -- B_TP1 (I/Transition Processus Conditionnee)
                                        precondition := true;
                                        if precondition
                                        then
                                                g_lst_loc := NULL;
                                                if gconsomme (PRG4)
                                                then
                                                        g_lst_loc := ajouter (PRG4, g_lst_loc);
                                                else
                                                        precondition := false;
                                                end if;
                                                if gconsomme (PRG3)
                                                then
                                                        g_lst_loc := ajouter (PRG3, g_lst_loc);
                                                else
                                                        precondition := false;
                                                end if;
                                                while g_lst_loc /= NULL loop
                                                        if not precondition
                                                        then
                                                                gproduire (contenu (g_lst_loc));
                                                        end if;
                                                        g_lst_loc := supprimer (g_lst_loc);
                                                end loop;
                                        end if;
                                        if precondition
                                        then
                                                proc_B_TP1;
                                                etat := processus_B_B_TP2;
                                                gproduire (PRG5);
                                        else
                                                etat := processus_B_B_PP1;
                                        end if;
-- Pas de code genere                                                        B_PP2 (I/Place Processus Simple)
                                when processus_B_B_TP2 =>        -- B_TP2 (I/Transition Processus Conditionnee)
                                        precondition := true;
                                        if precondition
                                        then
                                                g_lst_loc := NULL;
                                                if gconsomme (PRG7)
                                                then
                                                        g_lst_loc := ajouter (PRG7, g_lst_loc);
                                                else
                                                        precondition := false;
                                                end if;
                                                while g_lst_loc /= NULL loop
                                                        if not precondition
                                                        then
                                                                gproduire (contenu (g_lst_loc));
                                                        end if;
                                                        g_lst_loc := supprimer (g_lst_loc);
                                                end loop;
                                        end if;
                                        if precondition
                                        then
                                                proc_B_TP2;
                                                gproduire (PRG6);
                                                etat := processus_B_T_SYNC;
                                        else
                                                etat := processus_B_B_TP2;
                                        end if;
                                when processus_B_B_TP3 =>        -- B_TP3 (I/Transition Processus Conditionnee)
                                        precondition := true;
                                        if precondition
                                        then
                                                g_lst_loc := NULL;
                                                if gconsomme (PRG5)
                                                then
                                                        g_lst_loc := ajouter (PRG5, g_lst_loc);
                                                else
                                                        precondition := false;
                                                end if;
                                                while g_lst_loc /= NULL loop
                                                        if not precondition
                                                        then
                                                                gproduire (contenu (g_lst_loc));
                                                        end if;
```

```
                                                        g_lst_loc := supprimer (g_lst_loc);
                                                end loop;
                                        end if;
                                        if precondition
                                        then
                                                proc_B_TP3;
                                                etat := processus_B_B_TP4;
                                        else
                                                etat := processus_B_B_PP1;
                                        end if;
-- Pas de code genere                                                                        B_PP3 (I/Place Processus Simple)
                                when processus_B_B_TP4 =>        -- B_TP4 (I/Transition Processus Conditionnee)
                                        precondition := true;
                                        if precondition
                                        then
                                                g_lst_loc := NULL;
                                                if gconsomme (PR66)
                                                then
                                                        g_lst_loc := ajouter (PR66, g_lst_loc);
                                                else
                                                        precondition := false;
                                                end if;
                                                while g_lst_loc /= NULL loop
                                                        if not precondition
                                                        then
                                                                gproduire (contenu (g_lst_loc));
                                                        end if;
                                                        g_lst_loc := supprimer (g_lst_loc);
                                                end loop;
                                        end if;
                                        if precondition
                                        then
                                                proc_B_TP4;
                                                etat := processus_B_T_SYNC;
                                        else
                                                etat := processus_B_B_TP4;
                                        end if;
-- Pas de code genere                                                                        B_PP4 (I/Place Processus Simple)
                                when others =>
                                        null; -- pour le compilateur ADA
                        end case;
                end loop;
        end processus_B;
task body processus_A is                                                                     -- I/processus: processus_A
        chx_A_PP1: tab_etat (1..     2) := (1 => processus_A_A_TP3, 2 => processus_A_A_TP1);
        etat : natural;continue :boolean := true;precondition: boolean;reponse: boolean;bidon : boolean;
        cpt : natural;g_lst_loc : ptr_rg.pt_el;
        begin
                accept marq_init (init :in natural) do etat := init;end marq_init;
                while continue loop
                        case etat is
                                when processus_A_T_SYNC =>        -- T_SYNC (I/Transition de Synchronisation Simple)
                                        reponse := false;
                                        while not (reponse) loop
                                                T_SYNC.pret (IPP6, reponse);
                                        end loop;
                                        reponse := false;
                                        while not (reponse) loop
                                                T_SYNC.accuse (IPP6, reponse);
                                        end loop;
                                        etat := processus_A_A_PP1;
                                when processus_A_A_PP1 =>        -- A_PP1 (IPlace Processus Alternative)
                                        etat := tirage (chx_A_PP1);
                                when processus_A_A_TP1 =>        -- A_TP1 (I/Transition Processus Conditionnee)
                                        precondition := true;
                                        if precondition
                                        then
                                                g_lst_loc := NULL;
                                                if gconsomme (PR64)
                                                then
                                                        g_lst_loc := ajouter (PR64, g_lst_loc);
                                                else
                                                        precondition := false;
                                                end if;
                                                if gconsomme (PR63)
                                                then
                                                        g_lst_loc := ajouter (PR63, g_lst_loc);
                                                else
                                                        precondition := false;
                                                end if;
                                                while g_lst_loc /= NULL loop
                                                        if not precondition
                                                        then
                                                                gproduire (contenu (g_lst_loc));
                                                        end if;
                                                        g_lst_loc := supprimer (g_lst_loc);
                                                end loop;
                                        end if;
                                        if precondition
                                        then
                                                proc_A_TP1;
                                                etat := processus_A_A_TP2;
                                                gproduire (PR65);
                                        else
                                                etat := processus_A_A_PP1;
                                        end if;
-- Pas de code genere                                                                        A_PP2 (I/Place Processus Simple)
                                when processus_A_A_TP2 =>        -- A_TP2 (I/Transition Processus Conditionnee)
                                        precondition := true;
                                        if precondition
                                        then
                                                g_lst_loc := NULL;
                                                if gconsomme (PR67)
                                                then
                                                        g_lst_loc := ajouter (PR67, g_lst_loc);
                                                else
                                                        precondition := false;
                                                end if;
                                                while g_lst_loc /= NULL loop
                                                        if not precondition
                                                        then
                                                                gproduire (contenu (g_lst_loc));
                                                        end if;
                                                        g_lst_loc := supprimer (g_lst_loc);
                                                end loop;
                                        end if;
                                        if precondition
                                        then
                                                proc_A_TP2;
                                                gproduire (PR66);
```

```
                        etat := processus_A_T_SYNC;
                    else
                        etat := processus_A_A_TP2;
                    end if;
                when processus_A_A_TP3 =>       -- A_TP3 (I/Transition Processus Conditionnee)
                    precondition := true;
                    if precondition
                    then
                        g_lst_loc := NULL;
                        if gconsomme (PR65)
                        then
                            g_lst_loc := ajouter (PR65, g_lst_loc);
                        else
                            precondition := false;
                        end if;
                        while g_lst_loc /= NULL loop
                            if not precondition
                            then
                                gproduire (contenu (g_lst_loc));
                            end if;
                            g_lst_loc := supprimer (g_lst_loc);
                        end loop;
                    end if;
                    if precondition
                    then
                        proc_A_TP3;
                        etat := processus_A_A_TP4;
                    else
                        etat := processus_A_A_PP1;
                    end if;
-- Pas de code genere                                   A_PP3 (I/Place Processus Simple)
                when processus_A_A_TP4 =>       -- A_TP4 (I/Transition Processus Conditionnee)
                    precondition := true;
                    if precondition
                    then
                        g_lst_loc := NULL;
                        if gconsomme (PR66)
                        then
                            g_lst_loc := ajouter (PR66, g_lst_loc);
                        else
                            precondition := false;
                        end if;
                        while g_lst_loc /= NULL loop
                            if not precondition
                            then
                                gproduire (contenu (g_lst_loc));
                            end if;
                            g_lst_loc := supprimer (g_lst_loc);
                        end loop;
                    end if;
                    if precondition
                    then
                        proc_A_TP4;
                        etat := processus_A_T_SYNC;
                    else
                        etat := processus_A_A_TP4;
                    end if;
-- Pas de code genere                                   A_PP4 (I/Place Processus Simple)
                when others =>
                    null; -- pour le compilateur ADA
```

```
            end case;
        end loop;
    end processus_A;
task body processus_X is                                        -- I/processus processus_X
    type rl_processus_X is (X_PRL);
    package prl_processus_X is new ress_loc (rl_processus_X);
    use prl_processus_X;
    package ptr_rl_processus_X is new geslist (rl_processus_X);
    use ptr_rl_processus_X;
    etat : natural;continue :boolean := true;precondition: boolean;reponse: boolean;bidon : boolean;
    cpt : natural;l_lst_loc: ptr_rl_processus_X.pt_el;g_lst_loc : ptr_rg.pt_el;
    begin
        lproduire (X_PRL);lproduire (X_PRL);lproduire (X_PRL);  -- Marquage initial des ressources locales
        accept marq_init (init :in natural) do etat := init;end marq_init;
        while continue loop
            case etat is
                when processus_X_X_TP2 =>       -- X_TP2 (I/Transition Processus Conditionnee)
                    precondition := true;
                    lconsomme (X_PRL);
                    if precondition
                    then
                        g_lst_loc := NULL;
                        if gconsomme (PR60)
                        then
                            g_lst_loc := ajouter (PR60, g_lst_loc);
                        else
                            precondition := false;
                        end if;
                        while g_lst_loc /= NULL loop
                            if not precondition
                            then
                                gproduire (contenu (g_lst_loc));
                            end if;
                            g_lst_loc := supprimer (g_lst_loc);
                        end loop;
                    end if;
                    if precondition
                    then
                        proc_X_TP2;
                        etat := processus_X_X_TP3;
                    else
                        lproduire (X_PRL);
                        etat := processus_X_X_TP2;
                    end if;
-- Pas de code genere                                   X_PP2 (I/Place Processus Simple)
                when processus_X_X_TP3 =>       -- X_TP3 (I/Transition Processus Simple)
                    proc_X_TP3;
                    gproduire (PR67);
                    etat := processus_X_X_TP4;
--     X_PP3 (I/Place Processus Simple)
                when processus_X_X_TP4 =>       -- X_TP4 (I/Transition Processus Simple)
                    proc_X_TP4;
                    gproduire (PR64);
                    gproduire (PR63);
                    etat := processus_X_X_TP2;
--     X_PP1 (I/Place Processus Simple)
                when others =>
                    null; -- pour le compilateur ADA
            end case;
        end loop;
```

```
    end processus_X;
task body T_SYNC is     -- Code de la tache implementant la transition T_SYNC (I/Transition de Synchronisation Simple)
    maxpps : constant := 2;maxppa : constant := 0;pps : natural := 0;ppa : natural := 0;active : boolean := false;
        begin -- tache transition
            loop
                while not (active) loop
                    delay 0.0; -- provoque une election
                    select
                        accept pret (t_precond :in tplace; reponse :out boolean) do
                            case t_precond is
                                when IPPA =>
                                    ppa := ppa + 1;
                                when others =>
                                    pps := pps + 1;
                            end case;
                            if (pps = maxpps) AND (ppa = maxppa)
                            then
                                active := true;
                            end if;
                            reponse := true;
                        end pret;
                    or
                        accept accuse (t_precond :in tplace; reponse :out boolean) do
                            if t_precond = IPPA
                            then
                                ppa := ppa - 1;
                            end if;
                            reponse := false;
                        end accuse;
                    end select;
                end loop;
                proc_T_SYNC;
                gproduire (PRG0);
                while (ppa > 0) or (pps > 0) loop
                    select
                        accept pret (t_precond :in tplace; reponse :out boolean) do
                            reponse := false;
                        end pret;
                    or
                        accept accuse (t_precond :in tplace; reponse :out boolean) do
                            case t_precond is
                                when IPPA =>
                                    ppa := ppa - 1;
                                when others =>
                                    pps := pps - 1;
                            end case;
                            reponse := true;
                        end accuse;
                    end select;
                end loop;
                active := false;
            end loop;
        end T_SYNC;
    begin -- PARTIR
        aleatoire.init;max_att := 5;
        gproduire (PRG0);               -- Initialisation des ressources globales
        processus_B.marq_init (1);      -- Lancement du I/processus avec son marquage initial
        processus_A.marq_init (1);      -- Lancement du I/processus avec son marquage initial
        processus_X.marq_init (1);      -- Lancement du I/processus avec son marquage initial
    end PARTIR;
```

RAPPORTS MASI

DERNIERS TITRES

1988

RR88/241 EAO et enseignement de l'informatique à l'Université
B. de La Passardière, M.M Poc Paget

RR88/242 L'état de la normalisation de la couche application OSI
G. Pujolle

RR88/243 Multiclass discrete time queueing systems with a product form solution
G. Pujolle

RR88/244 Un environnement logiciel intelligent pour l'apprentissage de l'algorithmique
B. Delforge, F. Madaule

RR88/245 Une extension de Datalog avec les Agrégats
L. Chen

RR88/246 Quels outils informatiques pour réaliser des didacticiels : une étude sur l'Editeur Fonctionnel de DIANE
M.M Poc Paget, C. Albert

RR88/247 Le transputer T414 de INMOS
M.F Le Roch

RR88/248 OCCAM et les transputers
M.F Le Roch

RR88/249 La démarche de réalisation d'un didacticiel dans DEGEL
B. de La Passardière

RR88/250 Quels outils informatiques pour réaliser des didacticiels : une étude sur HYPERCARD
B. de La Passardière

RR88/251 Faut-il préférer les systèmes-auteurs sur Macintosh ?
B. de La Passardière, M.M Poc-Paget

RR88/252 Analyse opérationnelle de réseaux de files d'attente stochastiques
Y. Dallery, X. Cao

RR88/253 Task scheduling over distributed memory machines
Ph. Chrétienne

RR88/254 Relational storage and efficient retrieval of rules in a deductive DBMS
J.P Cheiney, Ch. de Maindreville

RR88/255 Détection de la Terminaison d'un Calcul Distribué Asynchrone avec Déséquencement
P. Blanc

RR88/256 Generation of ADA code from Petri Nets models
B. Cousin, P. Estraillier

RR88/257 Specification and Correctness of Distributed Algorithms by Coloured Petri Nets
M. Rukoz, R. Sandoval