Generation of ADA code from Petri Nets : an application

Bernard COUSIN Pascal ESTRAILLIER Fabrice KORDON

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

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 more 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** (**P**etri **N**et _ **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

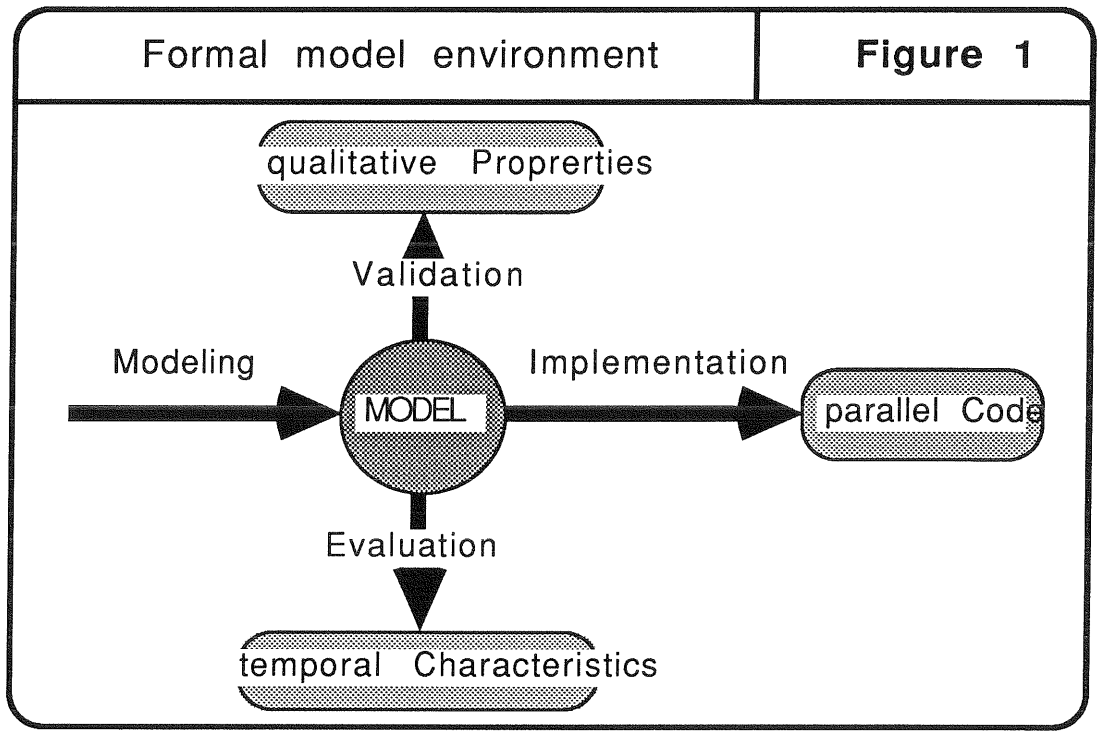
2.1 The phases

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.



2.2 The steps

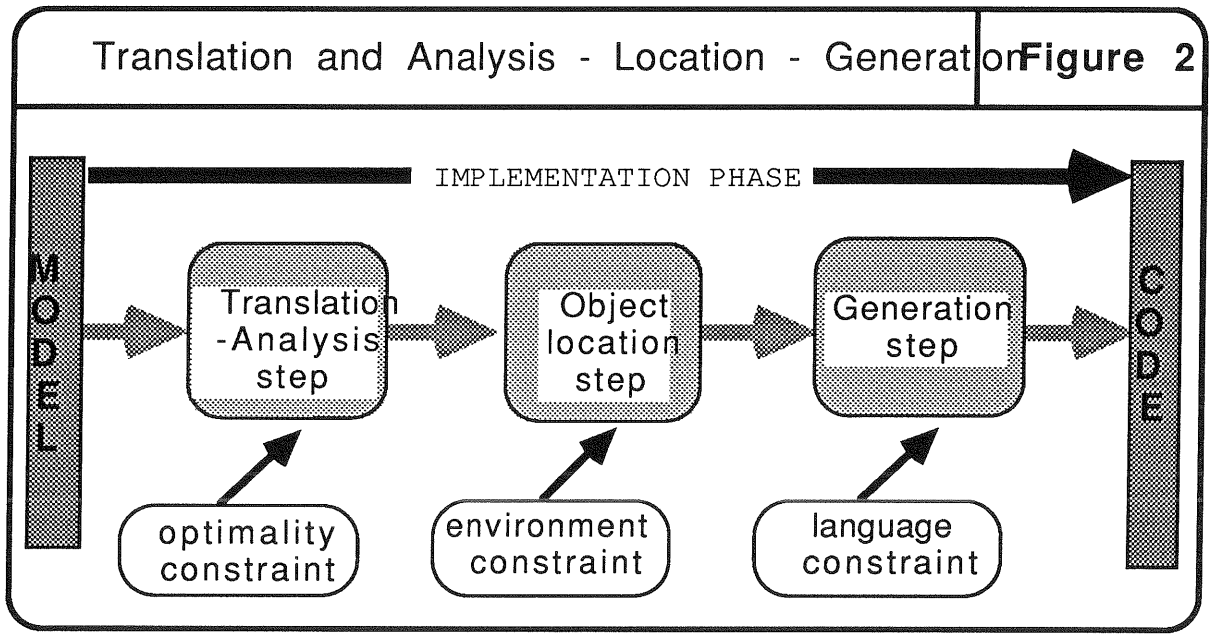
Our methodology organizes the implementation into 3 steps (translation and analysis, location, generation). These steps allow the progressive integration of the constraints respectively of the application, of the environment, and of the programming language, to meet the objectives (quality and performance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



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 of the action.

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,
. the resolution of distributed conflict.

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.

```
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 Simaple)
                                        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;
```

```
121:                                                    g_lst_loc := supprimer (g_lst_loc);
122:                                             end loop;
123:                                      end if;
124:                                      if precondition
125:                                      then
126:                                             proc_B_TP3;
127:                                             etat := processus_B_B_TP4;
128:                                      else
129:                                             etat := processus_B_B_PP1;
130:                                      end if;
131:  -- Pas de code genere                                                    B_PP3 (I/Place Processus Simple)
132:                               when processus_B_B_TP4 =>       -- B_TP4 (I/Transition Processus Conditionnee)
133:                                      precondition := true;
134:                                      if precondition
135:                                      then
136:                                             g_lst_loc := NULL;
137:                                             if gconsomme (PRG6)
138:                                             then
139:                                                    g_lst_loc := ajouter (PRG6, g_lst_loc);
140:                                             else
141:                                                    precondition := false;
142:                                             end if;
143:                                             while g_lst_loc /= NULL loop
144:                                                    if not precondition
145:                                                    then
146:                                                           gproduire (contenu (g_lst_loc));
147:                                                    end if;
148:                                                    g_lst_loc := supprimer (g_lst_loc);
149:                                             end loop;
150:                                      end if;
151:                                      if precondition
152:                                      then
153:                                             proc_B_TP4;
154:                                             etat := processus_B_T_SYNC;
155:                                      else
156:                                             etat := processus_B_B_TP4;
157:                                      end if;
158:  -- Pas de code genere                                                          B_PP4 (I/Place Processus Simple)
159:                               when others =>
160:                                      null; -- pour le compilateur ADA
161:                        end case;
162:                 end loop;
163:          end processus_B;
164:  task body processus_A is                                                          -- I/processus: processus_A
165:          chx_A_PP1: tab_etat (1..     2) := (1 => processus_A_A_TP3, 2 => processus_A_A_TP1);
166:          etat : natural;continue :boolean := true;precondition: boolean;reponse: boolean;bidon : boolean;
167:          cpt : natural;g_lst_loc : ptr_rg.pt_el;
168:          begin
169:                 accept marq_init (init :in natural) do etat := init;end marq_init;
170:                 while continue loop
171:                        case etat is
172:                               when processus_A_T_SYNC =>      -- T_SYNC (I/Transition de Synchronisation Simple)
173:                                      reponse := false;
174:                                      while not (reponse) loop
175:                                             T_SYNC.pret (IPPS, reponse);
176:                                      end loop;
177:                                      reponse := false;
178:                                      while not (reponse) loop
179:                                             T_SYNC.accuse (IPPS, reponse);
180:                                      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 (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_A_TP1;
                                etat := processus_A_A_TP2;
                                gproduire (PRG5);
                        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 (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_A_TP2;
                                gproduire (PRG6);
```

```
241:                                    etat := processus_A_T_SYNC;
242:                             else
243:                                    etat := processus_A_A_TP2;
244:                             end if;
245:                      when processus_A_A_TP3 =>       -- A_TP3 (I/Transition Processus Conditionnee)
246:                             precondition := true;
247:                             if precondition
248:                             then
249:                                    g_lst_loc := NULL;
250:                                    if gconsomme (PRG5)
251:                                    then
252:                                           g_lst_loc := ajouter (PRG5, g_lst_loc);
253:                                    else
254:                                           precondition := false;
255:                                    end if;
256:                                    while g_lst_loc /= NULL loop
257:                                           if not precondition
258:                                           then
259:                                                  gproduire (contenu (g_lst_loc));
260:                                           end if;
261:                                           g_lst_loc := supprimer (g_lst_loc);
262:                                    end loop;
263:                             end if;
264:                             if precondition
265:                             then
266:                                    proc_A_TP3;
267:                                    etat := processus_A_A_TP4;
268:                             else
269:                                    etat := processus_A_A_PP1;
270:                             end if;
271:  -- Pas de code genere                                          A_PP3 (I/Place Processus Simple)
272:                      when processus_A_A_TP4 =>       -- A_TP4 (I/Transition Processus Conditionnee)
273:                             precondition := true;
274:                             if precondition
275:                             then
276:                                    g_lst_loc := NULL;
277:                                    if gconsomme (PRG6)
278:                                    then
279:                                           g_lst_loc := ajouter (PRG6, g_lst_loc);
280:                                    else
281:                                           precondition := false;
282:                                    end if;
283:                                    while g_lst_loc /= NULL loop
284:                                           if not precondition
285:                                           then
286:                                                  gproduire (contenu (g_lst_loc));
287:                                           end if;
288:                                           g_lst_loc := supprimer (g_lst_loc);
289:                                    end loop;
290:                             end if;
291:                             if precondition
292:                             then
293:                                    proc_A_TP4;
294:                                    etat := processus_A_T_SYNC;
295:                             else
296:                                    etat := processus_A_A_TP4;
297:                             end if;
298:  -- Pas de code genere                                          A_PP4 (I/Place Processus Simple)
299:                      when others =>
300:                             null; -- pour le compilateur ADA
```

```
301:                        end case;
302:                end loop;
303:        end processus_A;
304: task body processus_X is                                                    -- I/processus: processus_X
305:        type rl_processus_X is (X_PRL);
306:        package prl_processus_X is new ress_loc (rl_processus_X);
307:        use prl_processus_X;
308:        package ptr_rl_processus_X is new geslist (rl_processus_X);
309:        use ptr_rl_processus_X;
310:        etat : natural;continue :boolean := true;precondition: boolean;reponse: boolean;bidon : boolean;
311:        cpt : natural;l_lst_loc: ptr_rl_processus_X.pt_el;g_lst_loc : ptr_rg.pt_el;
312:        begin
313:               lproduire (X_PRL);lproduire (X_PRL);lproduire (X_PRL);  -- Marquage initial des ressources locales
314:               accept marq_init (init :in natural) do etat := init;end marq_init;
315:               while continue loop
316:                      case etat is
317:                             when processus_X_X_TP2 =>       -- X_TP2 (I/Transition Processus Conditionnee)
318:                                    precondition := true;
319:                                    lconsomme (X_PRL);
320:                                    if precondition
321:                                    then
322:                                           g_lst_loc := NULL;
323:                                           if gconsomme (PRG0)
324:                                           then
325:                                                  g_lst_loc := ajouter (PRG0, g_lst_loc);
326:                                           else
327:                                                  precondition := false;
328:                                           end if;
329:                                           while g_lst_loc /= NULL loop
330:                                                  if not precondition
331:                                                  then
332:                                                         gproduire (contenu (g_lst_loc));
333:                                                  end if;
334:                                                  g_lst_loc := supprimer (g_lst_loc);
335:                                           end loop;
336:                                    end if;
337:                                    if precondition
338:                                    then
339:                                           proc_X_TP2;
340:                                           etat := processus_X_X_TP3;
341:                                    else
342:                                           lproduire (X_PRL);
343:                                           etat := processus_X_X_TP2;
344:                                    end if;
345:  -- Pas de code genere                                                                X_PP2 (I/Place Processus Simple)
346:                             when processus_X_X_TP3 =>       -- X_TP3 (I/Transition Processus Simple)
347:                                    proc_X_TP3;
348:                                    gproduire (PRG7);
349:                                    etat := processus_X_X_TP4;
350:  --     X_PP3 (I/Place Processus Simple)
351:                             when processus_X_X_TP4 =>       -- X_TP4 (I/Transition Processus Simple)
352:                                    proc_X_TP4;
353:                                    gproduire (PRG4);
354:                                    gproduire (PRG3);
355:                                    etat := processus_X_X_TP2;
356:  --     X_PP1 (I/Place Processus Simple)
357:                             when others =>
358:                                    null; -- pour le compilateur ADA
359:                      end case;
360:               end loop;
```

```
361:         end processus_X;
362:  task body T_SYNC is     -- Code de la tache implementant la transition T_SYNC (I/Transition de Synchronisation Simple)
363:         maxpps : constant := 2;maxppa : constant := 0;pps : natural := 0;ppa : natural := 0;active : boolean := false;
364:                begin -- tache transition
365:                       loop
366:                              while not (active) loop
367:                                     delay 0.0; -- provoque une election
368:                                     select
369:                                            accept pret (t_precond :in tplace; reponse :out boolean) do
370:                                             case t_precond is
371:                                                    when IPPA =>
372:                                                           ppa := ppa + 1;
373:                                                    when others =>
374:                                                           pps := pps + 1;
375:                                             end case;
376:                                             if (pps = maxpps) AND (ppa = maxppa)
377:                                             then
378:                                                    active := true;
379:                                             end if;
380:                                             reponse := true;
381:                                     end pret;
382:                                or
383:                                     accept accuse (t_precond :in tplace; reponse :out boolean) do
384:                                             if t_precond = IPPA
385:                                             then
386:                                                    ppa := ppa - 1;
387:                                             end if;
388:                                             reponse := false;
389:                                     end accuse;
390:                              end select;
391:                       end loop;
392:                       proc_T_SYNC;
393:                       gproduire (PRG0);
394:                       while (ppa > 0) or (pps > 0) loop
395:                              select
396:                                     accept pret (t_precond :in tplace; reponse :out boolean) do
397:                                             reponse := false;
398:                                     end pret;
399:                              or
400:                                     accept accuse (t_precond :in tplace; reponse :out boolean) do
401:                                             case t_precond is
402:                                                    when IPPA =>
403:                                                           ppa := ppa - 1;
404:                                                    when others =>
405:                                                           pps := pps - 1;
406:                                             end case;
407:                                             reponse := true;
408:                                     end accuse;
409:                              end select;
410:                       end loop;
411:                       active := false;
412:                end loop;
413:         end T_SYNC;
414:  begin -- PARTIR
415:         aleatoire.init;max_att := 5;
416:         gproduire (PRG0);                    -- Initialisation des ressources globales
417:         processus_B.marq_init (1);           -- Lancement du I/processus avec son marquage initial
418:         processus_A.marq_init (1);           -- Lancement du I/processus avec son marquage initial
419:         processus_X.marq_init (1);           -- Lancement du I/processus avec son marquage initial
420:  end PARTIR;
```