ISSN 0280–5316
ISRN LUTFD2/TFRT--5482--SE

Implementation of a toolbox for hybrid Petri nets in G2

Michel Colombo

Department of Automatic Control
Lund Institute of Technology
September 1993

Abstract

This report deals with the implementation of a toolbox for hybrid Petri nets (HPN) in G2. While developing the program improvements of the CCPN speed set calculation algorithm have been carried out. They have permitted to save calculations and to make it possible to solve effective conflicts by sharing as well as by priority. The modifications are detailed and the resulting HPN algorithm is given. In order to make every readers able to understand the report the basical notions on discrete Petri nets, continuous Petri nets with constant speed and Hybrid Petri nets are reviewed in annexes.

Résumè

Ce rapport presente la boite a outil pour les réseaux de Petri hybride qui a étè implementées dans G2. L'algorithme pour calculer les vitesses des transitions continues a étè modifié pour permettre de resoudre les conflits par partage du flot ou par priorité. Finalement deux methodes pour programmer les réseaux de Petri hybride dans G2 ont étè experimentées.

Contents

1. Introduction

Plant engineers and operators operating and monitoring a discrete or continuous process are often faced with the problem of making the correct decision about how the process should be run in order to optimize production. Several possibilities exist for assisting the users with these decisions. One possibility is to construct a model of the process which then is simulated in order to obtain the consequence of a decision. The problem with this approach is that it is often difficult to find all the rules and equations monitoring a process. This is specially true for discrete processes such as, e.g., manufacturing processes.
Around 1960 Carl Adam Petri introduced Petri nets for modeling relationships between conditions and events (Petri, 1962). Since then extensive research has been made in the area and Petri nets are now one of the most widely used methods for modeling various discrete phenomena such as , e.g., states of discrete devices, machine production lines, etc.
In 1987 continuous Petri nets were defined by Alla and David (David and Alla, 1992) at the Laboratory of Automatic Control of Grenoble (LAG). The main motivation for continuous Petri nets was to approximate timed discrete Petri nets when the number of tokens were large. Several different versions of continuous Petri nets have been developed, for example, continuous Petri nets with constant speeds, continuous Petri nets with variable speeds (VCPN), and asymptotically continuous Petri nets.
The second motivation for continuous Petri nets is the possibility to model continuous phenomena such as, e.g., flows. In this area little work has been done. None of the different continuous Petri net models have been derived for this purpose.
Hybrid Petri nets give the possibility to combine continuous elements and discrete elements in the same model. This is specially appealing in light of the large research interest on system that combine continuous and discrete phenomena.
A user-friendly computer environment for Petri nets requires a powerful graphical interface. G2 from Gensym Corp (Gensym Corporation, 1992) was originally developed as a real-time expert system but has gradually developed towards a general object oriented programming environment with a powerful graphical interface. In a previous project a toolbox for coloured Petri nets was implemented in G2 (Sarraut, 1990). The goal of this was to implement a toolbox for hybrid Petri nets were the nets could be easily created and simulated. Since the main motivation for the work was to be able to model combined continuous-discrete phenomena and none of the existing continuous Petri nets models were focused on continuous phenomena the correct solution would have been to define a new model for the continuous part of the hybrid Petri net that was better suited, i.e., more general, for modeling continuous phenomena. Due to time limitations this was, however, considered to be outside the scope of this thesis. Instead continuous Petri nets with constant speeds has been chosen for the continuous part of the hybrid Petri net toolbox.

Discrete Petri nets are briefly presented in Appendix A. Continuous Petri nets with constants speeds and hybrid Petri Nets are discussed in Appendix B. The G2 implementation of the toolbox is presented in Chapter 2. An improved algorithm for the calculation of speed states in hybrid nets is described in Chapter 3. The complete algorithm is found in Appendix G. Appendix F consists of a user's guide for the toolbox. Finally some problems encountered with continuous Petri nets with constants speeds are presented in Appendix E.

2. Implementation of HPNs in G2

This section gives an overview of G2 and presents two different ways to implement PNs with G2. Both these ways have been used in the project.

G2 overview

G2 from Gensym Corp, originally developed as a real-time expert system, has gradually evolved into a very powerful object oriented programming environment with strong graphical features. G2 is written in LISP that is automatically translated into C. All user programming , however, is done in G2's built in programming language using either rules or procedures. In order to make it easy to capture the knowledge of experts, G2 expresses it in terms of object classes. Therefore G2 includes many ways to define, reason about, and manipulate objects.
G2 can be summarized into six main parts : the knowledge-base, the real time inference engine, a procedural language, a simulator, a development environment, an end user's interface and external interfaces.

The knowledge-base : In G2 the knowledge base is constructed by creating subclasses of built-in classes. At the top of the hierarchy is *item* the most unrestricted class already exiting. Therefore each pieces of knowledge is grouped into a subclass of item called *object class.* From this object class, instances can be created. All individual instances (called objects) have the same graphical representation (called *icon*) and have the same attributes which have been specified in the definition of the object class. Object classes are hierarchically organized. A class inherits the attributes of its 'father' or superclass and it transmits its own attributes to its 'children. Consequently it is very important to define attributes at the correct level in the hierarchy. G2 also provides two other links between objects : *Connections* and *Relations.* Connections are arrows that graphically connect objects. Relations (such as $a < b$) establish a relation between objects but do not have a graphical visualization. Connections and relations are important elements of the knowledge-base because the expert system can reason about them.

The inference engine : The inference engine is responsible for the application of the rules written by the developer. The rules can be applied to an object (*specific rules*) or a class of objects and their children (*generic rules*). During run time these rules may be invoked by backward or forward chainings. G2 allows to group rules by *focus objects* or by *user categories* and to invoke the whole groups of rules. It is also possible to fix a *scan interval* for a rule and when the scan interval is elapsed to automatically invoke it. The advantage of using rules is that you do not have to think about the order in which the rules will be invoked. This can however be a problem sometimes, therefore this can be solved by using priorities or if this is not enough by using procedures instead.

Procedural engine : Procedures are written in a Pascal style procedural language. They are started by rules, by other procedures or by action-buttons. Similarly to rules a procedure can operate on numeric expressions, attributes or objects and can be specific or generic.

Development environment : Textual information in rules, procedures and other items are written with a structured natural language editor. After typing a word the next syntactically correct entries are proposed in a menu and the user can either click on words in the menu or type them in. To create icons an icon editor is provided. Each item in G2 is associated with a graphical representation and has a menu by which it can be manipulated. For example you can duplicate objects by using the *clone* menu choice of the object itself or the *create instance* menu choice of its class definition.

Simulator : The built-in simulator calculates simulated values which are used to test the knowledge-base or to give information in parallel during on-line operations.

End user's interface : To customize the knowledge-base numerous already built-in classes exist. They can be classified as follows :
Interaction objects : They allow you to change a variable from one value to another (for example to change the variable *run* from *off* to *on* by clicking on the right button).
Action buttons : By clicking on it an action or a procedure is started (for example : *change the back icon color to green*).
Sliders, type-in boxes : They allow changing the value of a variable dynamically.
Displays : Graphs, meters, dials and charts are different way to present the values of variables.

The developer can implement animations by using instructions such as rotate, move, change colors, and create or delete objects dynamicly. It is also possible to attach new menu choices to objects. Finally the developer can restrict the accesses to the knowledge-base for various user categories.

External interfaces : They are sold separately from G2 and can give the access to some external data servers (such as external data files, external simulators, C and Fortran functions).

G2 provides a powerful graphical interface and easy ways to manipulate and reason about objects. It is particularly well fitted to implement programs such as PNs which need graphical representations.

The HPN tool box implementation

To implement PNs in G2 the knowledge-base has been partitioned into two parts : the first part contains the class hierarchy describing the structural properties of PNs and the second consists of rules and procedures to govern the evolution.

The HPN class hierarchy :
The class hierarchy of Figure 1 has been implemented in G2. The class Petri-net-objects is used as the root class for of all petri net objects. It is split into three classes:

Marker class : The marker class specifies only the appearance of tokens (a black dot).

Place class : At this level the initial marking (of the place), the marking, and the available marking (non reserved marking) are defined as common characteristics of c and d-places. The Place class has two child classes: c-place and d-place. They correspond to continuous and discrete places and they are mainly used to give them different icons (double circle for c-places and simple circle for d-places).

Transition class : The class is used to define the attribute "priority" as a common attribute to c-transitions and d-transitions. It is used to solve effective conflicts by priority. The C-transition class defines a maximal speed (denoted 'Max speed'), an instantaneous speed (denoted 'old speed') and a 'flow part' as attributes of c-transitions. The 'Flow part' is required when the effective conflict is solved by sharing. There is also a specifical icon for c-transitions (two boxes , one inside the other). The D-transition class details the attributes of d-transitions : an icon (a simple box), a timing and one boolean variable (fireable). The two subclasses d-delay-transition and d-event-transition have the same attributes but their icons differ slightly (a simple box with a diagonal bar for d-event-transitions). D-delay-transitions correspond to T-timed-transitions whereas d-event-transitions correspond to synchronized transitions.

Another class of petri net objects has been created but does not appear in Figure 1 because it is rooted from another part of the G2 class hierarchy. It is called Petri-net-connection and it defines the attributes of arcs interconnecting places and transitions. Its attribute is the 'weight' of the arc.

Due to from the numerous capabilities of G2 two different ways have been used to simulate HPNs. The first one uses the G2 simulator whereas

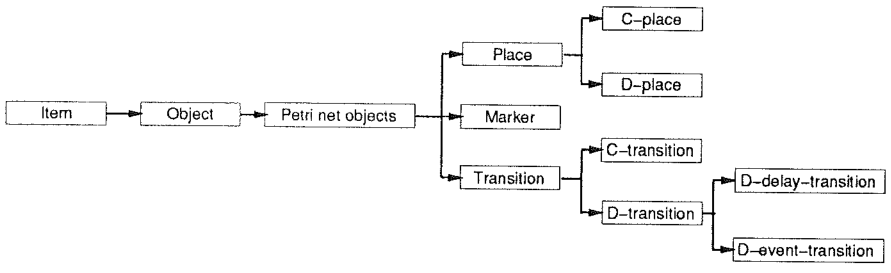


Figure 1. HPN class hierarchy

the second uses G2 procedures.

Implementation of CCPNs using the G2 simulator

The G2 simulator provides two facilities for simulation : *simulation formulas* and *simulation models*. Both are periodically updated and the user can set the update interval (called *time increment*). To govern the CCPN evolution *generic simulation formulas* have been used. Similarly to rules, the simulation formulas are named *generic* because they apply to a whole class of objects. In the below example the formulas applies to every tank.

```
SIMULATION FORMULA
The level of any tank = the water-volume of any tank/the
cross-section-area of any tank
```

It is important to notice that the following formulas are incomplete in order to simulate CCPNs correctly in all cases. Some new generic simulation formulas need to be introduced to solve the encountered problems (they are detailed after the formulas). However they form the framework of what will be, in my opinion, the shortest way to simulate CCPNs in G2 if the user is only interested in a version of CCPNs running in real time.

FORMULAS[1]

```
Formula 1 : The input-flow of any places P = The sum over each
                c-transitions T connected at an input of P of
                the speed of T

Formula 2 : The output-flow of any places P = If mi > 0 then
                 The sum over each c-transition T connected
                 at an  output of P of the maximal speed.
                                            else
                 Max(The sum over each c-transition connected
                 at an output of P of the maximal speed , The
                 input-flow of P)

Formula 3 : The speed of any c-transitions T = Minimum over
                 each c-places P connected at an input of T
                 of (the output-flow of P * the maximal speed
                 of T) /(The sum over each c-transitions
                 connected at an output of P of the maximal
                 speed)
Formula 4 : mi(t+delta) = Max(mi(t)+(input-flow - output-flow)
                 *delta, 0)
```

1 The real syntax of G2 generic simulation formulas has not been respected in order to make formulas clearer to read.
For the notations refer to Appendix H.

Remarks

: Max is a function returning the maximum of the values with the bracket.
: Effective conflicts are solved by sharing proportional to maximal speed (Formula 3).
: Delta is the time increment of the Generic simulation formulas.

The G2 simulator evaluates the formulas periodically. Therefore the problem illustrated in Figure 2, Example *a* arises. If a c-event occurs between two simulation time instants it results in a negative marking for the place responsible of the c-event (point A1) and an excess of markings for downstream places. In Example *a* the c-event occurred at $t = 2.5s$ and Formula 4 is executed at $t = 3s$. Therefore the marking of P2 is evaluated to 3 (m_2(t=3)=m_2(t=2)+1) instead of 2.5 and m_1 has the value -0.5. The problem of the negative marking is easily solved by comparing m_1 to 0 (Formula 4), but the one of the excess of markings remains. In the example it is possible to propose a solution : Point A1 can be calculated (Formula 4 without comparison to 0), therefore the excess of markings deposited in P2 is known and we only have to subtract this quantity to the marking of P2. However this approach seems to be insufficient since the situation can be much more complicated such as two c-events involving the same transition and both occurring as the simulation time increment has not elapsed yet (Figure 2, Example b).
Two other difficulties can be noticed :

- In case of effective conflicts Formula 3 solves it by sharing proportional to maximal speeds. Solving the conflict with priority is substantially more difficult to implement using this approach.
- We do not know the exact time when the c-event occurs. Therefore it is impossible to precisely graph the marking evolution.

The main reason why this way has not been continued is : it is impossible to 'jump' from a c-event to the next, we have to wait for the functioning interval being completed. An improved behavior is allowed by the second possibility to implement PNs in G2.

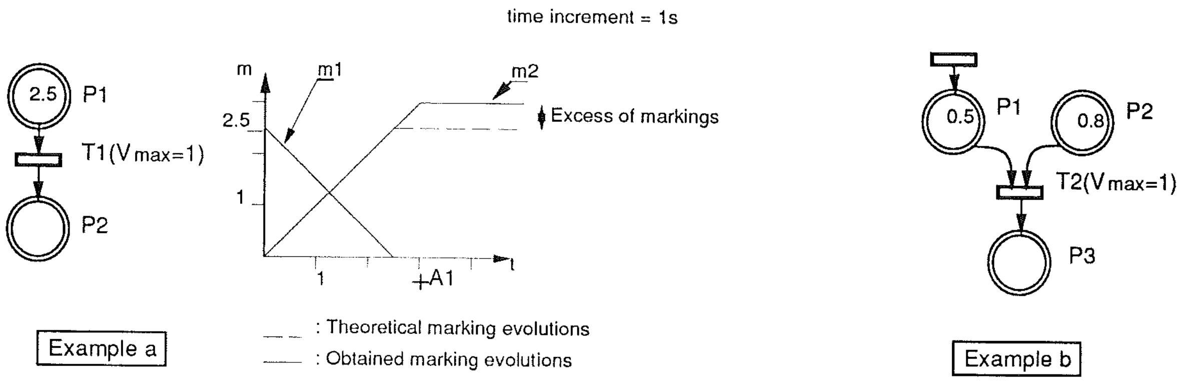


Figure 2. Difficulties arising when using the G2 simulator

Implementation of HPNs using procedures

G2 contains four kinds of procedures :

- System procedures : They consist of already built-in procedures for performing system operations e.g. accessing files and directories.
- Remote procedures : They are Fortran or C procedures called from within G2.
- User defined procedures : User defined procedures look like 'normal' Pascal procedures in which statements such as if-then, for, repeat,... are available. It is also possible to perform actions involving numeric expressions as well as actions that operate upon every object of a certain class in the knowledge-base. Therefore the statement in the next example is available.

  ```
  If the fireable of T is true then
     For each input d-places P connected at an input of T do
        Remove the marker from ...
  ```

  The HPN program mainly uses this type of procedures.
- Simulation procedures : They are similar to the above type, but they are used in connection with simulation models.

Procedures have been preferred to implement HPNs because they permit to control the execution order of operations. The HPN program is organized into five main procedures indicated in the following algorithm[2].

The HPN algorithm :

step 1 : HPN-initialization.
step 2 : *Time* $\longleftarrow$ *next-time-event.*
step 3 : Update-markings.
step 4 : Update-event.
step 5 : New-IB-state.
step 6 : Next-event-time.
step 7 : (Wait for *next-event-time - the current time*).
step 8 : *If stable state not reached then go to step 2.*

HPN-initialization : This procedure initiates all variables that need to have an initial value and performs all operations required initially before starting the procedures which really govern the PN evolution.

Update-markings : The procedure determines the value of the continuous marking in each c-place. To carry out this task the fundamental relation and the previous value of the marking at $t - dt$ (= the value of *Time* before step 2) are used.

Update-event : The procedure fires all transitions that need to be fired at *Time*. It also approximates the available marking of the c-place responsible for the c-event.

New-IB-state : The speed of each c-transition resulting from the new

2 A complete algorithm is provided in Appendix G

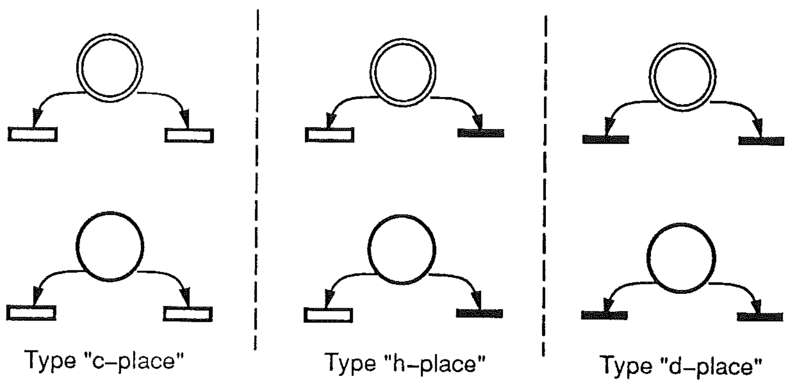


Figure 3. The 'type' of places

marking state and the reservation of marking is calculated by New-IB-state (IB-state is defined in Appendix C).

Next-event-time : In this procedure the instants when the next c-event, d-event, and h-event should happen are calculated and the minimum over these three instants is assigned to *next-event-time* (c-event, d-event, h-event have been defined in Appendix A, Appendix B and Appendix C).

From this algorithm can be realized a real-time version of HPNs by adding a wait statement at step 7. The delay of the wait statement is equal to *next-event-time - current time*, where current time is the time elapsed since the beginning of the application. Therefore the synchronization between the evolution of the HPN and the current time is quite accurate because the calculation time from step 2 to step 5 is taken into account. For example if at current time $= 20s$ we are at step 2 and at current time $= 22s$ we end step 5, in which the next-event-time has been determined to 30, hence $8s$ are assigned to the wait statement (the calculation time has been subtracted). However the synchronization is of course wrong if the theoretical time elapsed between two successive events is shorter than the delay required to complete the calculations from step 2 to step 5. In this case the delay of the wait statement is fixed to 0 and a warning message alerts the user.
To make the program as fast as possible a new attribute has been defined for each place to save calculations during run time. It is called the 'type' of the place and its values can be "d-place", "c-place", and "h-place"[3]. It is determined as follow : if the output transitions of a place are solely d-transitions then the 'type' of the place is "d-place", if they are only composed of c-transitions the 'type' is "c-place" and otherwise the 'type' is "h-place" (Figure 3). Using this attribute it is possible to selectively apply the previous procedures. For example the procedure New-speed-state will be started only if one of the following events has been detected

3 Be careful to not confuse the 'type' of a place and the class of a place. In the following text we use '...' to refer to the name of the attribute and "..." to refer to its values.

during step 2 or 3 :

- A marking reaches the value zero in places of 'type' "c-place" or "h-place".
- Tokens are deposited in places of 'type' "c-place' or "h-place".

Therefore the firing of a d-transition involving exclusively "d-places" in the HPN does not provoke any new calculations of speeds. The 'type' of a place is also used in many other occasions in the body of each procedures. We can notice the two following usages :

- When an effective conflict occurs it is more interesting to know the 'type' of the place rather than its class because it is directly related to the method to solve it. A conflict between d-transitions is exclusively solved by priority whereas between c-transitions it can also be solved by sharing. Further if a "h-place" is involved then it is solved by an other method (explained in the last part of the next chapter).
- In the body of Next-event-time it is unnecessary to look for a c-event in places of type "d-place" even if it is a c-place and similarly looking for a h-event or a d-event in "c-place".

Another way to save calculations has been adopted in the determination of speeds, it is detailed in the following chapter.

3. The implemented algorithm

In the following section the ideas for improving the speed set calculation algorithm are only presented in the context of CCPNs. The second part of this chapter briefly presents ways to generalize them to also cover HPNs.

Modification of the speed set calculation algorithm

In the Appendix B it was pointed out that the algorithm presented (David and Alla, 1992) has the following potential problems :

- It seems difficult to solve effective conflicts by sharing.
- The order in which transitions are classified influences the number of steps needed to reach the speed state.
- Extra iterations are required to check that the speed state is really obtained.

The revised algorithm has been deduced from the existing. Therefore both algorithms exploit the same basic formula introduced by David and Alla (1992).

Formula R1 [4]:

$$v'_j = \begin{cases} V_{jmax} & \text{if } T_j \text{ is strongly enabled} \\ Min(V_{jmax}, b_i + v_j) & \text{otherwise} \\ \text{with i such as } P_i \in {}^{\circ}T_j \text{ and } m_i(t) = 0 & \end{cases}$$

v_j=the speed before applying R1
v_j'=the speed after having applied R1

Remark 1
: Formula R1 is the mathematical expression of the definitions of strongly and weakly enabled[5].

In order to improve the algorithm three modifications have been performed, each of them solving one point. To not confuse the reader they are separately introduced. The last section is focused on a new problem encountered.

First modification

This change has been realized to allow the user to solve conflicts by priority or by sharing. It is made possible by initially specifying a place order instead of a transition order and by iterating on places rather than on transitions. To visualize the difference between the algorithms they are

4 Refer to Appendix H for the notations.

5 Refer to Appendix B for the definitions.

written one on the side of the other. Algorithm 1 is directly inspired by the David and Alla's algorithm. Algorithm 2 is a modification of the first one.

Algorithm 1[a]

```
Begin
Initialization :
Classify transitions
For j=1 to N do[b]
    $v_j \leftarrow 0$;
    $v_j' \leftarrow 0$;
End for
Body :
Repeat until Stop=true
    Stop←true;
    For j=0 to N do
        For each place $P_i \in {}^\circ T_j$ do
            Inflow of $P_i \leftarrow 0$;
            For each transition $T_k \in {}^\circ P_i$ do
                Inflow of $P_i \leftarrow$ Inflow of $P_i + v_k$;
            End for
            Outflow of $P_i \leftarrow 0$;
            For each transition $T_k \in P_i^\circ$ do
                Outflow of $P_i \leftarrow$ Outflow of $P_i + v_k$;
            End for
            $b_i \leftarrow$ Inflow of $P_i$ - Outflow of $P_i$;
            Calculation of $v_j'$ by R1;
            If $v_j' \neq v_j$ then
                $v_j \leftarrow v_j'$;
                Stop←false;
            End if
        End for
    End for
End repeat
End
```

a Refer to Appendix H for the used notations.

b N=the number of transitions in the CCPN.

Algorithm 2

```
Begin
Initialization :
Classify places;
For j=1 to N do
    If $T_j$ has no input places then
        $v_j \leftarrow V_{jmax}$;
        $v_j' \leftarrow V_{jmax}$;
    Else
        $v_j \leftarrow 0$;
        $v_j' \leftarrow 0$;
    End if-else
End for
Body :
Repeat until Stop=true
    Stop←true;
    For i=1 to M do[a]
        Inflow of $P_i \leftarrow 0$;
        For each transition $T_j \in {}^\circ P_i$ do
            Inflow of $P_i \leftarrow$ Inflow of $P_i + v_j$;
        End for
        Outflow of $P_i \leftarrow 0$;
        For each transition $T_j \in P_i^\circ$ do
            Calculation of $v_j'$ by R1;
            Outflow of $P_i \leftarrow$ Outflow of $P_i + v_j'$;
        End for
        If Inflow of $P_i <$ Outflow of $P_i$
        and $P_i$ no marked then
            Call Solve-effective-conflict;
        End if
        For each transition $T_j \in P_i^\circ$ do
            If $v_j' \neq v_j$ then
                $v_j \leftarrow v_j'$;
                Stop←false;
            End if
        End for
    End for
End repeat
End
```

a M=the number of places in the CCPN.

The procedure Solve-effective-conflict has not been detailed because it is application dependent. The user can implement whatever strategy he wants for solving effective conflict. The necessary information for this i.e., the inflow of place P and the maximal possible speed for each output transition, is available at this stage. An example of a procedure for solving the effective conflict by sharing is provided below:

Solve effective conflict

```
Begin For each transition $T_j \in P_i^\circ$ do
    $v_j' \leftarrow \frac{((\text{inflow of } P_i \text{ - real outflow of } P_i)^* \text{ flow part})}{\text{The weight of the arc between } T_j \text{ and } P_i} + v_j$
End for
```

Note :

- *The real outflow of P_i is the sum over the output transitions of v_j.*
- *The flow part is the proportion of the inflow that goes through T_j.*

The idea leading to Algorithm 2 is as follows. To detect an effective conflict we have to evaluate the quantity of markings entering and the maximum quantity able to leave the place at the same instant. To determine the last entity the relation R1 is applied to each output transitions as if the transition had the highest priority. In other words R1 returns the maximal flow able to pass through the transition but the flow is not assigned to it (this explains why a variable v_j' is required : it is used to record the calculated value of the maximal possible speed). Hence the maximal possible output flow is obtained by summing the value of v_j' over each output transitions. This operation is much more difficult to carry out with Algorithm 1 because the order in which transitions are classified does not normally give any information about the structural properties of the PN. The transitions that are part of the same structural conflict are not grouped together.
If, in both algorithms, places and transitions are randomly arranged Algorithm 2 generally needs more calculations because during a loop over all the places the same transition may be involved as many times as its number of input places. This problem is corrected by the second modification.

Second modification

The second modification aims to efficiently order places or transitions to reduce the number of calculations needed. The adopted idea is to "follow the flow". To picture this idea let us assume that each c-place is a tank, the continuous marking is a water level, transitions are valves and the associated speed is a flow. Therefore updating the speed of a transition corresponds to trying to change the flow of the valve. Four intuitive concepts can be deduced :

- Initially we cannot modify the flow of a transition if there is no water upstream, except for source transitions.
- While flowing only the valves on the flow have to be taken into consideration.
- Changing the input flow of a tank that already contains water does not modify the flow of the output transitions.

- When the flow of a valve cannot be revised, the flows of downstream valves are not affected.

Applying these concepts to CCPNs, they respectively become :

- Initially only the output transitions of marked places and source transitions have to be taken in consideration .
- If $v_j \neq v_j$'[6] then the output transitions of the non marked output places have to be examined.
- If $v_j = v_j$' then do nothing.

These three points can easily be implemented in the algorithm using, for example c-pointers (each place points to its output transitions and each transition points to its output places). In G2 it is even easier because it is directly possible to reason about the graphical connections linking places and transitions. Notice that there is not only a unique transition order even when the speed of transitions is calculated by an algorithm incorporating the last proposed modification. Several flows can be flowing simultaneously in the CCPN, therefore many solutions are available. We can decide for example to carry out calculations on a flow until it is dried out or to make calculations in parallel.

Rather than providing the reader with a new algorithm it has been preferred to confront the calculation evolution of Algorithm 1 to those of Algorithm 2 using the previously defined rules (Figure 4). In the tables of the figure the calculation order is represented by an arrow. The arrow comes from the speed that has just been calculated and goes to the speed of the transition that has to be examined. The next transition whose speed has to be updated is determined by applying the previously defined rules. Three points can be noted from the application of the modification:

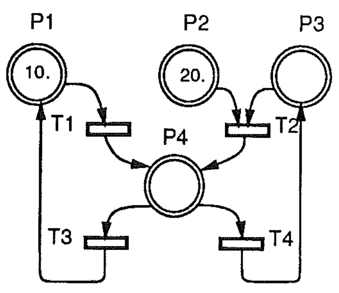
- We do no longer need to initially classify transitions because we replace the static list of transitions by a list dynamically built while running the program.
- The number of saved calculations depends on the structure of the net and the initial markings.
- It seems possible to carry out some calculations simultaneously when two flows are flowing independently of each other. For example the calculations noted A and B in the second table of Figure 4.

The inconvenience of this method is that you do no more have any 'flag' to signal that the speed state is reached. When a flow is stopped by the relation $v = v$' it is difficult to know whether or not there exists an other flow flowing in the CCPN at the same instant. Therefore the third modification can be useful.

Third modification

The purpose of the third modification is to find an alternative way to test if the speed state has been obtained. The idea is to count the number

6 Refer to notations introduced with Formula R1.

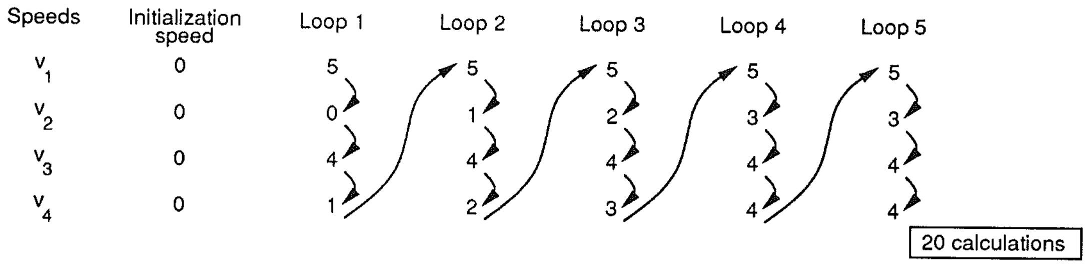


Hypothesis :

$V_{1max} = 5$	$V_{3max} = 4$
$V_{2max} = 3$	$V_{4max} = 6$

T3 has the priority over T4

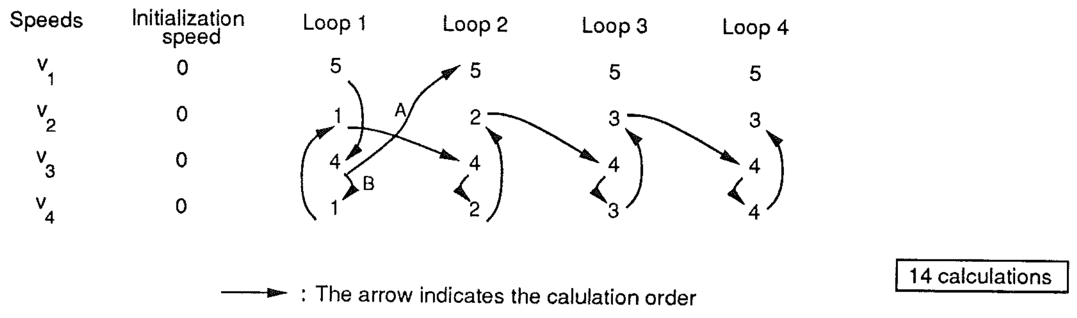
Algorithm 1 (Without the second modification)



Speeds	Initialization speed	Loop 1	Loop 2	Loop 3	Loop 4	Loop 5
V_1	0	5	5	5	5	5
V_2	0	0	1	2	3	3
V_3	0	4	4	4	4	4
V_4	0	1	2	3	4	4

20 calculations

Alghorithm 2 (Applying the second modification)



Speeds	Initialization speed	Loop 1	Loop 2	Loop 3	Loop 4
V_1	0	5	5	5	5
V_2	0	1	2	3	3
V_3	0	4	4	4	4
V_4	0	1	2	3	4

14 calculations

$\longrightarrow$: The arrow indicates the calulation order

NB : -In the second case the speed calculation of T3 is carried out before that of T4 in order to give it the priority.
-In table 2 it has arbitrarily been decided to begin by T1, the choice to begin by T2 leads to one extra iteration .
-Calculations noted A and B in the second table can be carried out simultaneously.

Figure 4. The second amelioration possible of the algorithm

of flows flowing in the CCPN. This is also inspired by the interpretation of the marking in terms of liquid. Initially some places (Case a)[7] and transitions (case b) have flows. In other words they are the sources of the liquid in the PN. While flowing these flows can be duplicated : in Case *c* one flow enters the place and two leave it and in Case *d* the flow reaching the transition is divided into two new flows. A flow can be dried up (it does not have any new flow) when it reaches a transition such that of Case *e* and a place such as illustrated in case *f*. A flow also disappears if condition *g* is met. The last case corresponds to the last point of the

7 Refer to Figure 5 for the cases

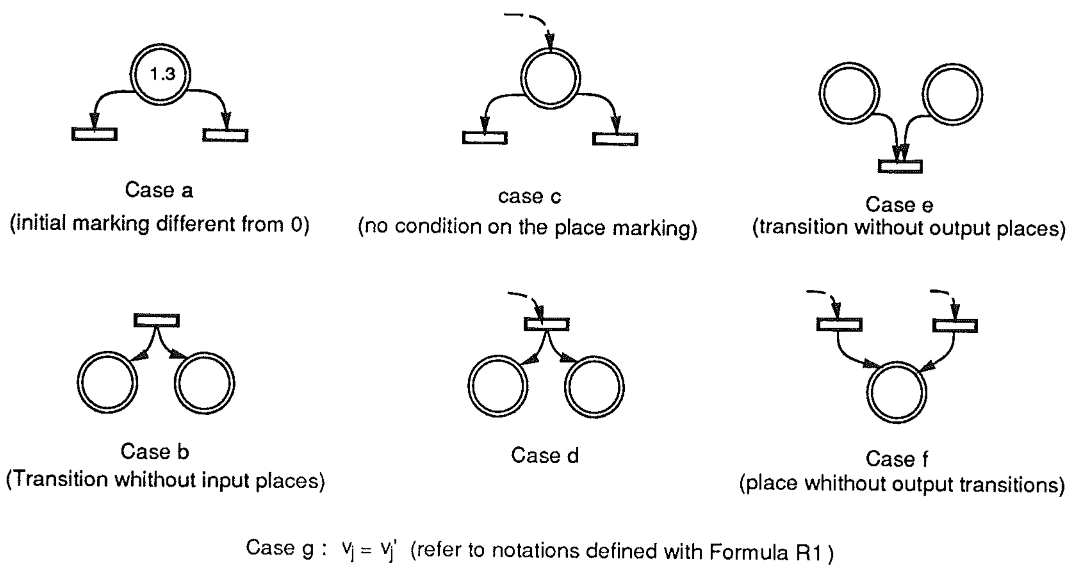

1.3
Case a
(initial marking different from 0)
case c
(no condition on the place marking)
Case e
(transition without output places)
Case b
(Transition whithout input places)
Case d
Case f
(place whithout output transitions)
Case g : $v_j = v_j'$ (refer to notations defined with Formula R1)

Figure 5. The count of the flow

rules defined in the second modification. When all the flows that have been created have disappeared a speed state is obtained.
The proposed solution has only been successfully implemented with Algorithm 2 but it seems possible to build a version adapted to Algorithm 1.

The rules to numerate flows are the following :

```
Initially :
    Rule 1 : For each structure a do :
                    nb-flow ← nb-flow +1 per output transitions.
             End for.
    Rule 2 : For each structure b do :
                    nb-flow ← nb-flow +1 per output places
             End for.
Body :
    Rule 3 : When a place is updated then :
                    nb-flow ← nb-flow +n-1
                    (n is the number of output transitions).
    Rule 4 : When a transition is updated then :
                    if v=v'then :
                          nb-flow← nb-flow -1
                    else
                          nb-flow← nb-flow +m-1
                          (m is the number of output places)
                    End if-else.
    Rule 5 : If nb-flow ≠ 0 then :
                    continue to iterate.
             end-if.
```

It is interesting to use the previous rules only if several speed calculations are carried out simultaneously because in this case it is difficult to know how many calculations are running at the same instant. Otherwise a speed state is obtained since there is no more operations to do. The latest solution has been adopted in the HPN program.

A new problem

A new problem arises when applying Algorithm 2. It has been called "conflict between places" because it is related to the presence of a structural conflict and it is dual to the problem that Algorithm 1 has in the case of a conflict between transitions. The problem can occur only if there is a structural conflict involving several places. The situation is illustrated in Figure 6, Example a.

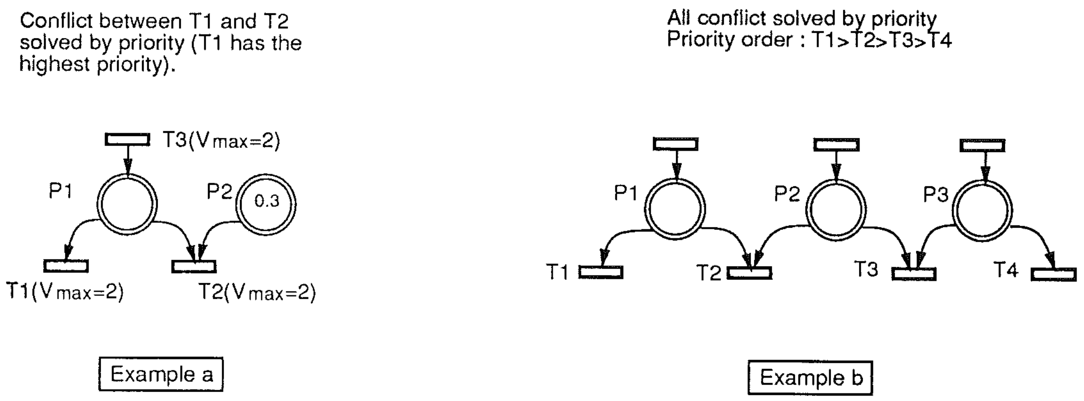

Conflict between T1 and T2 solved by priority (T1 has the highest priority).
T3(Vmax=2)
P1
P2
0.3
T1(Vmax=2)
T2(Vmax=2)
Example a
All conflict solved by priority
Priority order : T1>T2>T3>T4
P1
P2
P3
T1
T2
T3
T4
Example b

Figure 6. Conflict between places

Let us assume that T1 takes priority over T2 and we execute Algorithm 2. The chosen place order is P1/P2. Therefore the output transitions of P1 are updated first, i.e., v_1 and v_2 are respectively determined equal to 2 and 0. After that the output transition of P2 is tested and v_2 cannot be modified. The CCPN simulation seems all right. Now we keep the same initial conditions but we commute the order of P1 and P2 (it becomes P2/P1) and apply Algorithm 2 again. The output transition of P2 is updated and the value 2 is assigned to v_2 because the flow passing through T3 has not been oriented to T1 yet. When updating the output transitions of P1 v_1 is determined equal to 0 because there is no available flow (it has already been "taken" by T2). The problem arises because when using Formula R1 we are not aware of whether or not we have the right to "take" the flow. Rather than modify Formula R1 it has been preferred to define a new attribute for places, 'priority', in order to be able to update them by priority order. To avoid the user from having to select the priorities a small procedure has been introduced.

Procedure to set the priorities of places

Initialization :

```
    The priority of each places is equal to 10.
```

Body :

```
    If there is a structural conflict between the output transitions of P then :
        If the output transition of P with the highest priority (T)
        has several input places or the conflict is solved by sharing then :
            Subtract 1 to the priority of P;
        Else
            add 1 to the priority of P;
        End If-else.
    End If.
```

This is generally enough to solve the majority of the cases, but the situation illustrated in Example b is randomly solved (P3 and P2 have the same priority) if the user does not set himself the appropriate values of 'priority'. It can be noticed that the procedure can be improved by adding the value of the highest priority of the output transitions and subtracting (1/The highest priority) instead of adding +/- 1. The last improvement permits to carry out the situation of Example b. Effectively if the priority of T1/T2/T3/T4 are respectively 4/3/2/1 then the priority of P1/P2/P3 are 14/9.666/9.5. Therefore the calculation order is P1, P2, P3 and it reaches the desired result.

Implementation of the modifications

To make it possible to use all the presented modifications in the same program the following structure has been adopted :

```
Initialization
Repeat until List is empty
    Update-first-element-of-list
End repeat
```

Here List is a list of places (because the iterations have to be performed on places to be able to manage conflicts either by sharing or by priority). The list is built dynamically by the procedures Initialization and Update-first-element-of-list. Initialization inserts in List all places which are marked or at the output of a source transition (the first point of the second modification). Update-first-element-of-list carries out the speed determination of the output transitions of the first List element and inserts new elements in List. The new places are inserted if they are non marked and one of their input transitions fulfills the condition $v_j \neq v_j'$ (second and third point of the second modification). However a place is not inserted if it already belongs to List. In order to make possible to solve the new problem places are classified by priority order such that

the highest priority corresponds to the first element of List.
The motivation for using a dynamically built list of places when implementing the revised algorithm are :

- The two main loops of Algorithm 1 (*Repeat until end=true* and *For i=1 to i=N do...*) are replaced by a single loop : *Repeat until* List is empty.
- It is not necessary to use the third modification because there is no calculations in parallel. The speed state is reached when List is empty.
- It provides an easy way to update places by priority order.
- The usage of List can easily be extended to HPNs.

The procedure New-IB-state of the HPN program has been constructed for the same model. The main modification between the CCPN implementation and the HPN implementation is that List may contain continuous and discrete places in order to mix the speed determination and the reservation of tokens. The mixture of discrete and continuous operations is realized to avoid the problem prompted by situations such as in Figure 7.

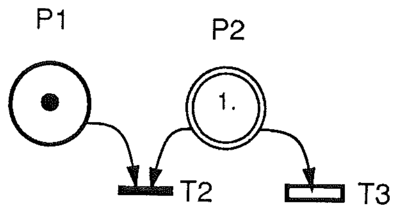


Figure 7. Mixing of speed calculations and token reservations.

Let us assume that no mixing takes place and the token reservation is carried out by one procedure whereas another determines the speed state. Therefore they have to be ordered in the program, for example reservation of marking before speed calculations. In Figure 7 if the reservation of tokens alway occurs before the speed calculation then it is impossible to give the priority to T3. The reciprocal order leads to the opposite problem : T2 can never get the priority. Moreover if the previous operations are separately executed the user has not the ability to choose between behaviors. The problem is mastered by generalizing the attribute 'priority' to d-places and by extending the capacities of Update-first-element-of-list (it will be able to either carry out a speed calculation or a token reservation). After adaptation the CCPN program structure also appears as a perfectly convenient structure for HPN program.

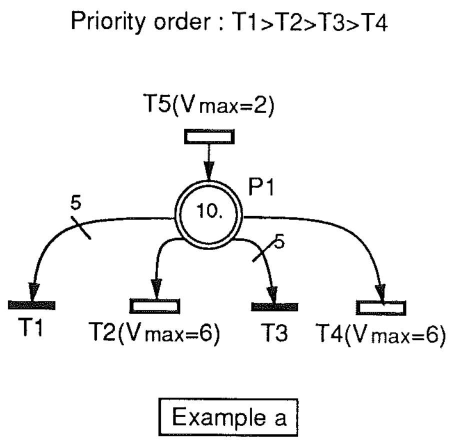
Resulting from the implemented way and from the modifications of the speed calculation algorithm some new conflict situations can be encountered and have to be treated. The first of them is met when solving an effective conflict by sharing. The adopted formula to calculate the speed of the output transitions in this case is :

$$v_j = \frac{\text{flow part} * \text{input flow}}{\text{the weight of the arc}}$$

In which : Flow-part is the proportion of the input flow that have to pass through the transition.
: Input-flow is the flow entering the place P (P and its output transitions form the structural conflict).
: The weight of arc is that of the arrow between the transition and P.

It may happen that the speed determined by this formula is superior to the maximal possible speed. Therefore it has been chosen to automatically solve the effective conflict by priority. We consider that the conflict is insolvable by sharing.
Other new conflict situations arise when a conflict involves continuous and discrete transitions, it is called h-conflict. It is the most complicated case because the available marking can be modified while solving the conflict. Figure 8, Example *a* illustrates this phenomena. When updating transitions by priority order in this example the following operations are performed :

1. T1 is enabled because the number of available marks in P1 is superior to the weight of the arc (10>5) $\Rightarrow$ 5 units of markings are reserved by T1. Left 5 non-reserved marking units in P1.
2. T2 is strongly enabled because P1 contains a quantity of non-reserved markings $\Rightarrow v_2 = V_{2max}$
3. T3 is enabled because the number of available marks in P1 is equal to the weight of the arc $\Rightarrow$ **Problem** : if we reserve the adequate marking for the firing of T3, T2 is no more strongly enabled.



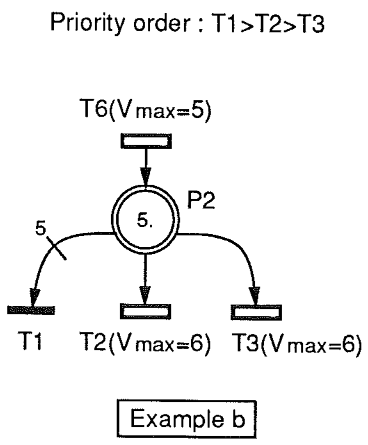


Figure 8. New conflict situations to take into consideration.

In order to preserve the priority order of the transitions the chosen behavior for T3 is to not reserve the marking although it is fireable. Notice that if in the illustrated case the speed of T5 is superior or equal to 12 (=maximal output flow of P1) T3 could have reserved the marking without altering the priority order. To solve the problem the following rule is used :

> **Rule 1** : An enabled d-transition (T) belonging to a h-conflict is not fired if the three next conditions are verified at the same time :
>
> - The marking of P[8] is equal to the weight of the arc connecting P to T.
> - At the starting of the calculations *inflow* < *outflow* (Refer to Algorithm 2 for the determination of both antities).
> - There exists a least one c-transition with a higher priority than T.

Using the previous rule and the possibility to solve conflicts between c-transitions by sharing a new behavior can be specified in case of h-conflict. The operation sequence to carry out Example *b* can be the following :

1. T1 is enabled : 5 units of markings are reserved.
2. There is no more available markings now. The situation is similar to a classical conflict between c-transitions so we can choose to solve it by sharing : v_2=2.5 and v_3=2.5.

Notice that this behavior neither corresponds to solving the conflict by priority nor to solving it by sharing. It is an 'hybrid' way.

8 P is the place with several output transitions.

4. Conclusion

The main difference between ordinary, discrete PNs and continuous nets is the nature of the marking in a place. In ordinary PN the marking is discrete, i.e., the value can be any integer greater or equal to zero.

The work on continuous PNs has two motivations. The first motivation is to be able to approximate discrete PNs with a large number of markers. The second motivation is to be able to model continuous phenomena. The majority of the work in continuous PNs are based on the first motivation. Different models of continuous PNs have been developed that to different degree approximate discrete PNs. Continuous PNs with constant speeds (CCPN) are very easy to simulate but can give poor correspondences with discrete PNs (Refer to Appendix E). Continuous PNs with variable speeds (VCPN) correspond very closely to the behaviour of discrete PNs but are time consuming to simulate. Continuous Petri nets with asymptotic speed (ACPN) constitute a compromise between CCPNs and VCPNs.

However, if we are mainly interested of the second potential of continuous PNs, modeling of continuous phenomena, none of the continuous Petri net model are sufficient. If we look upon a continuous PN from a continuous view point the places correspond to integrators that may only take positive values. This can be used. e.g., to model tank systems where tanks correspond to places and the evolution of markings between the places correspond to a flow between the tanks. With a CCPN it is possible to model tanks with constant in and outflows (refer to Appendix D). However none of the above models are sufficient for modeling, e.g., a tank with a free outlet where the outflow is a function of the level of the tank, i.e, the marking of the place.

In order to use PN formalism to model general continuous phenomena it is necessary to extend the existing continuous Petri net models in order to obtain the same expressive power that is available in ordinary differential equations. However, since good simulation tools exist for differential equations the value of this extension is questionable if only continuous PNs are concerned.

If, however, one also considers hybrid PNs where continuous and discrete places can be mixed together the situation becomes different. Combined discrete-continuous systems is a very useful here. With hybrid PNs the same model and graphical representation can be used to model both the continuous and the discrete parts of a combined or hybrid system. Therefore in this context it is definitely of interest to extend the continuous Petri net model.

In this thesis the main purpose has been to implement a system that makes it possible to simulate continuous and hybrid PNs in G2. The project has not allowed any extensions of the existing Petri net models. Due to its simplicity and the fact that it can model some continuous processes, e.g., tanks with constant in and outflows, it was decided to only look upon CCPNs and their use in hybrid PNs. When implementing the algorithm for the evolution of CCPN several problems were encountered. Three modification have been suggested to overcome these problems.

The developed G2 toolbox for HPN provides a user-friendly, graphical environment where HPN can be simulated.

5. References.

[1] C.A.Petri (1962), *Kommunikation mit Automaten, Schriften des Rheinisch*, Westfalischen Institutes fur Instrumentalle Mathematik and der Universitat Bonn, traduit par C.F Greene, Applied Data Research Inc., Suppl., 1 to tech.
Report RADC-TR-65-337, N.Y., 1965.

[2] R.David and H.Alla (1992), *Petri nets and Grafcet*, Prentice Hall.

[3] J.Le bail (1992), *Sur les réseaux de Pétri continus et Hybrides*, Thése de doctorat de l'INPG, Grenoble.

[4] P.Sarraut (1991), *Implementation of a toolbox for colored Petri nets in G2*, Thése de DEA de l'INPG, Grenoble.

[5] R.David (1993), *Cours de l'ENSIEG sur les réseaux de Petri*, ENSIEG,Grenoble.

[6] P.Ladet (1993), *Cours de l'ENSIEG sur les réseaux de Petri*, ENSIEG,Grenoble.

[7] Gensym Corporation (1992), *G2 reference manual*, Gensym Corporation, Cambridge.

Appendix A. Discrete Petri nets

The purpose of this annex is to present basic notions of PNs to make unexperienced readers understand all the parts of this report. For a formal presentation over the notions rewieved in this section refer to books enumerated in the bibliography. Therefore only basic notions related with this master thesis are gradually introduced and essential definitions given.

Graphical elements

A PN is composed of two types of elements : Places and Transitions, respectively represented by circles and bars (or boxes). These nodes are connected together by oriented arcs. An example of a PN is shown below. The notions of in and output places are defined with the help of the example of Figure 9. P1 is said to be an input place of T2 because there exists an arc from P1 to T2 and reciprocally P2 is said to be an output place of T2 because there is an arc from T2 to P2. In the same way input transitions and output transitions are defined.
Two particular graphical aspects need to be noticed in Figure 9 :

- Transition T7 does not have input places, it is called a *source transition.*
- Place P1 has two output transitions T2 and T3. We say that there is a *structural conflict.* More generally there is a *structural conflict* if there exists at least two output transitions linked to the same place.

Special rules are often used to govern the PN evolution when encountering one of the previous structures.

A marked PN is obtained from an unmarked PN by deposing an in-

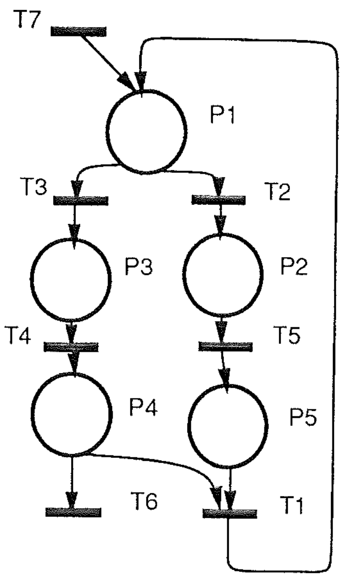


Figure 9. Non marked petri net

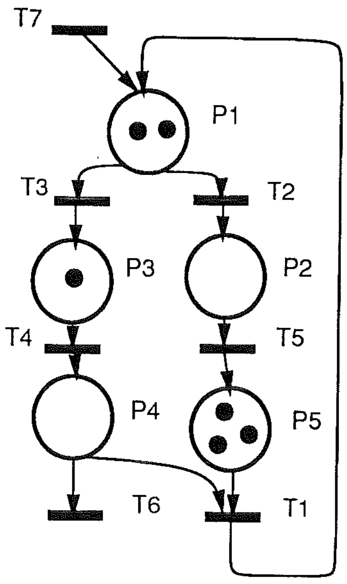


Figure 10. Marked petri net

teger number of marks or tokens in places. The integer number is called the marking of the place and it is noted m_i ($_i$ corresponds to the number of the place). The vector whose components are the 'm_i' of each places of a PN is called *the vector of markings* and it defines the state of a PN at a certain moment. For example the marking vector of Figure 10 is (2,0,1,0,3).

The evolution of a marked PN

A PN evolves from one state to another by firing transitions. The firing of a transition consists of taking a token from each of the input places of the transition and adding a token to each of its output places. However to be fired a transition must be enabled (or fireable).

Definition 1
: A transition is said to be fireable or enabled if each of the input places of the transition contains at least one token.

Remark 2
: A source transition is alway enabled.

Figure 11 illustrates the firing of transitions in a PN. In Example 1 T1 is fireable because each of the input places have at least one token. In Example 2 T2 is not fireable because there is no token in place P1.

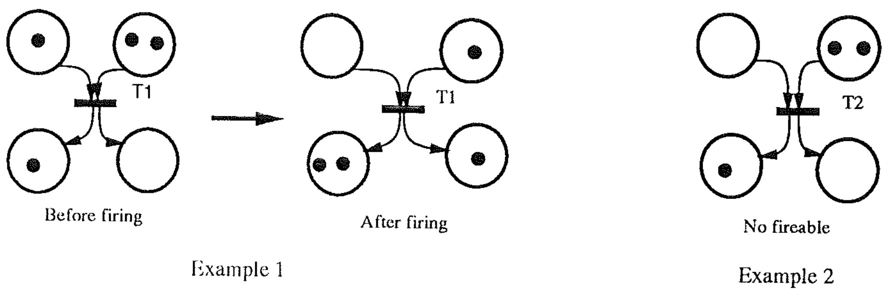


Figure 11. The firing of transitions

These rules are sufficient as long as we do not meet the situation illustrated by Figure 12, Example 1. In this figure T1 and T2 are enabled but they cannot be fired at the same time because there is only one token for two transitions. The user has to choose if the token is going to pass trough T1 or T2. In doing that the user imposes a priority order among the transitions involved in the conflict. This situation is called an effective conflict.

Definition 2
: An effective conflict is the existence of both the following conditions : there is a structural conflict between P and its output transitions, and the marking of P is smaller than the number of enabled output transitions.

Some more applications of this definition are shown in Figure 12.

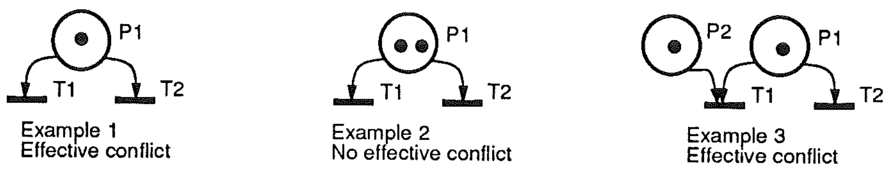


Figure 12. Conflict situation

When implementing PNs on a computer, we say that we solve the conflict *a priori* to signal that in case of a conflict the algorithm gives every time priority to the transitions in the same order. We simulate the intention of an user who chooses every time transitions in the same order.

PNs are divided into two groups: autonomous PNs and non-autonomous PNs. In autonomous PNs firing instants are unknown or not indicated. Consequently they can be used only to describe what happens. It is only a qualitative approach mainly used to explain the PNs evolution. All PNs on the previous pages are autonomous PNs.

In non-autonomous PNs the firing of a transition is associated with the occurrence of an event. Therefore it is possible to model what happens and also when it happens. Two types of non autonomous PNs have been implemented in G2 : synchronized PNs and T-timed PNs.

- In synchronized PNs the firing of a transition is related with the occurrence of an external event. For example in the PN of Figure 13 event E_1 occurs when the user clicks with the mouse on transition T1. In this case the PN is synchronized with the decision of the user to click on the transition.
- In T-timed PNs a timing is associated with each transitions. For example a timing of d=2 on a transition means that the firing of the transition occurs 2 units of time after it has been enabled. It can be noticed that similarly P-timed PNs are obtained by defining a timing attached to place.

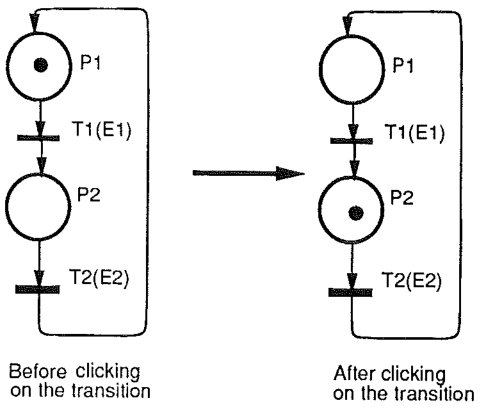

P1
T1(E1)
P2
T2(E2)
Before clicking on the transition
P1
T1(E1)
P2
T2(E2)
After clicking on the transition

Figure 13. A synchronized PN

T-timed PNs

Both the firing and the condition to enable a timed transition are slightly different from the previously described rules. The firing of a timed transition involves the following three steps:

1. The reservation of marks. The transition *reserves* the tokens that are going to be removed from the input places. A reserved tokens cannot be used to enable another transition. In Figure 14 reserved marks are visualized by white marks.
2. Waiting for the number of time units indicated by the timing of the transition.
3. Deleting reserved tokens from the input places and add non reserved tokens to the output places.

The enabling condition for a transition becomes:

Definition 3
: A T-timed transition is said to be enabled or fireable if each of its input places contain at least one **non reserved** token.

This type of PNs are often used with the hypothesis of evolution at maximal speed. It implies that there is no time elapsed between the instant when the transition can be enabled and the instant when the reservation of markings occurs. This hypothesis leads to the evolution illustrated in Figure 14, Example *a*.
In this example it can be noted that two marks are reserved by T1 between the time 0 and d_1. We often prefer a transition reserving its tokens only when the previous firing is completed. To reach this purpose we have to introduce a place such as P3 to each transitions in the PN (Figure 14, Example *b*). In order to avoid this excess of places an implicit limitation is defined (Figure 14, Example *c*). The evolution is the same as Example b, but P3 is no more represented: it is implicit. In all the parts of the report the implicit notation is used.

Generalized PNs

A PNs is said to be generalized if there is a *weight* (an integer number strictly greater than zero) attached to each arc. This new element can be added to all previous PNs. It modifies the firing and enabling conditions of transitions. For example they become for an autonomous PN:

Definition 4
: A transition is said to be enabled if each of its input places contains at least a number of tokens equal to the weight of the arc linking the place to the transition.

Remark 3
: Conventionally if the weight is not indicated on the arc it is equal to one.

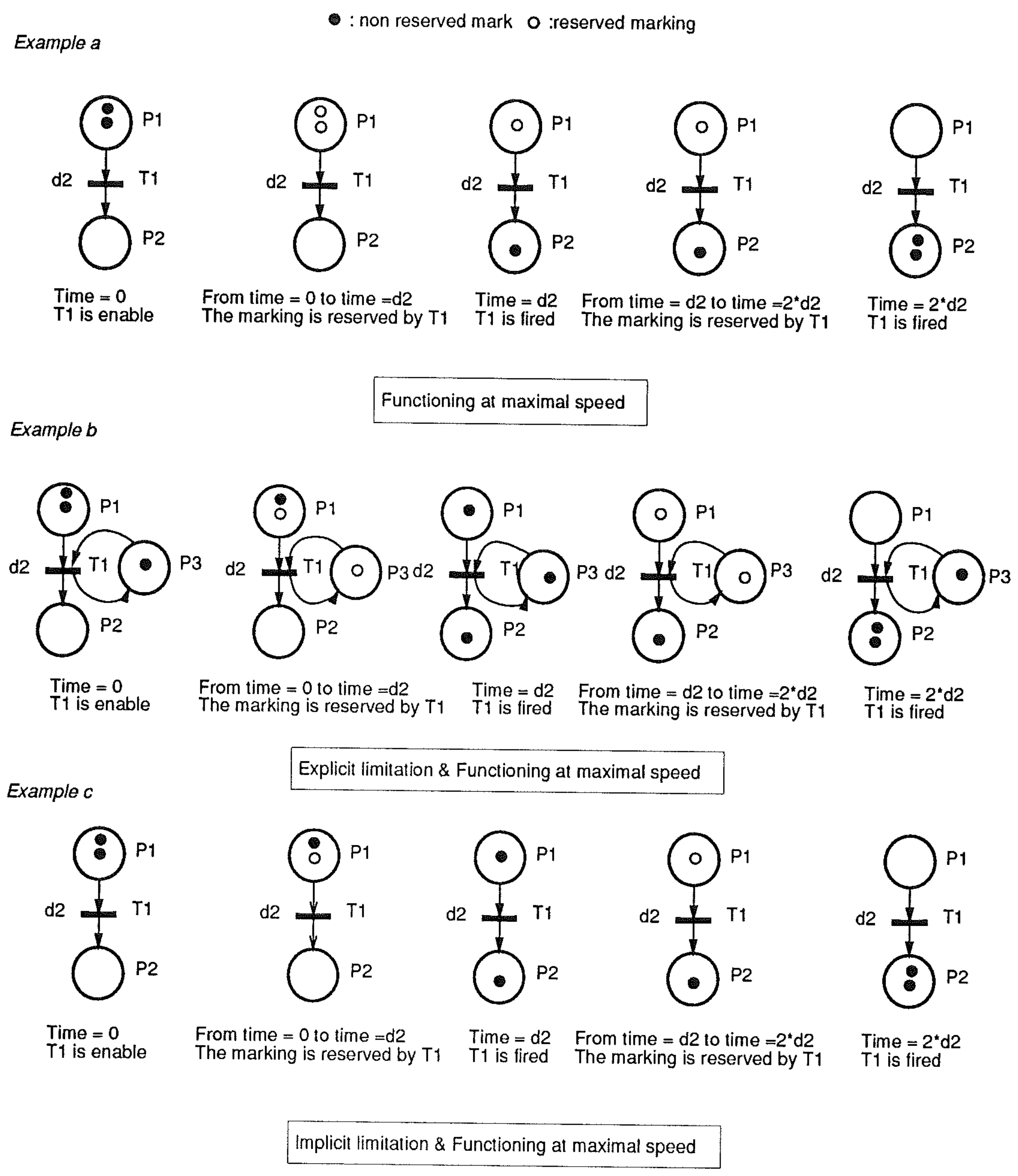

● : non reserved mark o :reserved marking
Example a
P1
d2
T1
P2
Time = 0
T1 is enable
From time = 0 to time =d2
The marking is reserved by T1
Time = d2
T1 is fired
From time = d2 to time =2*d2
The marking is reserved by T1
Time = 2*d2
T1 is fired
Functioning at maximal speed
Example b
P3
Explicit limitation & Functioning at maximal speed
Example c
Implicit limitation & Functioning at maximal speed

Figure 14.

Furthermore the firing of a transition is carried out by deleting from the input places and creating in the output places a number of tokens equal to the weight of the arc connecting the place to the transition.
In a similar way the functioning rules for generalized synchronized PNs and generalized T-timed PNs can be deduced. An example is given in Figure 15.

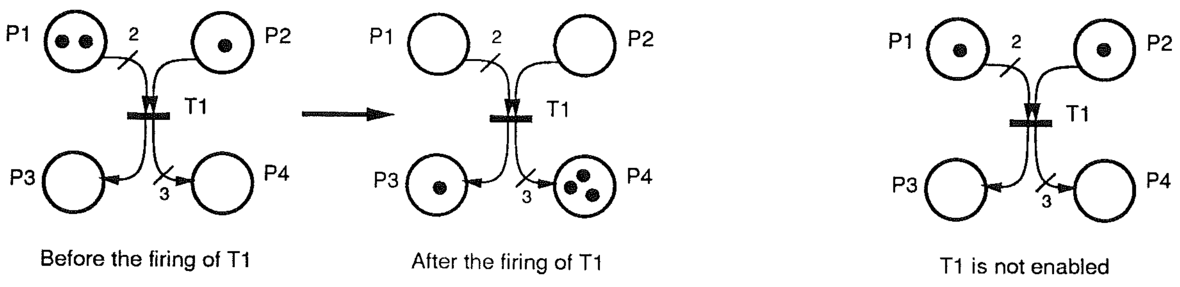


Figure 15. Generalized PN

Appendix B. Continuous Petri nets (CPN)

There exists three types of continuous petri nets : continuous petri nets with constant speeds (CCPN), continuous petri nets with variable speeds (VCPN) and asymptotically continuous petri nets. Only the first of these CPNs has been implemented in G2, therefore only CCPNs are presented. In this section the primary concerns of CCPNs are intuitively introduced. The evolution rules are dealt with for both autonomous and non-autonomous CCPNs. Finally the evolution of the marking is described and the algorithm governing CCPNs is overviewed.

Intuitive introduction of CCPNs

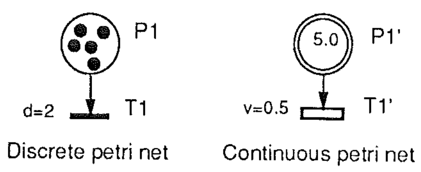


Figure 16. Discrete and continuous PNs

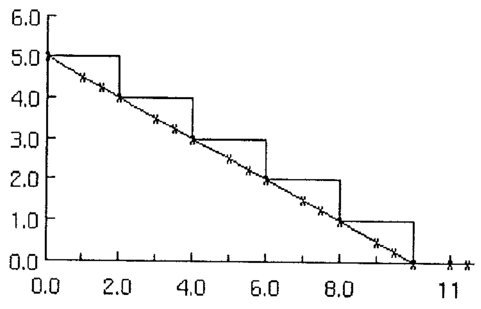


Figure 17. Evolution markings

Figure 16 shows a discrete T-timed PN and the associated CCPN. To distinguish continuous and discrete PNs the places of a continuous PN (called c-places) are visualized by two circles and the transitions (called c-transitions) by an empty box. Figure 17 presents the evolution of both markings in the d-place (called discrete or d-marking) and c-place (called continuous or c-marking).

It can be noticed that the marking in the c-place operates as an approximation of the d-place marking : this is the main concern of a CPN. To make the CCPN realize this approximation the timing associated with the T-timed transition (noted d) has been replaced by a maximal firing speed (noted V) equal to the inverse of the timing. The maximal speed associated with the output transition of P1' implies that the marking of P1' at $t + dt$ is deduced from the marking at t by the relation :

$$m_1(t + dt) = m_1(t) - v * dt \qquad (1)$$

Note : The quantity $v * dt$ represents the amount of markings leaving P1' during dt.

The CCPN realises a good approximation of the discrete PN on this example because the continuous marking and discrete markings are equal every firing instant ($t = 0; 2; 4 \ldots$). However they differ between these

moments, the marking in the c-place evolves continuously whereas the d-marking does not (the firing of the d-transition is discrete and occurs only every 2 units). Therefore we say that the c-transition is *continuously fired.* Indeed the continuous evolution of the marking in c-places implies that it is now indicated by a real number and no more by an integer number of tokens.
In the previous example we need only five calculations to draw the discrete marking evolution (one calculation corresponds to one firing). Therefore the motivation for using a CCPN is not an evidence. But now let us assume that initially fifty tokens belong to P1. Therefore fifty calculations have to be carried out to graph the evolution of the marking of P1 whereas only one is still required with the continuous PN to find the gradient of the line representing the evolution of the continuous marking.

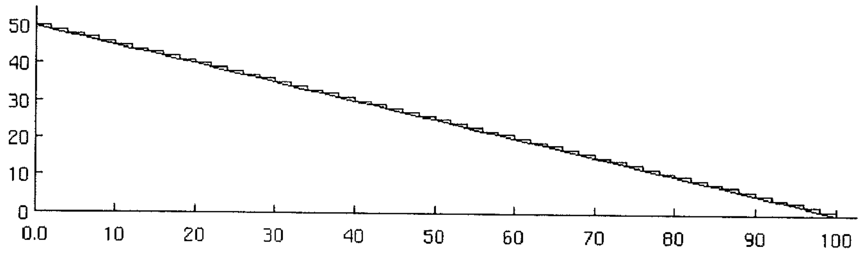


Figure 18. Marking evolution of P1 and P1' with an initial marking of 50 tokens

Three summarizing remarks can be stated :

1. Primarily CCPNs were designed to approximate the discrete marking of a PN in order to make simulations with large markings possible.
2. The approximation of CCPNs is better as the number of marks increases (to feel that compare Figure 17 and Figure 18).
3. The continuous firing of transitions in CCPNs makes it possible to model continuous processes that can be described by pure integrators such as, e.g, tanks with constant inflows and outflows.

Note also that it is often useful to interpret the marking of a continuous place as a level of liquid in order to fully understand the evolution rules dealt with the following paragraph.

Evolution rules

All structural properties defined for autonomous PNs are conserved for autonomous continuous PNs. Changes occur only in conditions to enable and fire transitions.
The following differences can be noticed :

- The initial marking may be a real number.
- Weights are real numbers.

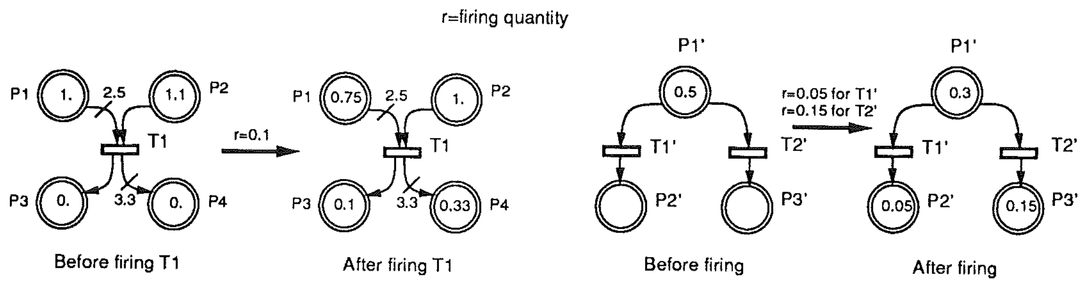


Figure 19. Firing of autonomous c-transitions

- A transition is said to be enabled if there does not exist an input place with a zero marking.
- The user chooses the firing quantity. Let us assume that the chosen firing quantity in Figure 19 is r. Therefore $2.5 * r$ are going to be removed from P1 and r from P2, similarly $3.3 * r$ are going to be added to P4 and r to P3. It is important to notice that the marking cannot be negative , consequently r must be inferior to $1/2.5$.
- The weight of an arc is not involved in enabling a transition. It only determines the maximal amount of marking able to pass through transitions.

A new property comes from the above points : there is no effective conflict possible in autonomous CPNs since the marking can alway be shared. In Figure 19 T1' and T2' are both enabled. They can alway be fired because it is possible to split the marking into two parts, one part going through T1' and the other through T2'.

In order to give a good approximation of the behavior of discrete T-timed PNs, CCPNs have been defined by assuming that it is possible to cut a mark into several tokens[9].The reasoning leading to CCPNs is graphically presented in Figure 20. To thoroughly preserve the same marking evolution in places P1 and P1' the transition (T1) must be duplicated into $T_1, T_2, \ldots, T_k$. If we do not duplicate T_1 only one token will be reserved in P_1' and only one fired at the instant d (bad approximation). But the transition duplication process becomes impossible when a mark is split into a large number of tokens (k becomes to large to represent the T_k). Therefore an approximation is adopted, it consists of replacing the k firings in parallel of transitions $T_1, T_2, \ldots, T_k$ by k firings in series of a single transition (T1" with a timing of d/k). Thereby the markings in places P1 and P1" are equal at the starting and end of the firing of T1 but they differ between both instants. Finally the continuous PN is obtained when k tends to infinity.

Because the timing d/k previously associated with T" is equal to zero when k is infinite, implying that the transition is continuously fired, we replace it by a maximal firing speed (denoted V_{max}) equal to the gradient of the line representing the marking evolution in cP1 ($=1/d$).

9 Notice the difference introduced now between marks and tokens

Hence two characteristics of CCPNs can be emphasized:

- CCPNs use the implicit limitation.
- The maximal speed of a c-transition must be chosen equal to the inverse of the timing attached to the associated d-transition in order to approximate the behavior of the discrete PN.

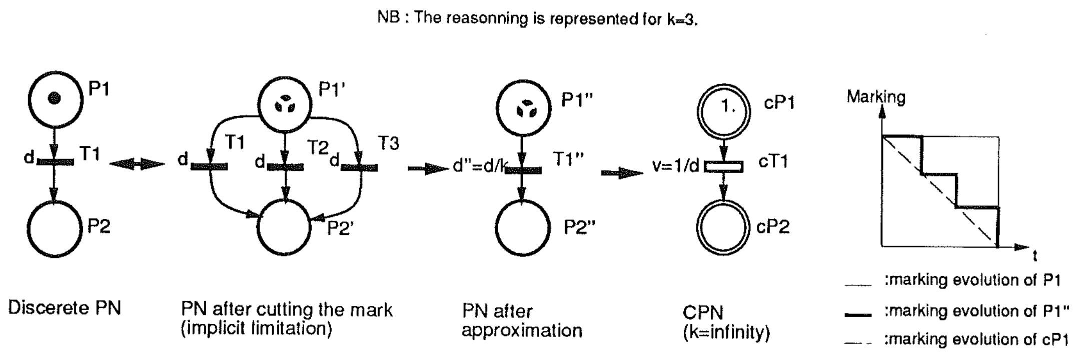

NB : The reasonning is represented for k=3.
P1
d
T1
P2
P1'
T1
T2
T3
d
d
d
P2'
P1"
d"=d/k
T1"
P2"
cP1
1.
v=1/d
cT1
cP2
Marking
t
Discerete PN
PN after cutting the mark (implicit limitation)
PN after approximation
CPN (k=infinity)
:marking evolution of P1
:marking evolution of P1"
:marking evolution of cP1

Figure 20. Evolution from discrete PN to continuous PN

The rules monitoring the evolution of a CCPN are quite different from these of a discrete PN. For example in CCPNs there are two kinds of enabled transitions called respectively *strongly enabled* and *weakly enabled*. Each of them implies a specifical way to calculate the speed of the transition.

Definition 5
: A transition is said to be strongly enabled at a time *t* if all its input places have a non negative marking.

Definition 6
: The speed or the instantaneous speed of a strongly enabled c-transition is equal to the maximal speed associated with the c-transition.

Definition 7
: A c-place is supplied at a time *t* if at least one of its input c-transitions is weakly or strongly enabled.

Definition 8
: A c-transition is weakly enabled at a time *t* if all input places with a zero marking are supplied.

Remark 4
: A c-transition source is strongly enabled.

Remark 5
: The maximal speed of a c-transition is chosen by the user whereas the speed (=instantaneous speed) is calculated.

The firing of weakly and strongly enabled transitions is illustrated in Figure 21. Example a illustrates the firing of a weakly enabled c-transition.

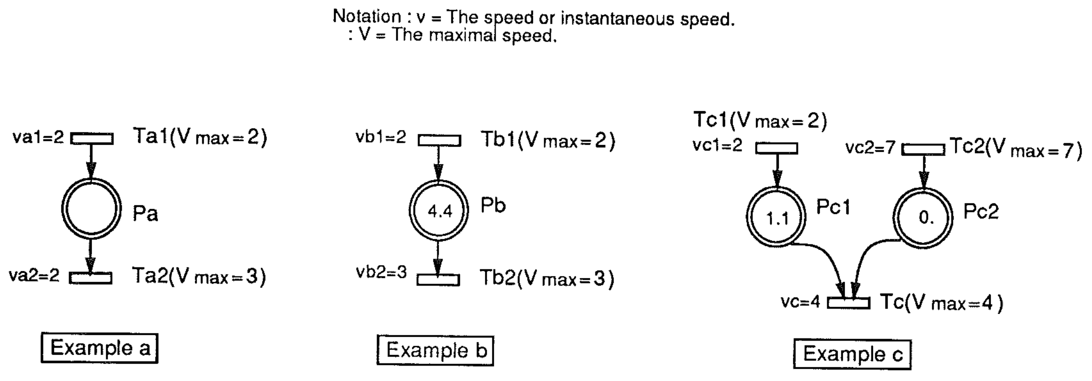


Figure 21. Firing of continuous transitions

During an elapsed time of *dt* a $2 * dt$ quantity[10] of markings is deposited in Pa through Ta1. Therefore a maximal quantity of $2 * dt$ can leave the place during the same time (we cannot take more than we receive in order to preserve the marking greater or equal to 0). Consequently the speed of the output transition (va2) can be chosen between 0 and 2. According with the hypothesis of the evolution at maximal speed the speed is fixed to 2. It is important to note that when a transition is weakly enabled its firing speed may be inferior to the maximal speed. The second example shows a strongly enabled transition. In this case the firing speed is alway equal to the maximal speed because we do not have to wait for the arrival of some markings, it is already in the place. In Example c Tc is weakly enabled and still the firing speed is equal to the maximal speed because Pc2 is supplied more than enough ($7 * dt$ enters Pc2 whereas only $4 * dt$ is able to leave Pc2 in the same time).
Similarly to discrete PNs difficulties are encountered in case of effective conflicts[11].

Definition 9
: An effective conflict exists when the three following conditions are fulfilled :

- There is a structural conflict between the output transitions of place P.
- P is not marked.
- P is not supplied enough to allow the firing of each of its output transition at maximal possible speed.

In other words we need a decision from the user each time an effective conflict occurs. To avoid that a predetermined behavior is adopted. A priority order can be specified for the output transitions in the same way as with discrete PNs. However with CCPNs another solutions is also possible. The input flow can be split between the transitions. In the latter

10 Refer to equation 1.

11 Note that effective conflicts exist only for CCPNs not for autonomous continuous Petri nets.

solution there is an infinity of ways to share the flow. The illustrated partition (Figure 22) is called sharing proportional to maximal speed.

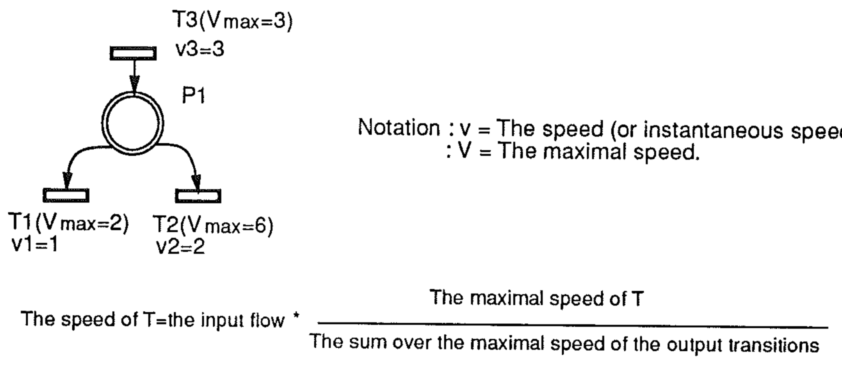

T3(Vmax=3)
v3=3
P1
T1(Vmax=2)
v1=1
T2(Vmax=6)
v2=2
Notation : v = The speed (or instantaneous speed).
: V = The maximal speed.
The speed of T=the input flow *
The maximal speed of T
The sum over the maximal speed of the output transitions

Figure 22. Sharing proportional to maximal speed

The marking evolution of a CCPNs

An important point appears from the previous rules : the speed of a CCPN does not depend on the quantity of markings in c-places. To determine the firing speed of a transition we need only to know whether or not the input places are marked, no matter if we have an 100 marking or a 0.1 marking. Therefore the speeds of a CCPN are changed only if a marking reaches a zero value (a phenomenon called c-event). Note that the opposite event, when a marking changes from 0 to a non zero marking does not modify speeds because it can occur only if the the quantity of markings entering is superior to the quantity leaving the place during the same period. Consequently the output speed was already equal to the maximal possible speed. In Figure 21 Example c, at $t = 0$, v_c is determined equal to 4, at $t = dt$ the marking in place Pc2 has become positive but the speed of T_c is not be changed.
The period of time between two successive c-events is called *functioning interval* and it is associated with a *speed state.* Speeds are unchanged over the functioning interval but the marking evolves. If we look at Figure 21 Example b, during one unit of time an amount of 2 passes through T_{b1} and a 3 quantity leaves through T_{b2}. It implies that the *balance* of P_b (denoted b_b) is equal to -1. More generally the balance of a place is defined as the quantity reaching the place minus the quantity leaving. It is easily calculated by subtracting input speeds from output speeds. Therefore the marking evolution in c-places is governed by the *fundamental relation* : $m_i(t + dt) = m_i(t) + b_i * dt$ or $\frac{dm_i}{dt} = b_i$ (where $_i$ indicates the number of the c-place). It is now possible to guess why CCPNs perform quicker than discrete PNs : using the fundamental relation we are able to precompute the next instant when a c-marking reaches a zero value. We do not have to wait until the marking evolves to this value. Furthermore the integration of the fundamental relation is easy to carry out because b_i is a constant .
The evolution of a CCPN from a speed state to another is represented by

a PN (named *evolution graph*, Figure 23) in which the instants when the changing of speed state occurs and the vector of markings at this time are indicated on the transitions. Places show the speed state vector during the functioning interval. Places responsible for the new speed state correspond to the new zero markings appearing in the vector of markings. It can be noticed that there is alway a terminal node, called *stable speed state* in which all balances are positive or nil implying that there will be no more c-events. There are no functioning rules linked with this PN it is only a way to visualize the different states of the net.

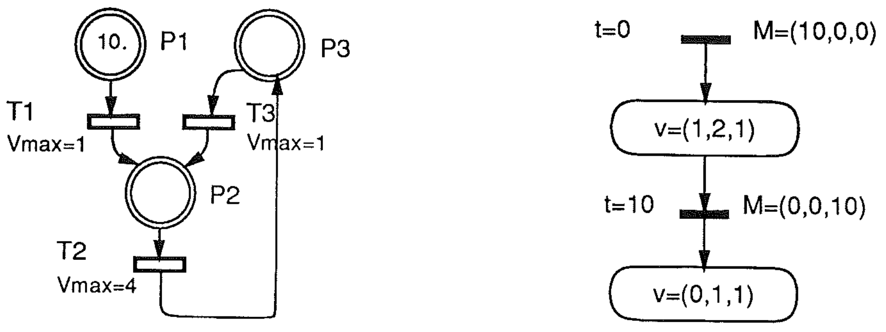


Figure 23. Continuous PN and its associated evolution graph

Calculation of the set of speeds

It is not so easy to calculate speeds only by applying the rules introduced in the second paragraph. Effectively to determine the speed of a transition it can be necessary to know the speed of other transitions (case weakly enabled). Therefore the situation illustrated in Figure 23 can arise. To determine v_2 we need to know v_3 and to know v_3 we need the value of v_2. To solve this recursive problem implied by definitions 7 and 8 an algorithm has been developed by David and Alla (1992). The algorithm will be explained with the help of Example 23.
The initialization of the algorithm consists of setting the instantaneous speed of each c-transitions equal to zero and classifying the transitions. In this example the order chosen for the transitions is : $T_1/T_2/T_3$. The speed of the transitions will now be updated in the previously defined order until a speed state is obtained.

1. T_1 is strongly enabled $\Rightarrow v_1 = 1$ (= the maximal speed[12] of T_1, refer to definition 5).
2. T_2 is weakly enabled $\Rightarrow v_2 = 1$ (because P_2 is supplied by T_1, refer to definition 8).
3. T_3 is weakly enabled $\Rightarrow v_3 = 1$ (because P_3 is supplied by T_2).
4. T_1 is strongly enabled $\Rightarrow v_1 = 1$ (the speed is unchanged).

12 The instantaneous speed are indicated with a small v and maximal speed are indicated with a capital V.

5. T_2 is weakly enabled $\Rightarrow v_2 = 2$ (because P_2 is now supplied by T_1 and T_3).
6. T_3 is weakly enabled $\Rightarrow v_3 = 1$ (the speed cannot be increased because it is already equal to the maximal speed).
7. The speed of T_1 cannot be changed.
8. The speed of T_2 cannot be changed.
9. The speed of T_3 cannot be changed $\Rightarrow$ a speed state is reached because the speeds can no longer be changed with this marking.

To graph the whole evolution of the PN further steps consist of calculating the instant of the next c-event with the fundamental relation and determining the new speed state at this moment. These operations are iterated until the stable speed state is reached. Finally the evolution graph of Figure 23 is obtained.
From the behavior of this algorithm three remarks can be made :

- The number of calculation steps is related to the order in which transitions are arranged. In the example if the order was T_3, T_2, T_1, the speed state would have been reached in twelve steps.
- One more iteration over each transitions is needed to check that their instantaneous speed can no more be changed (step 7, 8, 9 in the previous example).
- It is difficult to solve effective conflicts by sharing.

In order to solve these points a new algorithm has been implemented on G2.

Appendix C. Hybrid petri nets (HPN)

To model complex systems involving parts with a large number of tokens or continuous processes and parts with few tokens, HPNs has been developed. An HPN is the mixture of discrete PNs and CPNs where the discrete net can influence the continuous net and vice versa. Connections between discrete and continuous PNs can be regrouped into three classes :

1. The discrete part influences the continuous part (Figure 24, Example a). The illustrated net can be used to represent the logical state of a device and its consequences on the production flow. Notice that if a discrete place is linked to a c-transition the reciprocal arc **always** exists in order to preserve the marking as an integer.
2. The continuous net can influence the discrete net (Figure 24, Example b). It can, for example, model the decision to stop a machine (T_1) when the level in place P1 is higher than 6.5.
3. It is also possible to convert continuous markings to discrete markings and vice versa (Figure 24, Example c). It can be noted that there are no restrictions about types and manners to connect places to inputs or outputs of a d-transition.

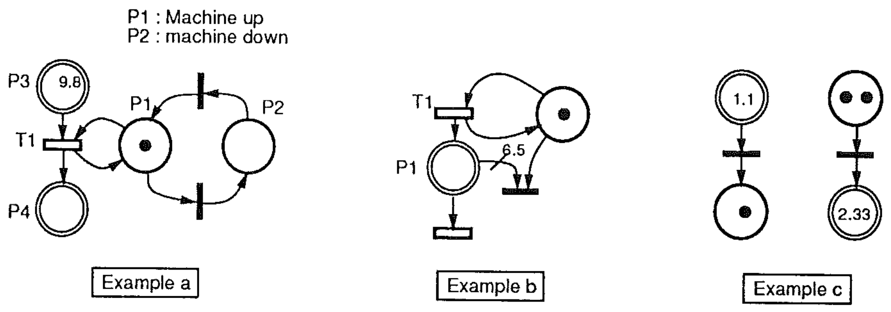


Figure 24. Hybrid petri nets

To allow these behaviors the enabling conditions of continuous and discrete transitions must be extended.

Definition 10
: A d-transition is said to be enabled at time t if each of its input places has a reserved marking greater than the weight of the arc linking them to the transition.

Remark 6
: In definition 10 it does not matter whether the input place is a continuous or a discrete one, the only difference is that the weight may be a real number in case of continuous places.

Definition 11
: A c-transition is said to be enabled at time t if all its input c-places have a positive non reserved marking or are supplied and all its input

d-places have a non reserved marking superior to the weight of the arc connecting them to the c-transition.

The way to fire continuous and discrete transitions remains unchanged and effective conflicts are solved like in the previous sections. The only new element to take into consideration is the possible presence of reserved markings in c-places (an example is given in Figure 25).
Finally also the notion of functioning interval is altered, it is replaced by the notion of *invariant behavior state* (IB-state). During an IB-state the marking in d-places, the speed vector and the reserved marking must be constant. Consequently three events can lead to a new IB-state. Both of them are already known : c-events (defined in the section about CPNs) and d-events which correspond to the firing of a d-transition. The new altering event (called h-event) occurs when the marking in a c-place reaches a sufficient level to enable a d-transition, as illustrated in Figure 25.

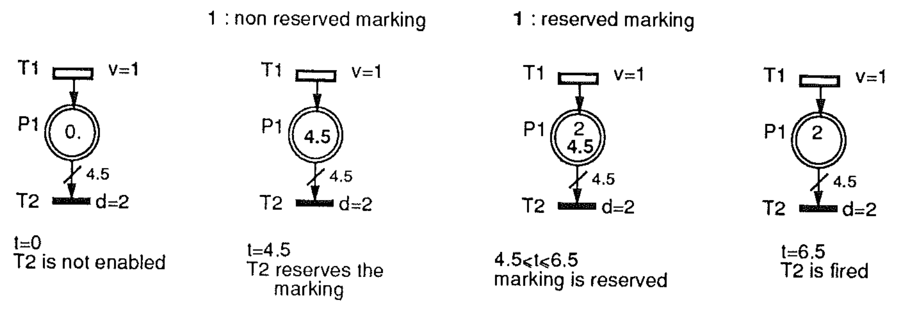


Figure 25. Hybrid event

Appendix D. An example of modeling by HPN

This appendix presents an example of a modeling by HPN. The system is represented in Figure 26. The evolution of the process to realized the mixture of the liquids contained in Tank 1 and Tank 2 is described below :

1. Valve 1 and Valve 2 are opened to fill Tank 1 and Tank 2. When one of the tanks reaches its maximal level, the input valve is automatically turned off.
2. When Tank 1 and Tank 2 are full, Pump 1 and Pump 2 might be started by the operator by clicking on the Button T1. Then Tank 3 is filled.
3. When Tank 1 and Tank 2 are empty Pump 1 and Pump 2 are stopped and Valve 1 and Valve 2 are opened to initiate a new cycle. These operations are automatically carried out. At the same time Valve 3 is opened to empty Tank 3.

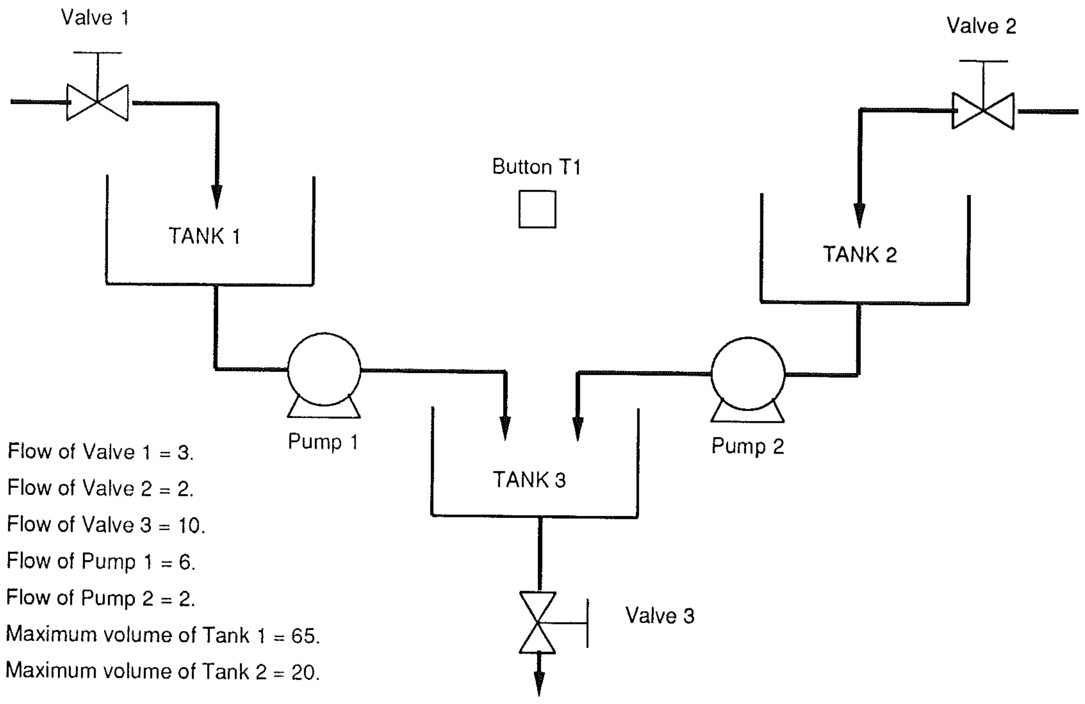


Figure 26. Feature of the process

The continuous part of the system (part where the liquid flows) has been modelized by the CCPN of Figure 27. It can be noticed the simplicity of the model : valves and pumps become c-transitions, tanks are represented by c-places and the tubes connecting valves, pumps and tanks are the arcs linking c-transitions and c-places. Note also that the flow of valves and pumps set the maximal speed of the c-transitions of the model.

The discrete part of the HPN monitors the model (Figure 28). D-places P1, P2, P3 are used to “turn on” or “ turn off” the c-transitions

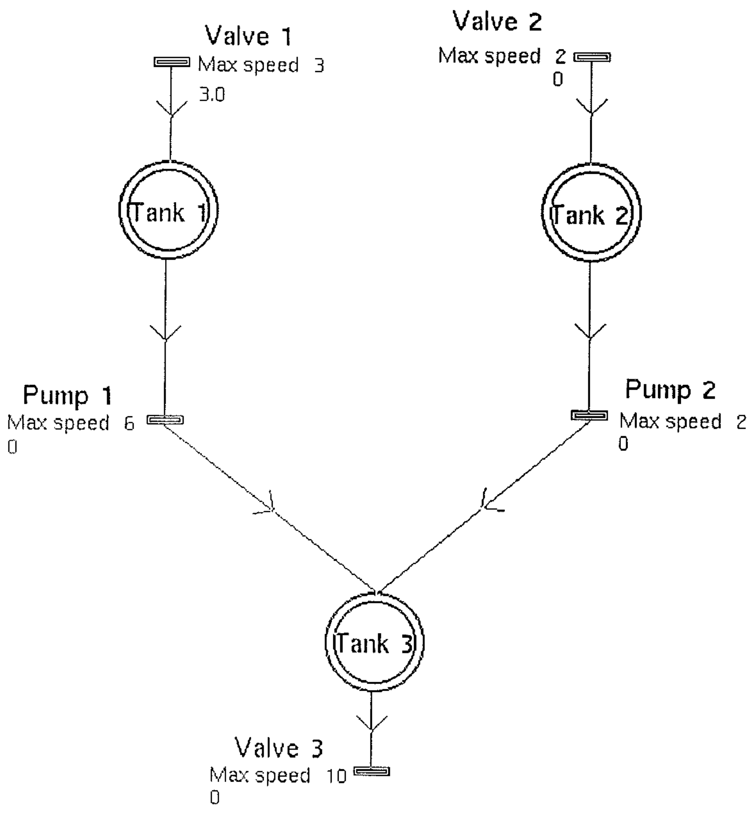


Figure 27. Modelization of the continuous part of the system

Valve 1, Valve 2, and Valve 3. Place P4 switches on or switches off both Pump 1 and Pump 2. The d-transition T2 is fired if the marking in Tank 1 is equal to 65 (the weight of the arcs connecting Tank 1 and T2 are equal to 65) and if P1 is marked (equivalent to Valve 1 opened) so the mark in P1 is removed and the speed of Valve 2 is set to 0. Transition T3 is used in a similar way, it is fired if the marking in Tank 2 is equal to 20 because the arcs connecting Tank 2 and T3 are equal to 20 and if P2 is marked. Button T1 is modeled by the d-event-transition T1[13]. T1 is enabled if the marking in Tank 1 is equal to 65 and if the marking of Tank 2 is equal to 20 (the weight of the arcs linking T1 and Tank 1 are 65 and the weight of the arcs linking T1 and Tank 2 are 20). When T1 is enabled its color is green and it is fired by clicking on it. Finally by firing T4 C-transitions Pump 1 and Pump 2 are "stopped" and the speeds of the C-transitions Valve 1, Valve 2, and Valve 3 are recalculated equal to their maximal speed. Note that T4 is fired when the marking in Tank 3 is equal to 85 (the weight of the arcs between T3 and Tank 3 are equal to 85) because this value implies that Tank 1 and Tank 2 are empty.

13 Refer to section 2 for the definition of d-event-transitions.

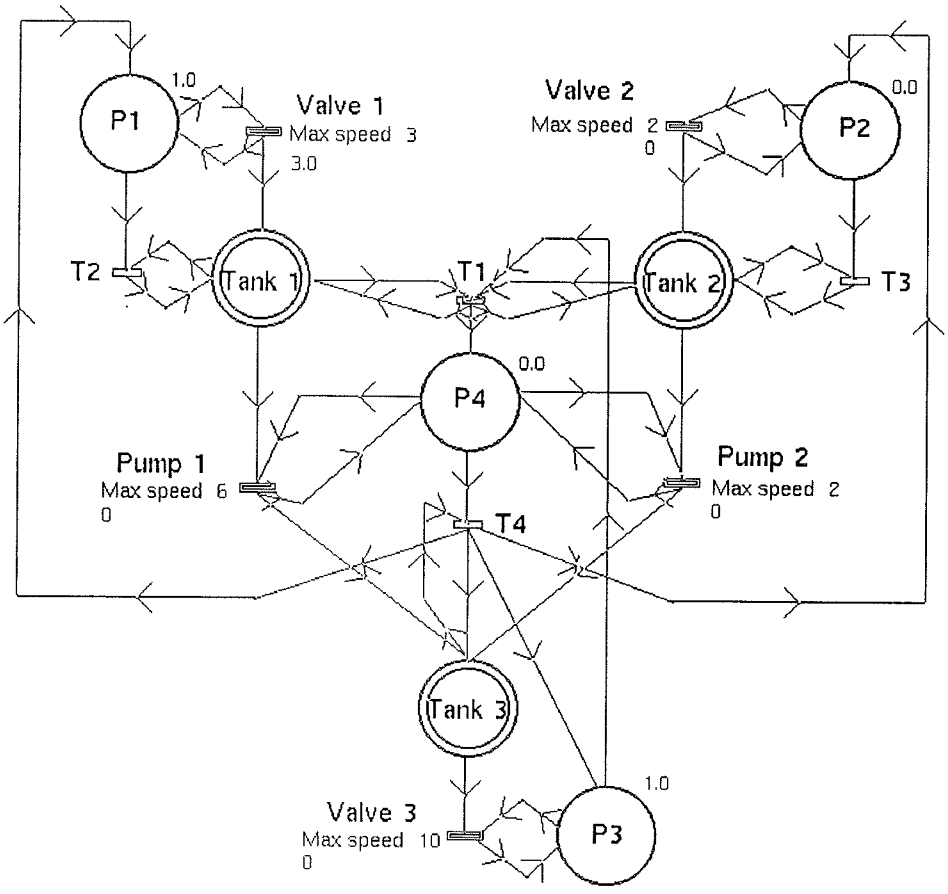


Figure 28. The HPN model

Figure 29 represents an example of the marking evolution of Tank 1, Tank 2, and Tank 3 obtained while simulating with G2.

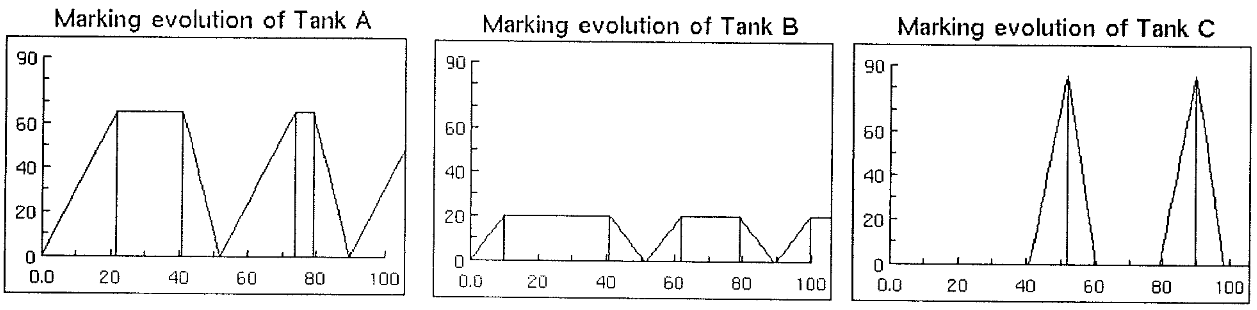


Figure 29. A sample of obtained marking evolution

Appendix E. CCPN examples

A discrete PN can be simulated only if a small number of tokens is involved in the simulation, otherwise the required time for the simulation is too long. CCPNs have been primarily developed to solve this problem by being able to rapidly provide an approximation of the discrete marking. Indeed the CCPN approximation is better as the number of tokens increases. While testing the HPN program on some examples, good and 'bad' approximations of the discrete behavior have been noted. This annex presents examples of the three different kinds of 'bad' behaviors detected.

The problem of periodical stable state

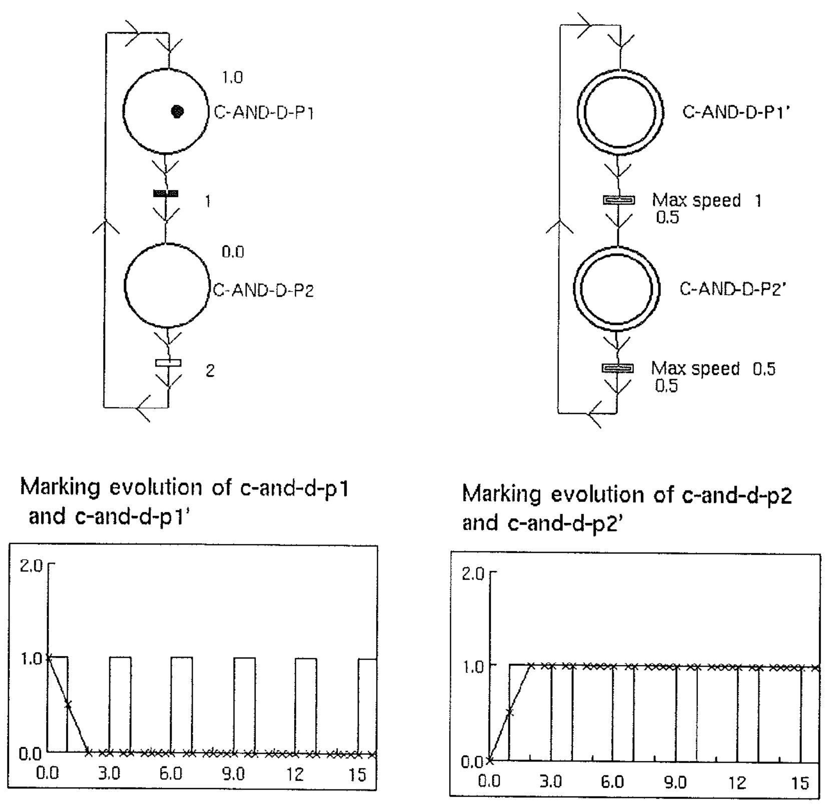


Figure 30. The problem of periodical stable state

In Figure 30 a stable periodical state is reached by the discrete PN. A good continuous approximation of this phenomena would have been a constant marking (after a transitional state) equal to the average value of the discrete marking on a period. It implies that in the illustrated example the continuous stable state should be : marking of C-and-d-p1' = 1/3 and marking of C-and-d-p2' = 2/3. This is not the case. The CCPN exaggerates the situation (0 instead of 1/3 and 1 instead of 2/3).

Note that the error between the theoretical good c-marking and the obtained c-marking is not necessarily inferior or equal to 0.5, it depends on the weight of the involved arcs. The weight of the arc linking C-and-d-p3 to T4 in Figure 31 is 10 and the graph of the marking shows an error of 5.5 (the average value of the marking in c-and-d-p3 is 5.5 and the value of the continuous marking is 0). This problem is solved by VCPNs or asymptotically continuous Petri nets.

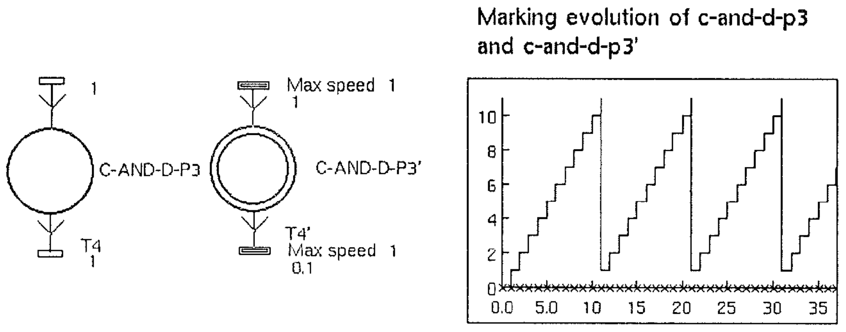


Figure 31. Periodical stable state involving weight different from 1

Problem of delays

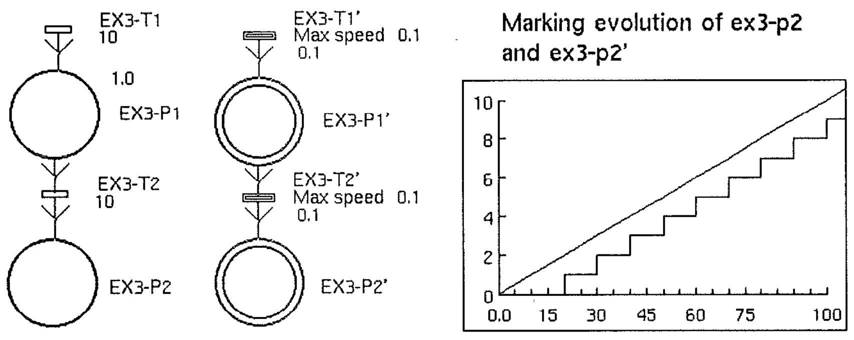


Figure 32. Problem of delays

The first deposited token in Ex3-p2 (Figure 32) occurs 20s after starting the simulation and, however the first infinitesimal fraction of c-marking is immediately deposited in Ex3-p2'. Therefore the 'bad' approximation of Figure 32 is produced. In other words the continuous flow is instantaneously established whereas the discrete flow is not. To correct this error a modification of the CCPN algorithm could be carried out (For example wait $1/V_{jmax}$ before starting the speed calculation of the output transitions of the output places of T_j).

Divergence of markings

In case of effective conflict it is possible to construct examples so that the difference between the marking evolution of the d-place and the equivalent c-place tends to infinity.

In Figure 33 the marking evolution of the place Pb4-p3 diverges whereas the marking of Pb4-p3' remains equal to 0. The 'bad' behavior results from the fact that two tokens reach the Pb4-pl place at the same time. Hence Pb4-t3 and Pb4-t4 can reserve one of them. In the CCPN marks continuously enter and leave the place Pb4-P1' and the sum of both the flow passing by Pb4-tl' and by Pb4-t2' is small enough to go through the same output transition (Pb4-t3').

Figure 34 presents a case where the continuous marking diverges whereas the discrete marking does not.

Note : *A better behavior of the CCPN is obtained in both previous cases if the effective conflict is solved by sharing (flow part of each output transitions equal to 0.5).*

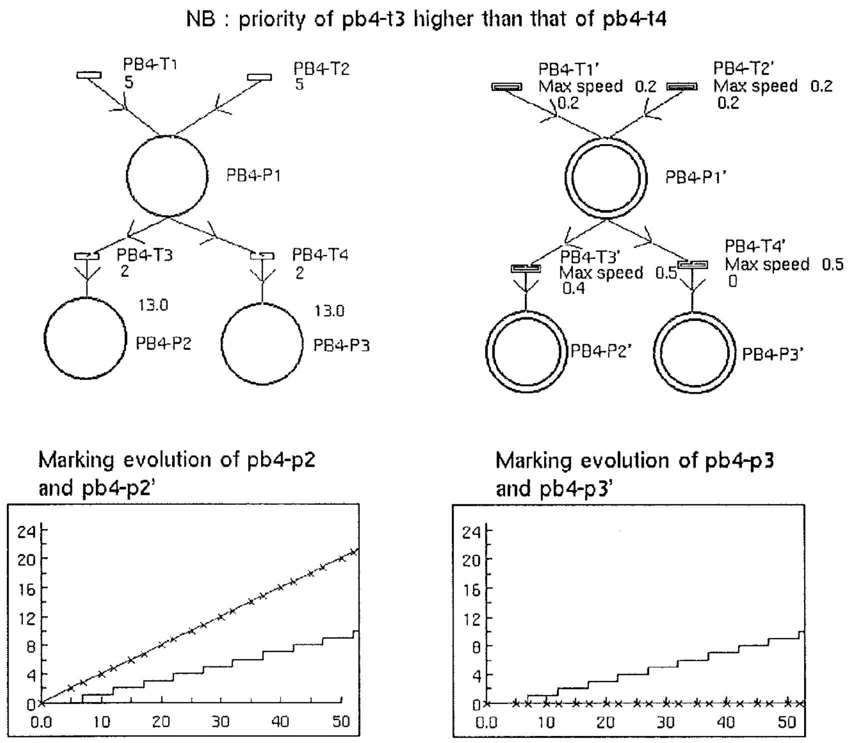
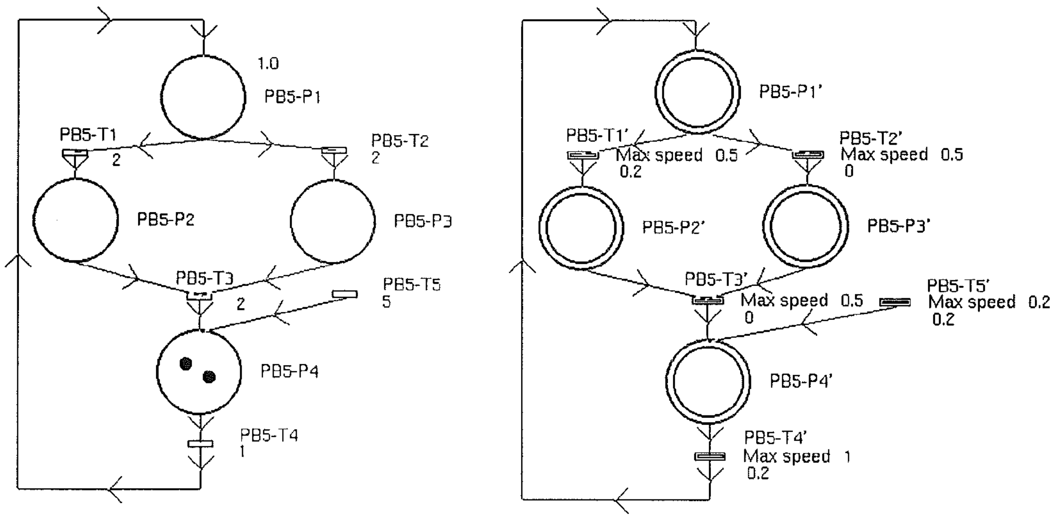


Figure 33. Divergence of the d-markings



Marking evolution of pb5-p2 and pb5-p2'

NB:Pb5-T1 has the priority over Pb5-T2

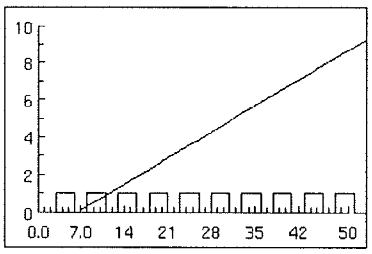


Figure 34. Divergence of the c-marking

Appendix F. User's guide

The purpose of this annex is to make the user able to use the main capabilities of the HPN program. When starting the program three modes can be selected: the creating mode, the running mode and the administrator mode. The last mode allows an unrestricted access to the whole knowledge base (dedicated to people who want to modify the program for example) whereas the others offer the user facilities to create and run the simulation of a PN. The next paragraphs deal only with the available possibilities in the creating and running mode[14].

The creating mode

The creating mode is mainly composed of two workspaces (Figure 35): a workspace situated at the top of the screen (called top-workspace) and one at the upper right angle (called right-workspace). They contain all elements to build a PN.

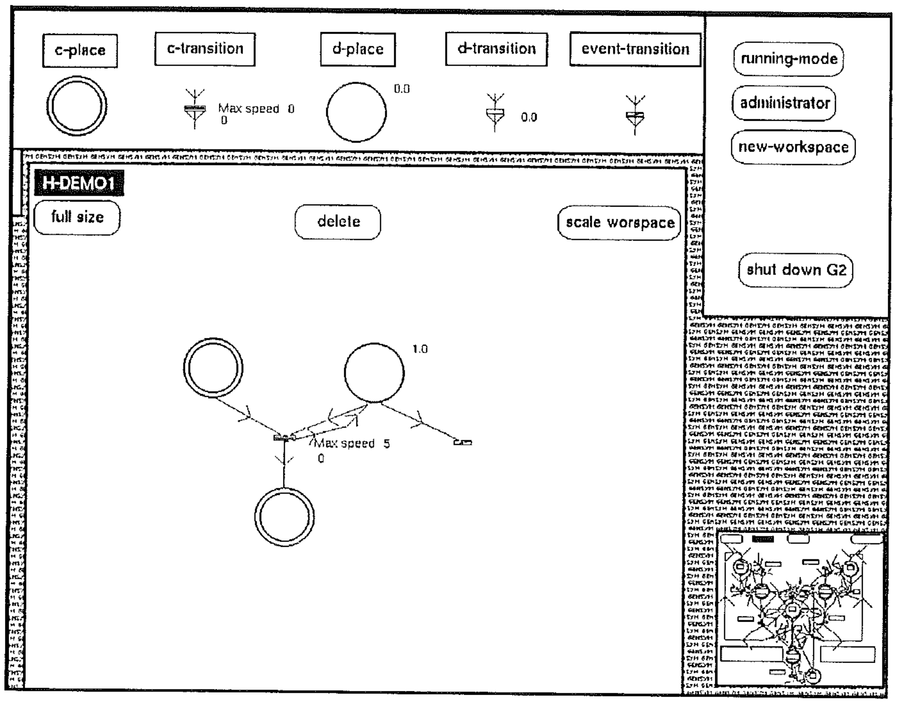


Figure 35. The creating mode window

There are two ways to get a PN. If it has already been created you have to click on the background of the screen (to make the *main menu* appear) and select "get workspace". A list of all the named saved workspaces is created. Selecting one of these makes it appear. A new PN can also be created by following the next instructions :

14 For the administrator mode read the introduction of the chapter "Application package security facility" in the reference manual for G2 version 3.0, 1992

Operation 1 : Select the "new workspace" button on the right workspace. A new workspace with three buttons is displayed in the middle of the screen (called middle workspace).

Scale-workspace button : The button "scale workspace" reduces the size of the workspace to one quarter of its full size and places it in the lower right angle of the screen.

Full-size button : The "Full size" button reestablishes the initial size of the middle workspace.

Delete button : The "delete" button hides the workspace and makes it transient, therefore when the knowledge-base is reset the workspace is deleted.

The middle-workspace is unrestricted therefore all the available operations on a G2-workspace are allowed[15] (such as lift to top, drop to bottom, clone,. . . etc).

Operation 2 : A petri-net object is created on the middle-workspace by :

- Clicking on the appropriate petri net object represented on the top workspace. A copy of the petri net object is attached to the mouse.
- Transfer the copy to the middle workspace and click at the place where you want it to appears.

These operations can be carried out with all petri net objects on the top workspace.

Note : *After having been pasted on the middle workspace the object can still be moved by selecting and pulling it to a new place.*

Operation 3 : To establish the connection between a transition and a place select the end point of the arrow linked to the transition, stretch it to the place and click on the place.

Note : *A transition has only five input arcs and five output arcs and only the first one of each input and output arrow are visible. Therefore to select an invisible arc click exactly at the same place as the visible arrow.*

Operation 4 : By clicking on each created petri net objects the menu of the object is displayed. The menu offers the user with several possibilities[16] to act on the object such as rotate, change size, color, delete. . . .

Operation 5 : Selecting "table" from the menu of the object displays a table containing all the attributes of the object :

15 Refer to the G2-guide, chapter "Workspaces and the Workspace hierarchy"of the reference manual for G2 version 3.0 (1992) for more information.

16 Refer to the G2-guide subchapter "Working with objects" p324 of the reference manual for G2 version 3.0.

- For event and d-transitions the user can set the "timing" (default value 0) and the "priority" in case of conflict (default value 5).
- For a c-transition the "max speed" attribute has to be set otherwise the default value is 0.
- The "initial marking" attribute of a place represents the number of initial marks in the place (default value 0), it can be set by the operator. The "marking" attribute and the "balance" attribute for c-places are provided only to be consulted, *they do not have to be set.* The "marking" gets the value of the marking in the place at the beginning of the IB-state and the "balance" is the balance of a c-place.
- The attribute of an arc is its "weight" (default value 1).

  Note : *The weight of an arc can be a float only if the arc is connected to a c-place.*

The attributes not described above are used in case of conflict and are explained in the following operation.

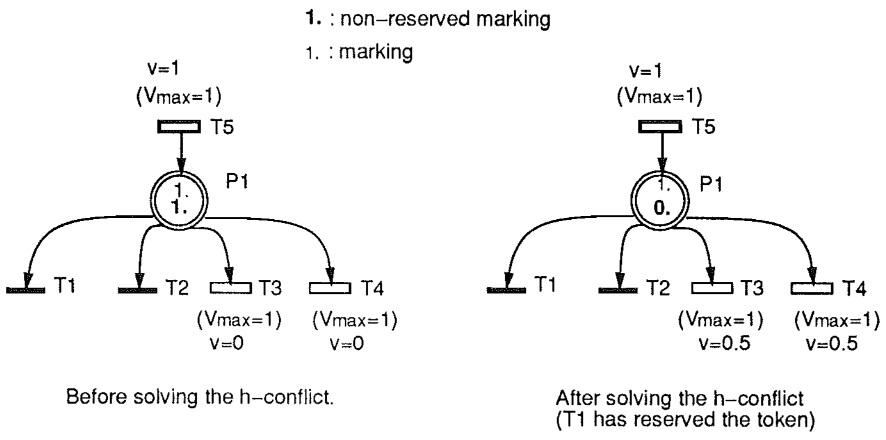
Operation 6 : Look for structural conflicts and choose one of the following predetermined behaviors :

- *Conflict between d-transitions :* They can only be solved by priority, therefore the attribute "priority" of the involved d-transitions has to be set.

  Note : *The lowest priority is 1 and the highest available priority is 1e99.*
- *Conflict between c-transitions :* A predefined behavior has to be determined only if the place involved in the conflict is a c-place. For this class an attribute "conflict case" can be assigned to "priority" (this cause the conflict between the output transitions be solved by priority) or "sharing" (conflict are solved by sharing). If the first solution is chosen different values have to be assigned to the "priority" attribute of the output c-transitions in the same way as d-transitions (refer conflict between d-transitions). If the attribute is assigned to "sharing" the attribute "flow part" of the output c-transitions has to be set. The flow part represents the proportion of the inflow that is going to pass through the c-transition in case of effective conflict.

  Note : *The sum over the flow parts of the output transition must be equal to 1.*
- *Conflict involving c and d-transitions :* In the case of an hybrid structural conflict and if the place is marked the HPN program carries out the calculations on the output transitions by priority order. Therefore different values have to be assigned to the priority attribute of the output transitions (even to c-transitions). Similarly to conflict between d-transitions we do not have to set any attribute of

the place to get this behavior. The attribute "conflict case" is used only if the marking of the place is nil and if there is an effective conflict between output c-transitions (similarly to conflict between c-transitions). Both previous situations can be involved while solving the same hybrid conflict (the "hybrid way"describes at the end of Section 3 to solve an hybrid conflict). The attributes of the example of Figure 36 are set such that the 'hybrid way' is obtained.



1. : non–reserved marking
1. : marking
v=1
(Vmax=1)
T5
P1
1.
1.
T1
T2
T3
T4
(Vmax=1)
v=0
(Vmax=1)
v=0
Before solving the h–conflict.
v=1
(Vmax=1)
T5
P1
1.
0.
T1
T2
T3
T4
(Vmax=1)
v=0.5
(Vmax=1)
v=0.5
After solving the h–conflict
(T1 has reserved the token)

Attributes : Initial marking of P1 =1
Conflict case of P1 = sharing
Priority of T1 = 5
Priority of T2 = 4
Priority of T5 = 5
Flow part of T3 = 0.5
Priority of T3 = 3
Flow part of T4 = 0.5
Priority of T4 = 2

Figure 36. Hybrid conflict solved by the hybrid way

Note :

- *It is impossible to enter fractional numbers for the flow part. Flow parts such as the following 0.333/0.333/0.333 between three c-transitions are an error because the sum over the flow parts is not exactly equal to 1.*
- *Do not forget to set a priority order to the transitions involved in a structural conflict, even if you want to solve it by sharing because in case of insolvable sharing the conflict is solved by priority*[17].
- *To obtain the hybrid way to solve a conflict give the highest priorities to the d-transitions.*

Operation 7 : By selecting the action "create subworkspace" in the popup menu of transitions and places a subworkspace associated with the object is created. It can be used to define actions in relation with the state of the object.
The following rule can be built on the subworkspace of a place P1 :

17 Refer to Section 3.

```
RULE 1

If (the marking of P1 > 10) then
  Inform the operator that ''Overflow!!''
```

It is also possible to build actions to act on the PN. For example rule 2 implies the firing of the event-transition T if the marking in the place P1 is less than 6.5.

```
RULE 2

If (the marking of P1 <= 6.5) and (the fireable of T is true)
then
    Start event-event-trans(T)
```

***Note :** These possibilities require good knowledge of G2 and of the HPN program therefore a non aware user will probably not be able to use them.*

Operation 8 :Name the workspace using the name attribute of the workspace menu and save it by selecting the save-kb action from the main menu.

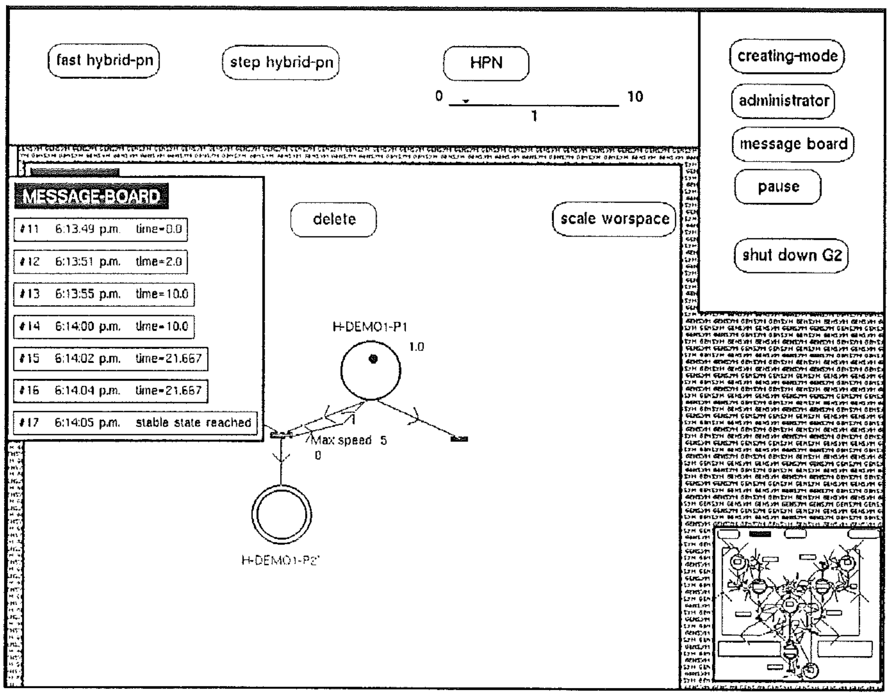
***Note :** It is important to name the subworkspace because otherwise after saving and hiding the workspace it is more difficult to find it again.*

Operation 9 : If when simulating the model the behavior is different from the one expected you have to yourself set the priority order of the places in order to obtain the calculation order that lead to the desired solution.

Operation 10 : Finally to simulate the PN you have to select the button "Running mode" (on the right workspace).

The running mode

The running mode is also composed of three workspaces which due to their location are named top-workspace, right-workspace, and middle-workspace. The running mode is used to start the simulation of PNs that are on an enabled, **visible or hidden**, workspace. Therefore, before starting the simulation of a PN do not forget to check that the workspace where it is located is enabled (for example click on the workspace to display the menu and if the action "desable" belongs to it the workspace is enabled). Reciprocally disable a workspace that is not used by selecting the action "desable" from the workspace menu. Remember that the simulation speed improves as the number of enabled workspaces decreases (because there is less calculations to carry out on one PN than on several ones).
The three main running buttons are situated on the top-workspace (the Fast-hybrid-PN, the Step-hybrid-PN, and the HPN button) whereas buttons to monitor the simulation are located on the right-workspace.



MESSAGE-BOARD

```
#11   6:13.49 p.m.   time=0.0
#12   6:13:51 p.m.   time=2.0
#13   6:13:55 p.m.   time=10.0
#14   6:14:00 p.m.   time=10.0
#15   6:14:02 p.m.   time=21.667
#16   6:14.04 p.m.   time=21.667
#17   6:14:05 p.m.   stable state reached
```

Figure 37. The running mode window

Step hybrid button : The step-hybrid button starts a program that determines the speed state from the marking of the PN, fires all the transitions that have to be fired at *time*, sets the *time* to the time of the next event and stops. Therefore by clicking again on the button the state of the PN can be visualized at the beginning of the following IB-state. This version of the program is useful to construct the evolution graph of PNs.

Note :

- *The time at the beginning of a speed state or a message indicating that the stable state is reached are displayed on the message-board.*
- *When clicking on the button after having reached the stable speed state the marking evolution of the PN is calculated at* time = *1.5*(the last previous* time*).*

The HPN button : The HPN button on the right of the top-workspace is used to simulate in real-time the PN. This means that if the next event time is in 10*s* the program waits 10*s* before automatically changing of IB state (it is not necessary to click again on the button as with the Step-hybrid-PN button). The HPN button can be used in relation with the slider (situated close to it) to regulate the speed of the simulation. For example in the previous case if the slider was on the value 5 the program would have wait for 5*10*s* instead of 10*s*.

Note :

- *It can occur that the theoretical time of the IB-state is shorter than the time to carry out the calculations, therefore it is impossible to run the simulation in real time. In this situation a message appears on the message board to approximately indicate how long the program has been late.*
- *During the time elapsed between two events some new values of the marking are calculated. The number of values depends on how long the program waits between the events.*
- *Changing the speed of the simulation does not interfere with the graph of the marking (the graph is alway graphed with the real time ordinates).*
- *If you modify the value of the slider while simulating, the modification will be taken into account only at the date of the next event time. Therefore it is often quicker to "restart" the simulation with the good speed instead of waiting for the next event time.*
- *This version is the only version that allows you to dynamically fire the event transitions by clicking on them.*

The Fast-hybrid-PN button : The Fast-hybrid-PN button starts a version of the program that carries out all calculations as quick as possible during 30s. The main usage of the program is to quickly get the graph of the markings.
Note :

- *The running time may be less than 30s if the stable speed state is reached before.*
- *The running time can be set by changing the value of the attribute "Uninterrupted procedure execution limit" of the procedure Fast-hybrid-PN.*
- *It is impossible to interrupt the program while running.*

Right workspace buttons : The left workspace contains the following buttons to monitor the simulation :

Shut down G2 : Shut down G2 whithout saving.

Pause button : The pause button stops the program. All graphs and parameters conserve their values. It can be restarted by using the "resume" action from the main menu.
Note : The pause action stops the program but the current time continues to elapse, therefore the time ordinate of the values graphed after a pause (if you resume the program) are false.

Message-board button: The message-board button allows you to visualize or hide the message-board.

Creating-mode button : The creating-mode button permits you to return to the creating-mode.
Note : The button does not reset or stop the program.

Administrator button: The administrator button changes the mode to administrator without reseting the program.

Independently from the available buttons in the running mode, by clicking on a place a graph displaying the marking evolution of the place is automatically created. The graph appears at the left of the screen and is transient (it means that it will be removed when reseting or restarting the program). A solution to make them permanent is to clone the graph and to paste the copy on a middle-workspace.

Note :

- *To get the graph of the marking evolution of a place the place must be named.*
- *To manipulate graphs (for example to graph several marking evolutions on the same chart) refer to the G2 reference manual, version 3.0 (1992), chapter "Display", p122.*

Only the main functions have been presented for both the previous modes, hence some undescribed possibilities may be encountered while using this program. If the problem arises the best solution is to refer to the G2 manual. Note also that the user interface for the HPN program has primarily been developed to present the work realized in G2, therefore nothing prevent the user from creating an unanticipated situation leading to a bug. If the situation occurs restart the program.

Appendix G. The HPN algorithm

The algorithm of the HPN program provided below is a simplified version of the implemented program. For example all parameters used for the graphical animation or for the user's interface have been suppressed. Also the algorithm does not monitor the Event-transitions.

Variable explanations

b_i: The balance of P_i.
m_i: The marking of P_i.
md_i: The non reserved marking of P_i.
w_{ij}: The weight of the arc connecting Place P_i to Transition T_j.
v_j: The old speed of T_j.
v_j' : The new calculated speed of T_j.
The event-time of T_j: This is the time when the transition will be fired.
The event-time of P_i: This is the time when a c or h-event may occurs in P_i.
C-event-list : List of all the places where a c-event may occurs.
H-event-list : List of all the places where a h-event may occurs.
D-event-list : List of the transitions that will be fired.
List : List of all the places to update.

Global variables

time : The current time.
Δ : Time equal to how long an IB-state lasts.
New-speed-state : The variable is 'true' if a new speed state must be examined, otherwise 'false'.
End-simu : The variable is 'true' if a stable IB-state has been reached.
Start-phase : Record the beginning time of the IB-state.

Main procedure : HPN

```
Begin
    Call Init-HPN();
E1  $time \leftarrow \Delta + time$;
    Call Update-marking();
    Call Update-event();
    If new-speed-state = true then
       Call Init-for-new-speed-state();
    End if
    Call Update-trans-within-input-places();
    Call Update-place-to-update();
    Call New-balance();
    Start-phase$\leftarrow time$;
    New-speed-state$\leftarrow$false;
    Call Next-event-time();
    If End-simu = false then
        Goto E1;
    End if
End
```

Procedure Init-HPN

```
Begin
$time \leftarrow 0$;
$\Delta \leftarrow 0$;
Start-phase$\leftarrow$0;
New-speed-state $\leftarrow$false;
End-simu $\leftarrow$false;
For each d-transition $T_j$ do
    If there is no input places then
        The event-time of $T_j$ $\leftarrow$the timing of $T_j$;
        Insert in the D-event-list;
    End if
End for
For each c-transition $T_j$ do
    $v_j \leftarrow 0$;
    $v_j' \leftarrow 0$;
    If there is no input places then
        $v_j \leftarrow V_{jmax}$;
        $v_j' \leftarrow V_{jmax}$;
        Insert the non marked output places of $T_j$ in List;
    End if
End for
For each place $P_i$ do
    $b_i \leftarrow 0$;
    If $m_i \neq 0$ then
        Insert $P_i$ in List;
    End if
End for
End
```

Procedure Update-marking

```
Begin
For each c-place do
    $m_i \leftarrow m_i + b_i*(time$ - start-phase);
    $md_i \leftarrow md_i + b_i*(time$ - start-phase);
End for
End
```

Procedure Update-event

```
Begin
Repeat until the number of elements in C-event-list = 0
```

```
          If $time$= The event-time of C-event-list[0] then
            New-speed-state ←true;
          End if
          Remove C-event-list[0];
End repeat
Repeat until the number of elements in H-event-list = 0
          If $time$= the event-time of H-event-list[0] then
            Insert H-event-list[0] in List;
          End if
          Remove H-event-list[0];
End repeat
Repeat until (the event-time of D-event-list[0]>$time$)
          Call Fire-transition(D-event-list[0]);
          Remove d-event-list[0];
End repeat
End
```

Procedure Fire-transition T_j:
This procedure carries out the firing of T_j and inserts each input places in List. If the type[18] of the output place is "d-place" inserts it in List otherwise (type "c" or "h-place") concludes that New-speed-state is true.

Procedure Init-for-new-speed-state

```
Begin
For each place $P_i$ such as the type of $P_i$ ="c-place" or "h-place" then
      If $md_i>$ 0 then
         Insert $P_i$ in List;
      End if
End for
For each c-transition $T_j$ do
      $v_j$←0;
      $v_j$'←0;
      If there is no input place then
         $v_j$←$V_{jmax}$;
         $v_j$' ←$V_{jmax}$;
         Insert the non marked output transitions of $T_j$ in List
      End if
End for
End
```

Procedure Update-trans-within-input-place

```
Begin
For each d-transition $T_j$ such as there is no input places do
```

18 Refer to section 2

```
    If $T_j$ does not already belong to D-event-list then
      The event-time of $T_j$ $\leftarrow$ $time$+ the timing of $T_j$;
      Insert in D-event-list;
    End if
End for
End
```

Procedure Update-place-to-update

```
Begin
Repeat until the number of elements in List = 0
    If the available marking (=$md_i$) of List[0] > 0 then
      For each output transition (T) of List[0] (They are selected
      by priority order) do
          If T is a d-transition then
            h-conflict $\leftarrow$the value returned by the function H-conflict;
            If h-conflict = false then
              Call Update-d-transition(T);
            End if
          Else
            If the available marking of List[0] > 0 then
              Call New-speed(T);
            Else
              If already-solve=false then
                Call Weakly-enable(List[0]);
                already-solve $\leftarrow$true;
              End if
            End if-else
          End if-else
      End for
      Call Check-output-c-transitions(List[0]);
    End if
    If the available marking of List[0] = 0 then
      Call Weakly-enable(List[0]);
      Call Check-output-c-transitions(List[0]);
    End if
End repeat
End
```

Function H-conflict :
This function checks that the d-transition can reserve the marking if necessary[19].

Procedure Update-d-transition T_j:

19 Refer to section 3

Update-d-transition checks that the transition is fireable. If the transition is fireable the reservation of the marking is carried out and T_j is inserted in D-event-list after having set the correct value for its event-time (*time*+ the timing of T_j).
Note that if after reservation the remaining available marking = 0 and the type of the place is "h-place" then the procedure Init-for-new-speed-state is started.

Procedure New-speed for T_j:
New-speed carries out the calculation of the speed of a c-transition T_j (update v_j').

Procedure Weakly-enable of P_i:
Weakly-enable determines the speed of the outout c-transitions of P_i. In case of effective conflict it solves it by priority or sharing according to the value of the attribute "Conflict case" of P_i(update v_j').

Procedure Check-output-c-transitions for P_i:
Check-out- put-c-transitions tests if $v_j' \neq v_j$ for the output c-transitions of P_i. If the previous condition is met then all the output non marked places of T_j are inserted in List and v_j' is assigned to v_j.

Procedure New-balance

```
Begin
For each c-place $P_i$ do
    Calculation of the balance of $P_i$;
End for
End
```

Procedure Next-event-time

```
Begin
For each c-place $P_i$ such as $b_i$ <0 do
    The event-time of $P_i \leftarrow time - \frac{md_i}{b_i}$;
    Insert $P_i$ in C-event-list;
End for
For each d-transition $T_j$ that does not have reserved any markings do
    $d \leftarrow$The maximum over all the input c-places of $T_j$ of $\frac{w_{ij}-md_i}{b_i}$;
    For each input c-place $P_i$ such as $b_i > 0$ do
        The event-time of $P_i \leftarrow d$;
        Insert $P_i$ in H-event-list;
    End for
End for
$\Delta \leftarrow$The minimum over the event-time of all the elements of
C, H, and D-event-list;
```

```
If $\Delta$ has no value or greater than 1e99 then
    End-simu $\leftarrow$true;
End if
End
```

Appendix H. Notation

v_j is the instantaneous speed of T_j.

V_{jmax} is the maximal speed of T_j.

$^\circ T_j$ is the group of the input places of T_j.

T_j° is the group of the output places of T_j.

$^\circ P_i$ is the group of the input transitions of P_i.

P_i° is the group of the output transitions of P_i.

b_i is the balance of P_i.

m_i is the marking of the place P_i.

md_i is the no reserved (or the available) marking in place P_i.

w_{ij} is the weight of the arc connecting P_i to T_j.

Inflow of P_i is the sum over the input transitons T_j of P_i of v_j.

Outflow of P_i is the sum over the output transitions T_j of P_i of v_j.