IFAC

International Federation of Automatic Control

SOFTWARE FOR COMPUTER CONTROL

Preprints of the 3rd IFAC / IFIP Symposium
Madrid, Spain
5 - 8 October 1982

Editors

G. Ferraté
Barcelona, Spain

E.A. Puente
Madrid, Spain.

PREPRINTS

SOFTWARE FOR COMPUTER CONTROL

Preprints of the 3rd IFAC / IFIP Symposium.

Madrid, Spain

5 - 8 October 1982

Editors

G. FERRATE

Barcelona, Spain.

E.A. PUENTE

Madrid, Spain.

Published for the

INTERNATIONAL FEDERATION OF AUTOMATIC CONTROL

First edition 1982

Printed in Spain by Gráficas Rublán, Sevilla

Depósito Legal: SE-334-1982.

3rd IFAC/IFIP SYMPOSIUM ON SOFTWARE FOR COMPUTER CONTROL

PROGRAMMABLE LOGIC CONTROLLERS AND PETRI NETS: A COMPARATIVE STUDY

M. Silva & S. Velilla

Departamento de Automática, Escuela Superior de Ingenieros Industriales. Universidad de Zaragoza.

Abstract. In a Petri Net (PN) model of a system, a small subset of transition can usually be fired. Therefore several special Programmable Logic Controllers (PLCs) have been proposed looking for more performant simulation schemas. Some of them are general purpose microcomputer-based. Others are constructed using specialised microprocessors (microprogrammed or not).

This paper presents a conceptual framework to systematize concepts introduced in different PLCs specifically designed for safe (1-bounded) PN simulation. To compare their "conceptual" performances, basic proposed schemas (and other new ones) were programmed on a same microcomputer (M 6801). The technique used for comparison of performances consists of: (1) building different data structures to represent PN and its marking and (2) building performance models using a small set of parameters that caracterizes the complexity of PN-models.

It is shown that there is no "optimum" schema. The "best" one is a function of the net to be realised.

Keywords. Computer control; machine oriented languages; microprocessors; programming language; programmable logic controllers; performances; Petri nets.

INTRODUCTION

The advantages of programmed logic make possible the growing interest in a special class of computers for system control named Programmable Logic Controllers (PLC). On the other hand, the complexity of modern logic control systems contributes to the increasing use of Petri Nets (PN) as a modelling tool. In this paper safe (1-bounded) PN and special PLCs are considered.

PLCs are a very simple class of computers that work cyclically (a cycle is named a Treatment Cycle, TC). In classical PLCs, the simulation of logical automatisms modeled with PN can be done in a very easy way: programming all transitions of the net in such a manner that every logic equation will be executed in each TC.

In a PN model of a system, only a small subset of transitions can usually be fired. Therefore, several special PLCs were proposed to make the simulation more performant.

Proposed systems vary from general purpose based computers [(Silva & David, 1979; Chocrón, 1980)] to special purpose based computers [microprogrammed as in (Daclin, 1976) or in (Tafazzoli, 1979), or not microprogrammed as in (Defrenne, 1979) or in (Silva & Velilla, 1980)].

This paper focusses on the two main design problems for PN based PLCs: (1) the definition of PN simple specification languages and (2) the choice of an adequate PN internal representation (data structure) and an algorithm to interprete the data structure in a correct and efficient manner.

The choice mentioned in the last point relates directly to memory ocupation and execution time of a TC. A conceptual framework is presented (1) to systematize concepts introduced in different PLCs speciafically designed for safe PN simulation, and (2) to "compare" their respective "conceptual performances". All presented simulation schemas(and others) have been programmed on a M 6801.

It will be assumed that readers are familiar with basic PN terminology (Daclin, 1976; Peterson, 1981).

PETRI NETS MODELS ESPECIFICATION LANGUAGE

To simplify the programming of PN models, the language should be non-procedural (non-algorithmic : the lexicographic order in wich the system is specified is not important).

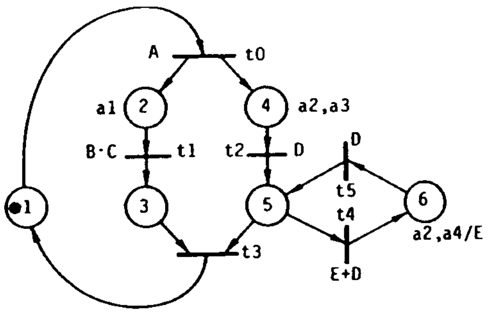


Fig. 1 A safe PN model ({A,B,C,D,E,F} are input variables and {a2,a4,a6} actions).

The main advantages of this decision are:(1) direct specification (there is not translation by the designer), (2) programming hazards need not be solved by the designer (they must be solved by the PLC system), (3) specification may be validated (Martínez & Silva,1982) and (4) it is very easy to give a redundant specification to avoid errors in the data input process.

Languages may be on a "fill-in-the blank" level, or on a "higher" one. There PN model structure and interpretation may be specified separated or mixed. We will speak about two different cases.

Fill-in-the-blank and separated. The structure of the PN is specified by giving, for each transition; (1) its input and output places and (2) the associated event. The interpretation is specified by defining (1) events and (2) actions associated with places. The initial marking is defined after the structure specification.

TABLE I A specification of Fig.1 PN model

```
STRUCTURE
  t0:p1/p2,p4 $ e0;    t1:p2/p3 $ e1;
  t2:p4/p5 $ e2;       t3:p3,p5/p1;
  t4:p5/p6 $ e4;       t5:p6/p5 $ e2 $
INITIAL MARKING   p1 $
EVENTS
  e0:=A; e1:=B·C; e2:=D; e4:=E+D̄  $
ACTIONS
  p2:a1;  p4:a2,a3;  p6:a2,a4/E $
```

Table 1 shows a specification of the Fig.1 PN model. If it is judged interesting to make the data input process redundant, the structure should be defined place by place, giving its input and output transitions.

For the Fig.1 PN we may write (it is possible to fuse this phase with the ACTIONS phase):

```
p1:  t3/t0;  p2:t0/t1;
p3:  t1/t3;  p4:t0/t2;
p5:  t2,t5/t3,t4;  p6:t4/t5 $
```

"Higher level" and mixed. With this kind of languages it is possible to use symbolic names for places and transitions. The structure is usually defined transition by transition with its actions and events. Actions associated with places are defined place by place as before.

Some other important facilities to consider here are the specifications of Macroplaces, Macrotransitions, Subprograms, ... (Martínez &Silva,1982).

From the translation point of view, the kind of language to be selected is a function of the implementation method to be used.

SOME PREVIOUS CONSIDERATIONS ABOUT IMPLEMENTATIONS.

To implement specialized on safe PN simulation PLCs, there is a wide range of freedom. That will be briefly considered. The conditions under which different realizations will be compared are also defined.

PLC simulation power. There are several PLC systems in wich events are only a variable (complemented or not) and actions are only unconditionnaly associated with places. At the other extreme there are PLC systems in which events and conditions may be specified by any boolean function of external and internal variables. Conditional actions can be employed.

In the following paragraphs we will consider any boolean function as defining events but only inconditional actions as being associated with places and/or transitions. With this restriction, Fig. 2 shows how it is possible to consider that actions are only associated with transitions. This is a very im-

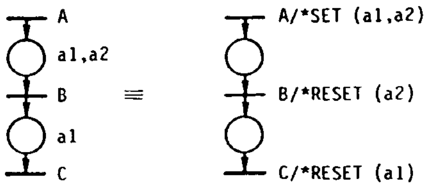


Fig. 2 Inconditional actions associated to places can be represented around transitions

portant decision for the simplification of TC, because most PLCs that permit conditional actions work in two phases: (1) marking evolution and generation of actions associated to transitions and (2) conditional action generation. Only one special PLC (Silva & David, 1979) works with conditional actions in a single phase. It is based on a slightly more complicated but more performant simulation schema.

Computer Support. Some PLCs work on special hardware. For example, COLERES (Daclin, 1976) is constructed around a microprogramming sequencer (INTEL 3000). Defrenne (1979) has built a special hardware using a specialized microprocessor (MOTOROLA 14500 B) and external hardware. Other PLCs use general purpose microcomputers and are specialized by software only. For this type there are several differences because some systems execute, after a macroexpansion process, the code of the support microcomputer. At the other extreme an interpreter or simulator reads a data structure representing the PN and its interpretation.

To compare the conceptual performances of special PLCs, the basic proposed schemas will be considered by using a general purpose microcomputer with interpreters (each reading a data structure) or executing directly code (in one case, a hardware SCANNER will be added to the system).

Synchronous or non-synchronous interpretation. Most proposed PLCs take inputs when a TC starts. This is done to avoid hazards due to value changes during a cycle. The evolution of some systems is defined as synchronous if PN marking evolues globally at the TC end or just before conditional action calculations. A PLC is synchronous even if there is no real time clock to start the next TC.

Non-synchronous simulation schema can cause problems. For example, the Fig. 3 PN simulation will give a non-safe intermediate marking if t_1 is treated before t_2. Also, it is interesting to note that with non-synchronous evolutions the final marking depends on the order in which transitions are considered by the interpreter. Synchronous interpretation is always longer than non-synchronous interpretation.

Internal representation. Data structure representing the PN may use matrices or lists. Lists may be organized in different fashions as will be seen later on.

Basic schemas will be compared in the following paragraphs.

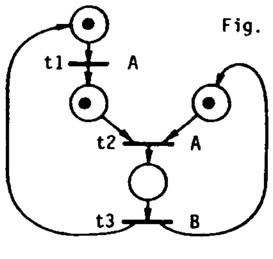


Fig. 3 A PN model that is safe due to the interpretation.

MATRIX REPRESENTATION OF PETRI NETS

In this paragraph we present a data structure and a driving algorithm to simulate PN. The basic schema will be complicated a little more in order to obtain faster TC.

A Matrix-Based Basic Schema. PN structure will be represented by two matrices: (1) pre-matrix, E_{nxm} (n is the number of places and m the number of transitions) and (2) flow-matrix, F_{nxm}. They are defined as:

- IF p_i is an input place of t_i THEN $E_{ij} = 1$ ELSE $E_{ij} = 0$
- $F_{ij} = \alpha + \beta$, where:

$$\begin{cases} \text{IF } p_i \text{ is an input place of } t_j \text{ THEN } \alpha = 1 \\ \qquad\qquad \text{ELSE } \alpha = 0 \\ \text{IF } p_j \text{ is an output place of } t_j \text{ THEN } \beta = -1 \\ \qquad\qquad \text{ELSE } \beta = 0 \end{cases}$$

Let us represent the marking by the boolean vector M_{px1}, and let us define A^j the j-th column of matrix A. With this notation it is easy to show that:

(a) t_j is enabled by M iff $E^j \Rightarrow M$ (1)

(b) If t_j is fired, the new marking, M^+, is: $M^+ = M + F^j$ (2)

Now: $[E^j \Rightarrow M] \equiv [E^j \cdot M = E^j] \equiv [E^j \cdot \overline{M} = 0]$ (3)

Then t_j is enable iff $E^j \cdot M = 0$. Now let us define matrix $\emptyset$ by:

IF $F_{ij} \neq 0$ THEN $\emptyset_{ij} = 1$ ELSE $\emptyset_{ij} = 0$

Since M^+ and M are boolean, eq. (2) may be rewritten as ($\oplus$ = exclusive-OR):

$$M^+ = M \oplus \emptyset^j \quad \text{or} \quad \overline{M^+} = \overline{M} \oplus \emptyset^j$$

In conclusion, all operations are boolean vectorized, and thus PN simulation may be performant in most general purpose microcomputers.

A faster Matrix-Based-Schema. To accelerate the simulation it is possible to think of executing iterations only for enabled transitions (they drive the simulation process). Since a transition may have several input places, it takes a long time to obtain the transition enabled boolean vector, θ^*.

To reduce the enabling concept to a boolean one, each transition will be "represented" by one of its input-places (Silva & David, 1979).

To enable t_j, represented by p_i, it is necessary (but not sufficient) that p_i be marked. Let us define the boolean vector θ_{mx1} as: "$\theta_j = 1$ iff the place (p_i) that represents t_j is marked [$M(i) = 1$]".

In the above presented conditions, when a transition is fired it is very easy to obtain the new vector, θ^+.

Let us define the boolean matrices A_{mxm} and B_{mxm} where: (1) $A_{ij} = 1$ iff t_i is represented by a t_j output-place. (2) $B_{ij} = 1$ iff it is also represented by the place that represents t_j.

When t_j is fired, it is easy to see that

$\theta^{+} = \theta \oplus A^{j} \oplus B^{j}$.

To simplify calculations and reduce memory occupation we will use only matrix D, where $D^{j} = A^{j} \oplus B^{j}$ and then $\theta^{+} = \theta \oplus D^{j}$. Table 2 shows the new simulation algorithm.

Performances between basic and faster schemas will be shown later. The faster schema has ≃50% more memory occupation (matrix D and vectors θ and θ_{aux}) and its execution time is ≃60%.

To reduce memory occupation, list representations may be used because E, F and D are usually sparse-matrices.

LIST-BASED REPRESENTATIONS OF PETRI NETS

The memory occupation of matrix representations is mainly proportional to nxm. With list-based representations, memory occupation will usually be linear on m and n.

List-Based Basic Schema. The PN structure is represented by a transition list. Each element list describes: (1) its input and output places; and (2) associated events and actions

The marking is represented by a boolean vector, M (for synchronous simulation a M_{aux} vector may be used). Net simulation is done by considering transition by transition. The simulation algorithm is very simple, but execution time will be very important because it is necessary to test all the PN transitions. This schema has been adopted by Tafazzoli (1979) to simulate Capacity PN. He has developed a special microprogrammed computer. Some faster list-bases schemas will be considered below.

P-T List and Marking Driven Simulation schemas. To avoid the non-performant simulation schema presented in the preceding subparagraph, transitions to be tested will be restricted to those for which their representing place is marked. Figure 4a shows a data structure associated

TABLE 2 Control algorithm for matrix represented PN using the faster schema

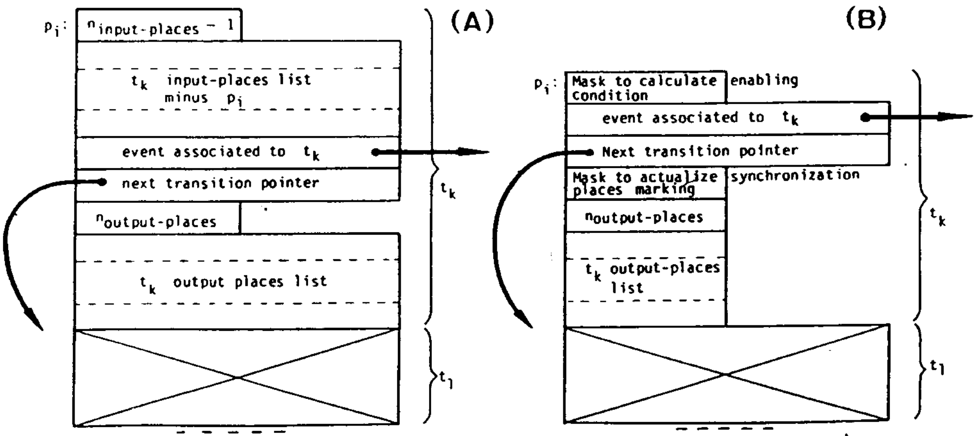
```
(1) Data input;
(2) FOR j:=1 TO m DO
      IF $\theta_j$=1 {test on a boolean variable}
        THEN | IF $M \cdot E^j$ =0
             |   THEN | calculate event, $e_j$;
             |        | IF $e_j$=1
             |        |   THEN | generate $t_j$-actions;
             |        |        | $M_{aux} := \overline{M}_{aux} \oplus \emptyset^{j}$;
             |        |        | $\theta_{aux} := \theta_{aux} \oplus D^{j}$
(3) $\overline{M} := \overline{M}_{aux}$ ; $\theta := \theta_{aux}$ ;
(4) Actions output ;
```

with each place. It consists of a represented transition list. The data structure associated with p_i will be inspected only if p_i is marked. In this case only a subset (usually small) of transitions will be considered.

In a first schema, it is possible to assume that marked places are selected by scanning a marking vector. This scanning may be done: (1) by the microprocessor (by serializing it with the selected-places treatment), or (2) by a simple SCANNER-processor (Daclin, 1976; Silva & Velilla, 1980).

An alternative to scanning techniques (Silva & David, 1979) is to use a pointer for each place that represents at least one transition. Each place having an output transition not represented by the place will be implemented in a marking boolean subvector (usually very small). These places will be named synchronization places (they will be considered as software flags). To avoid redundant implementations of the marking of any place (a place may simultaneously serve the function of representation and synchronization), it is interesting to choose two disjointed subsets ($P_r \cup P_s = P$, $P_r \cap P_s = \emptyset$).

This condition is usually verified, but when this is not so, for a given PN, then there



(Note: If next transition pointer = 0000, there is no more transition represented by p_i)

Fig. 4 Two place-transition lists PN structure representation.

are always equivalent nets for which the condition holds (see Fig. 5).

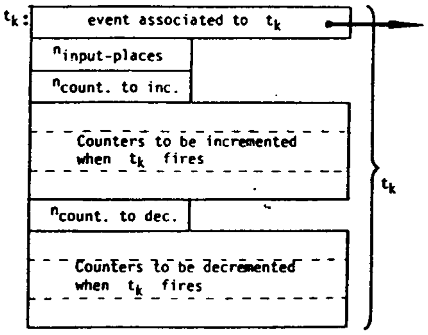
Figure 4b shows another form to represent PN structures. The simulation algorithm takes pointers corresponding to representing marked places from a list (normally a stack) and builds the list (stack) for the next TC. When the TC ends, the rôle of the two lists is changed: the built list becomes the treatment list while the older list is forgotten and a new one will be built.

As a general comment, it is interesting to point out that partitions between representation and synchronization places allow better performances, but regularity in place representation and treatment is lost.

T-List and Enabled Transition Driven Simulation Schemas. This last schema may be viewed as an improvement of the "List Basic Schema" because only data structure associated with enabled transitions will be considered in a TC. Then as in the former schema, execution-time will be reduced.

To make a performant simulation, a counter will always be associated to each transition (Chocron, 1980). This will count the number of input places that are not marked. Obviously t_j is enabled iff $C_j = 0$. Figure 5 shows the data structure associated to a transition. The simulation algorithm proceeds as follows:

(1) When an enabled (possibly fired) transition treatment is finished, then another one is looked for.
(2) When a transition t_k is fired, then: (a) C_k is set to the number of t_k input places (this is correct because only safe PNs are considered); (b) counters associated with the other output transitions of the fired transition input places will be incremented (because the firing of t_k increments the number of their non-marked input places); and (c) counters associated with the output transitions of the output places of t_k will be decremented.



t_k:
event associated to t_k
$n$input-places
$n$count. to inc.
Counters to be incremented when t_k fires
$n$count. to dec.
Counters to be decremented when t_k fires
t_k

Fig. 5 A transition list definition for representing PN structure.

As when the simulation schema is marking-driven, an enabled transition may be searched for: (1) by scanning the counter vector C; and (2) by using two lists (stacks). An important difference between the latter schema and the representing synchronization marking driven schema is that vector C can not be eliminated, because it is essential to know when a transition reaches or leaves the enabled state. "t_j"-pointer, or its transition number, will be added to the enabled transition list when decrementation of C_j gives $C_j = 0$.

Finally, it is interesting to note that, in fact, vector C defines indirectly the marking of the PN. In the schemas defined in this subparagraph, marking does not appear directly. It will state an important problem when conditional actions are associated with places. In this case a double marking representation is normally used: (1) marking vector M; and (2) counter vector C. Obviously this redundant schema will reduce performances.

PERFORMANCE COMPARISON

The approach used to make performance comparison consists in building some simple evaluation models using a small set of parameters. These parameters should characterize the implementation complexity of the safe PN. The approach to the construction of the performance models is presented in (Silva, 1979). In this paper, we will only analyze results obtained for the six PLC classes studied below (the six columns in Table 3). In fact, ten different special PLSCs have been programmed on a M6801.

To avoid long formula manipulations, comparison will be made by using four test cases (in table 3 rows). The six parameters used to define "the complexity" of a PN are: (1) the number of places, n, and transitions, m; (2) the mean number of input places of a transition, n_{it}, and of input transitions of a place, m_{ip}; and (3) for a given TC, the number (a maximum estimation) of marked places, n_m, and of fired transitions, m_f.

Table 3 shows performance estimation. In this table only PN structure and marking are considered (i.e.: event calculation and action generation are not considered at all). It may be easily seen that matrix-based methods are not performant. They may be of some interest for very small PN ($n < 16$, $m < 16$) [memory occupation and execution-time grows mainly as $K_1\, n.m + K_2\, n + K_3\, m + K_4$]

When comparing only list-based methods it is clear that stack-driven simulation schemas are better, except when a hardware scanner is used. It may also be seen that, in case 2 and for data structure interpretation, the "T-list & stack" method is slightly more performant than marking-driven solutions. It should be easily understood because case 2 PN is a Marked Graph (dual of a State Machine). As a general com-

TABLE 3 Execution time in machine cycles (M6801) and memory ocupation estimation

Data Structure / Interpretation \ Executable Code (+Hardware Scanner)		Matrix		P-T List &		T List &	
		Basic	Faster	Scanning	Stacks	Scanning	Stacks
1 STATE GRAPH	n=23, m=32, n_{it}=1, m_{ip}=1.4, n_m=3, m_d=2	1749 / —	1172 / —	587 / 394 (104)	364 / 180	587 / —	471 / 315
2 MARKED GRAPH	n=32, m=23, n_{it}=1.4, M_{ip}=1, n_m=4, m_d=1.5	1577 / —	1030 / —	715 / 523 (128)	356 / 187	425 / —	347 / 249
3 PN	n=m=32, n_{it}=1.4, m_{ip}=1.4, n_m=4, m_d=2	2147 / —	1211 / —	832 / 533 (138)	436 / 210	609 / —	487 / 333
4 PN	n=m=100, n_{it}=1.4, m_{ip}=1.4, n_m=12, m_d=6	17295 / —	8860 / —	2585 / 1557 (362)	1249 / 530	1829 / —	1339 / 872
Comparative performances over a (1),(2) and (3) mix		9.45 / —	5.9 / —	3.7 / 2.5 (0.64)	2 / ①	2.8 / —	2.26 / 1.55
Memory ocupation (bytes) per transitions in list-repres.		—	—	11.5 / 15.5	10.0 / 26.5	12.0 / —	10.5 / 31

ment, it may also be stated that "usually" marking-driven simulation schemas are more performant than transition-driven ones. In this manner, when a "T-list & stack" schema is used, before every push or after every pull of a transition pointer a zero test on the corresponding counter-vector element is needed (operations will be realised only if the transition is enabled).

For list-based schemas, memory occupation is "mainly" linear with m (see Table 3,last row). At this point, it may be stated, for examole, that the "P-T list & Stacks" execution code version is twice as fast than the interpreted one, but consumes about 2,65 times more memory space.

CONCLUSIONS

We will concentrate on the extensibility of the list-based representation principles for realising: non-safe PN, PN based "self-testing", conditional actions and translation from specification language.

Transition-based presented schemas can not be extended for non-safe PN when the counter-vector is used. Self-testing techniques (usually marking-based) and conditional actions associated to places are difficult to realize because we have not a direct representation of the marking. Translation is easy.

Place-based presented schemas may be easily extended for non-safe PN and conditional actions associated to places. The stack-based method is difficult for self-testing PN-based procedures. These reasons together with perfomance results, make place-based schemas "better" in most cases. The Stack-based method needs a more complicated translation process.

Note that for list represented methods, table 3 gives performances for non-synchronous evolutions. Stack-based methods will maintain them for synchronous evolutions,but scanning based methods will be slower. At this point,it is interesting to note that the use of microprocessors with autoincrement addressing or two stack pointers will make stack-based schemas much more performant. For that reason,the study is being partially remade by using M6809, LSI-11 or similar microprocessors.

Finally, it is important to recall that event calculations and action generation (a usually very important part of each TC) will lend considerable uniformity to the performances rendered by most of the considered methods.

REFERENCES

Chocron, D.(1980). Un Système de Programmation par RdP de Controleurs Industriels. Master Comp. Sc.,Montreal.

Daclin, E. & M.Blanchard (1976). Synthèse des Systèmes Logiques. Ed. Cepadues. Toulouse. pp 201-214.

Defrenne, J. & co-workers (1979). Gestion Rapide des RdP par μP monobit. MIMI 79, Zürich, pp.62-66.

Martínez, J. & M. Silva (1982). A package for computer design of concurrent logic control systems. SOCOCO-82, Madrid.

Peterson, J.L. (1981). Petri Net Theory and the Modelling of Systems. Prentice-Hall, Englewood Cliffs, N.J.

Silva, M. & R. David (1979). Synthèse programée des automatismes logiques décrits par RdP. RAIRO-Automatique, vol. 13, n°4, pp 369-393.

Silva, M. (1979). Evaluation des performances des applications temps réel de type logique. MIMI 79, Zürich, pp 152-157.

Silva, M. & S. Velilla (1980). Sistema especializado en la simulación de redes de Petri sanas. I Simp. Nacional sobre Modelado y Simulación (IFAC).Sevilla.pp 81-88.

Tafazzoli, M.E. (1979). Realisation d'un interpreteur matériel de réseau de Pétri. Thèse 3eme. Cycle.Nice.