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.
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
3rd IFAC/IFIP SYMPOSIUM ON SOFTWARE FOR
COMPUTER CONTROL

Sponsored by
IFAC Technical Committee on Applications
IFAC Technical Committee on Computers
IFAC Technical Committee on Education

Co-Sponsored by
IFIP Technical Committee on Computer Applications in Technology

Organized by
Comité Español de la IFAC
Departamento de Ingeniería de Sistemas y Automática de la
Universidad Politécnica de Madrid

International Program Committee (IPC)
G. Ferraz, E (Chairman)
P. Albertos, E
A. Bazan, MEX
C.M. Doolittle, USA
P. Etzer, D.
D.G. Fisher, CDN
J. Garlier, H
R. Isermann, D
P.M. Larsen, DK
M. Mansour, CH
J.S. Meditch, USA
R. Mezencev, F
M. Novak, CS
E A Puentes, E
V. Strejc, CS
B. Tamm, SU
E.A. Trakhtengerts, SU
J D N Van Wyk, ZA
T.J. Williams, USA

National Organizing Committee (NOC)
E A Puente (Chairman)
M. Alique
R. Aracil
A.J. Avellano
L. Basañez
E. Bautista
J.G. Bernaldo de Quirós
E.F. Camacho
M. Collado
J.M. Coronado
R. Huber
J.A. Martín-Pereda
M. Méndez
P. de Miguel
J.A. de la Puente
R. Purjañer
A. Rodríguez
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 Control-
ners (PLCs) have been proposed looking for more performant simulation sche-
mas. 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 intro-
duced in different PLCs specifically designed for safe (1-bounded) PN simula-
tion. 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 dif-
ferent data structures to represent PN and its marking and (2) building per-
formance models using a small set of parameters that characterizes the com-
plexity 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 possi-
ble the growing interest in a special class
of computers for system control named Progra-
mable Logic Controllers (PLCs). 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 Treat-
ment Cycle, TC). In classical PLCs, the si-
mulation of logical automatism modeled with
PN can be done in a very easy way: program-
ming all transitions of the net in such a man-
ner that every logic equation will be execu-
ted in each TC.

In a PN model of a system, only a small sub-
set of transitions can usually be fired.
Therefore, several special PLCs were pro-
posed to make the simulation more performant.

Proposed systems vary from general purpose
based computers (Silva & David, 1979; Cho-
cron, 1980) to special purpose based com-
purers (microprogrammed as in (Daclin, 1976) or
in (Tafazzoli, 1979), or not microprogrammed
as in (Defrenne, 1979) or in (Silva & Vel-
illa, 1980).

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

The choice mentioned in the last point re-
lates directly to memory occupation and execu-
tion time of a TC. A conceptual frame-
work is presented (1) to systematize concepts
introduced in different PLCs specifically
designed for safe PN simulation, and (2) to
"compare" their respective "conceptual per-
fomances". All presented simulation sche-
mas (and others) have been programmed on a
M 6801.

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

PETRI NETS MODELS SPECIFICATION
LANGUAGE

To simplify the programming of PN models, the
language should be non-procedural (non-algo-
riticm: the lexicographic order in which
the system is specified is not important).
The main advantages of this decision are: (1) direct specification (there is no 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 1 A specification of Fig. 1 PN model**

<table>
<thead>
<tr>
<th>STRUCTURE</th>
<th>EVENTS</th>
<th>ACTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0:p_1/p_2, p_4 \equiv e_0$; $t_1:p_2/p_3 \equiv e_1$; $t_2:p_5/p_6 \equiv e_2$; $t_3:p_3/p_5/p_1$; $t_4:p_5/p_6 \equiv e_3$; $t_5:p_6/p_5 \equiv e_2$ $</td>
<td>e_0:=A; e_1:=B-C; e_2:=0; e_4:=e_1 \equiv 0$ $</td>
<td>p_2:=a_1; p_4:=a_2,a_3; p_5:=a_4/a_5/E $</td>
</tr>
</tbody>
</table>

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).

$p_1$: $t_3/t_0; p_2:=t_0/t_1$; $p_3$: $t_3/t_3; p_4:=t_0/t_2$; $p_5$: $t_2, t_5/t_3, t_4; p_6:=t_4/t_5$ $\equiv$ "higher level" and mixed. With this kind of language, 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, Macrottransitions, 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 which events are only a variable (complemented or not) and actions are only unconditionally 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 unconditional 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 important decision for the simplification of PLCs, 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, COLORES (Daclin, 1976) is constructed around a microprogramming sequencer (INTEL 3000). Dovation (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 phase 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 evolves globally at the PLC end or just before conditional action calculations. A PLC is synchronous even if there is no real time clock to start the next cycle.

Non-synchronous simulation schema can cause problems. For example, the Fig. 3 PN simulation will give a non-safe intermediate marking if t2 is treated before t1. 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 fashion as will be seen later on.

Basic schemas will be compared in the following paragraphs.

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

- If p_i is an input place of t_j, THEN F_{ij} = 1
  ELSE F_{ij} = 0

- F_{ij} = k lead, where:
  \{ IF p_i is an input place of t_j, THEN a = 1
  ELSE a = 0
  IF p_j is an output place of t_i, THEN a = -1
  ELSE a = 0

Let us represent the marking by the boolean vector M_{ij}, 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_{jA} \cdot M (1)
(b) If t_j is fired, the new marking, M' is:

\[ M' = M + F_{ij} \]

Now:

\[ E_{jA} \cdot M - E_{jA} \cdot F_{ij} = E_{jA} \cdot M - 0 = E_{jA} \cdot M \]

Then t_j is enabled iff E_{jA} \cdot M = 0. Now let us define matrix \( B \) by:

- IF F_{ij} = 0 THEN \( B_{ij} = 1 \) ELSE \( B_{ij} = 0 \)

Since \( M' \) and \( M \) are boolean, equation (2) may be rewritten as (exclusive-OR):

\[ M' = M \oplus B_{ij} \] or \[ M' = M \oplus B'_{ij} \]

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

A faster Matrix-Based Scheme. To accelerate the simulation it is possible to think of performing 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, \( \Theta \).

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 \( M_{max} \) as:

\[ M_{max} = 1 \text{ iff the place } (p_i) \text{ that represents } t_j \text{ is marked } [M(i)\times 1] \].

In the above presented conditions, when a transition is fired it is very easy to obtain the new vector, \( \Theta \).

Let us define the boolean matrices \( A_{max} \) and \( B_{max} \) where:

1. \( A_{max} = 1 \) iff \( t_j \) is represented by a place input place.
2. \( B_{max} = 1 \) iff \( t_j \) is also represented by a place that represents \( t_j \).

When \( t_j \) is fired, it is easy to see that
\( q^0 : \emptyset \otimes A^j \otimes B^j \).

To simplify calculations and reduce memory occupation we will use only matrix \( D \), where \( D^j = A^j \otimes B^j \) and then \( \emptyset : \emptyset \otimes D^j \). Table 2 shows the new simulation algorithm.

Performances between basic and faster schemes will be shown later. The faster schema has 50% more memory occupation (matrix \( D \) and vectors \( \emptyset \) and \( \emptyset_{aux} \)) and its execution time is \( \approx 60\% \).

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

LIST-BASED REPRESENTATIONS OF PETRI NETS

The memory occupation of matrix representations is mainly proportional to \( mn \). With list-based representations, memory occupation will usually be linear in \( m \) and \( n \).

List-Based Basic Schema. The PN structure is represented by a transition list. Each element list describes: (1) 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-based schemas will be considered below.

P-T List and Marking Driven Simulation schemes. 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

![Figure 4a](image)

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

---

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

1. Data input:
2. \( \text{FOR } j = 1 \text{ TO } m \text{ DO} \)
   - \( \text{IF } q_j = 1 \text{ (test on a boolean variable)} \)
     - \( \text{THEN } \text{IF } e_j = 1 \)
       - \( \text{THEN} \quad \text{generate } e_j \)-actions; \( M_{aux} := M_{aux} \otimes e_j \);
       - \( \text{THEN} \quad \phi := \phi_{aux} \);
3. \( M := M_{aux} ; \phi := \phi_{aux} ; \)
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 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 (Bacilin, 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 places). 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 \( (P_\emptyset \cup P_{\emptyset} = P, P_\emptyset \cup P_{\emptyset} = \emptyset) \).

This condition is usually verified, but when this is not so, for a given PN, then there
Programmable Logic Controllers and Petri Nets

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 role 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 Schemes. This last scheme 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 scheme, 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, if \( C^i \) is enabled iff \( C^i = 0 \). 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 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 outputs 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.

\[ t_k \]
\[ \begin{array}{l}
\text{event associated to } t_k \\
\text{input-places} \\
\text{count. to inc.} \\
\text{count. to be incremented when } t_k \text{ fires} \\
\text{count. to dec.} \\
\text{count. to be decremented when } t_k \text{ fires} \\
\end{array} \]

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. "\( t_k \)"-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 PLCS have been programmed on a MS801.

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, \( n_{i_k} \); (3) and of input transitions of a place, \( m_{i_k} \); and (4) for a given TC, the number (a maximum estimation) of marked places, \( m_{m_k} \), and of fired transitions, \( m_k \).

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 (\( n \leq 16, m \leq 16 \)) [memory occupation and execution time grows mainly as \( K^2 n + m + K^2 n + K^2 m + K^2 \)]

When comparing only list-based methods it is clear that stack-driven simulation schemes 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 (M6800) and memory occupation estimation

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Executable Code (Hardware Scanner)</th>
<th>Matrix</th>
<th>P-T List &amp; Stacks</th>
<th>T List &amp; Stacks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Basic Faster Scanning</td>
<td>Scanning</td>
<td>Scanning Stacks</td>
<td>Stacks</td>
</tr>
<tr>
<td>1 STATE GRAPH</td>
<td>n=23, m=32, n1=1.4, m1=3, m2=2</td>
<td>1749</td>
<td>1732</td>
<td>587 (104)</td>
</tr>
<tr>
<td>2 MARKED GRAPH</td>
<td>n=32, m=23, n1=1.4 m1=1, m2=4, m4=1.5</td>
<td>1576</td>
<td>1035</td>
<td>715 (128)</td>
</tr>
<tr>
<td>3 PN</td>
<td>n=32, m=1.4, n1=1, m1=1.4 m1=1, m2=4, m4=2</td>
<td>2147</td>
<td>1211</td>
<td>812 (130)</td>
</tr>
<tr>
<td>4 PN</td>
<td>n=100, m=1.4, n1=1, m1=1.4 m1=12, m5=6</td>
<td>17295</td>
<td>8660</td>
<td>2585 (362)</td>
</tr>
<tr>
<td>Comparative performances over a (1), (2) and (3) mix</td>
<td>9.45</td>
<td>5.9</td>
<td>3.7</td>
<td>2.6 (0.64)</td>
</tr>
<tr>
<td>Memory occupation (bytes) per transitions in list-repres.</td>
<td>—</td>
<td>—</td>
<td>11.5</td>
<td>10.0</td>
</tr>
</tbody>
</table>

For list-based schemas, memory occupation is "mainly" linear with m (see Table 3, last row). At this point, it may be stated, for example, that the "P-T list & Stacks" execution code version is twice as fast as 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 realizing non-safe PN, PN based "self-testing", conditional actions and translation from specific language.

Transition-based presented schemas cannot 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 performance 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 M6800, LSI-11 or similar microprocessors.

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

REFERENCES


