A PETRI NET BASED PROGRAMMABLE LOGIC CONTROLLER

R. Valette, M. Courvoisier, JM. Bigou, J. Albukerque

Laboratoire d'Automatique et d'Analyse
des Systèmes du C.N.R.S.
7, Avenue du Colonel Roche
31400 Toulouse
FRANCE

The complexity of modern industrial installations requires the utilization of interconnected programmable logic controllers. A tool for formal specification and validation of the control procedures is necessary, altogether with methods allowing a direct implementation of these specifications in order to avoid the introduction of bugs at this step. The aim of this paper is to show that Petri net is useful in this context. A prototype of a logic controller directly programmable by a Petri net based language is presented.

1. INTRODUCTION

Logical controller complexity is increasing more and more with the high degree of automatization of modern industrial installations. The aim is now the obtention of flexible manufacturing plants and this implies the existence of a lot of interconnected sophisticated logic controllers. Specification of such systems implies the utilization of tools sufficiently powerful in order to cope with this complexity.

Firstly at the specification step, there is a need for a formal unambiguous description. It must be underlined that a validation of the specifications is necessary when the interconnection between the controllers is not straightforward. Secondly a direct implementation of the specifications is required in order to avoid the introduction of bugs at this step. Flexibility and versatility constraints lead now to software solutions and the utilization of standard microprocessors.

Section 2 presents the problem of the specification of boolean control systems and introduces Petri nets. An illustrative example is depicted. Section 3 describes the software and the hardware of the programmable logic controller prototype. A discussion concludes the paper.

2. SPECIFICATION AND VALIDATION OF BOOLEAN CONTROL SYSTEMS

2.1. Introduction and basic concepts

2.1.1. Control part and data part

A controlled system is made up of two parts: the production process
and the controller. Interaction between these two parts results from the existence of sensors and actuators. A correct specification of a controller consists of depicting exactly its responses to each state modification of sensors values. As any digital system, a controller can be decomposed into a control part describing the operation sequences and a data part (see figure 1). For boolean control systems, the control part is essential and a good specification tool must stress this part. Consequently, tools derived from state machines and automata are useful.

Furthermore evolutions of sensor values are naturally concurrent and this produces a combinatorial explosion of the number of states. A solution is a decomposition of the control part into various cooperating sub-systems. But this cooperation has to be clearly specified and validated.

Consequently a good specification tool must be such that validation procedures are applicable to the model of the system obtained by means of it. Among different specification tools, Petri net is the only one which allows an easy specification of cooperating sub-systems and the utilisation of validation procedures at the same time 'VAL 80'.

2.1.2. Petri nets

A Petri net 'PET 81' is made up of places, transitions and arcs. Places are represented by circles and transitions by bars. Arcs connect places and transitions, weights can be associated with them. Generally places are used to express "conditions" i.e. partial states of the system. Transitions correspond to control evolutions from partial states to partial states.

A place can contain tokens. A marking of a net is a distribution of tokens into the places of the net. It represents a state of the system. A transition is said to be "enabled" if the token load of each of its input places is at least equal to the weight of the connecting arc. An "enabled" transition can be fired. Transition firing consists of two indivisible operations:
- a number of tokens equal to the weight of the connecting arc is removed from each input place of the transition,
- a number of tokens equal to the weight of the connecting arc is put into each output place.

The Petri net is used to represent the possible sequences of the control part of the controller. In order to depict how the environment interacts with the controller, an interpretation is associated with the Petri net. With each transition a boolean condition involving input signals (sensors) is associated. Then a transition is fired if and only if it is enabled and if its corresponding boolean condition is true (event).

Actions are represented by means of labels attached to transitions or places. Transient actions (pulses, set or reset operations for example) are associated with transitions whereas permanent ones (level signals for example) are associated with places. When a transition is fired, the attached actions are executed. Actions associated with places are executed during all the time the corresponding places have at least one token.

Computations that have to be done and timers devices are described in the data part. It is sometimes useful to depict graphically the
data part 'VAL 78'. Sensors, actuators and internal registers are
drawn as rectangles, operators corresponding to the control part
operations as circles and predicates corresponding to the boolean
conditions attached to the transitions as diamonds. The arcs connec-
ting these nodes express the fact that some registers or sensors are
input values for some operators or predicates and that some regis-
tors or actuators are output values.

2.2. Example

The following example is a simplified version from one extracted
from 'TEL 74'. Figure 2 depicts roughly a concrete production
station. A1 and A2 are aggregate stocking hoppers, B1 and B2
stocking hoppers for concretes of various qualities.

When the operator initiate an operation cycle, the number N of
mixings is memorized if the initial conditions are satisfied. Then
the following operations are executed:
- weighing hopper A is loaded by aggregates, firstly aggregate A1 is
  measured out and gate VA remains open until the weighing machine
  indicator reaches position a1; then aggregate A2 is measured out;
- the operator selects the desired concrete quantity (c1 or c2) and
  then concrete is measured out on the second weighing hopper C,
  concurrently with the aggregate measuring out, until indicator
  reaches correct position (c1l or c1h);
- the mixer having been started up at the beginning of the cycle it
  is now filled up in the following way:
  
  .hopper A is emptied through gate VA and concurrently conveyor
  belt TPA is started up, it remains functioning 10 seconds
  after the weighing machine indicator reaches zero (za),
  
  .hopper C is emptied through gate VC, 5 seconds after the
  opening of VA; concurrently warm concrete conveyor is started up and
  remains in operation 3 seconds after weighing machine indicator has
  reached position zc; when the weighing machines reach zero the
  corresponding gates are closed.
- when these two operations are terminated, mixing is done during 60
  seconds; however inflow of water begins 10 seconds after only and
  lasts until the end of this operation;
- then emptying of the mixer (VTD) lasts 10 seconds.
These operations are repeated N times, then the system stops.

Figure 3.a represents the description of the control part of the
desired controller by means of an interpreted Petri net. A data part
is necessary because the system involves timing operations and the
use of a counter. The data part corresponding to the specifications
is represented by figure 3.b. Such a description points out the
devices that will be necessary for the implementation i.e. a certain
number of timers, a counter and a comparator. This information is
useful even when the controller directly allows a software or a
hardware implementation of such devices.

2.3. Analysis of a Petri net specification

The advantages of a Petri net specification result from the fact
that a detailed analysis is possible, allowing to detect a large
number of design errors before any implementation.

Firstly a Petri net representing the control part of a controller
have necessarily some "good" properties (bounded or safe, live
eetc...). Some tools allowing a computer aided analysis of uninter-
preted Petri nets exist, for example Ogive/Ovide 'PRA 79' which is
now maintained and developed by the firm SYSECA. An error detected at this level means that the structure of the control part is not consistent. For example deadlocks are possible or some control sequences exist that can never be executed. It must be pointed out that this analysis does not imply the enumeration of all the reachable states (reachable markings) when the structure of the net is not too much complex. It is the case of the Petri net of Figure 4.a which can be proved safe and live by a mere automatical reduction procedure.

This first analysis is complemented by another one based on the notion of linear invariant 'LAU 74, BER 79'. Linear relations involving the marking of set of places can be obtained automatically. These relations allow to detect inconsistencies at the level of the interpreted net. For example from the net of the figure 3.a the following property can be derived: the control structure implies that gate VA will never be open concurrently with gate VA1 or VA2. As a matter of fact the following invariant:

\[ M(p4)+M(p6)+M(p11) \leq 1 \]

is such that it is not possible that more than one of the three places p4, p6 and p11 contains a token at any time.

Linear invariants are derived automatically by the computer without enumeration of all the states and induction is only necessary to prove that some required specification constraints are implied by them. This way a large number of constraints can be proved to be verified, including the absence of contradictory commands or of dangerous control sequences. When the proof is not possible it can be deduced that some design error is present or that the correctness relies on some temporal relations (and not purely logical ones). When flexibility and reliability are required this second case is generally considered as a design error also.

These two analysis procedures allow to detect the major part of the design errors and enable to reduce the simulation step. As a matter of fact only temporal relations remain to be verified.

3. THE PROGRAMMABLE CONTROLLER

The principles of the programmable controller under elaboration are presented first. Then the programmation console and the controller are described.

3.1. Principles

A programmable controller is generally made up of two parts. One is concerned by the programmation i.e. the description of the required operation. The other one is concerned by the on-line control of the process. Frequently these two parts are split on two different devices: the programmation console and the controller itself.

A traditionnel programmation console generally translates a program written in a relatively high level language into code that can be executed by the controller. Our approach is a little more sophisticated because it is based on the utilization of a formal model. Consequently the programmation console of the prototype which is being built includes an editor, a syntactic and semantic analyser, a Petri net analyser and a simulation tool.

When the designer is satisfied with his specification, the control-
ler is loaded by tables describing the interpreted net. The controller then emulates this net taking into account the real values of the inputs (sensors) and actually controlling the actuators (figure 5).

3.2. The programmation console

The programmation console is a mere personal 8-bit microcomputer with the operating system CP/M. Required software has been exclusively developed in PASCAL in order to remain relatively machine independent.

The editor is any one able to produce a standard CP/M file on a floppy disk. Consequently edition and syntactic analysis are split into two different programs, each one capable of been run on the 8-bit microcomputer.

The language used to described the required operation of the controller is structured and non procedural. A program is then a mere sequence of declarations depicting the interpreted Petri net corresponding to the control structure and giving details of the operations and timers utilized in the data part.

Each identifier involved in the specification has an associated type. The following types and sub-types are defined:
- integer and boolean with sub-types input, output, timer (only for boolean), internal,
- place,
- transition,
- predicate (or boolean expression).

The program control is structured by means of composed instructions. The four possible composed instructions are the following ones:
- NODES that allows the declaration of place and transition identifiers,
- ARCS that allows the description of the graph corresponding to the Petri net by means of three instructions corresponding to the declarations of input or output place list for transitions and the description of graph paths,
- VAR allowing variables declarations including input and output addresses corresponding to sensors and actuators and timer values,
- INTERPRETATION specifying the boolean conditions attached to the transitions and the operations that have to be executed.

Table 1 presents the program describing the interpreted Petri net of figure 3.a.

We are now elaborating a new version of this language that enable the specification of a collection of communicating interpreted Petri nets. Communication will be expressed by the use of common places and common transitions. The description of various nets of the same structure by one declaration only, will be also possible.

The syntactical and semantical analyser checks the syntax of a given program, verifies that the identifiers are used correctly (type verification) and translates the interpreted Petri net into tables. These tables are exactly the ones the controller needs for the emulation of the net. At the present point nets of 100 places, 100 transitions and 300 arcs can be translated. The interpretation is limited to 100 input/output variables.
Up to now the analysis of the Petri net ("good" properties and invariants) is done off-line on a large computer. All the analysis procedures are not implementable on a 8-bit computer. Nevertheless the fact that frequently the structure of Petri nets specifying control are relatively simple encouraged us to design a mini-analyser. It will certainly include marking enumeration, reduction (with simple rules) and circuit search (a circuit is not necessarily an invariant but any linear invariant corresponds to a circuit).

At the present point a simulator has been realized 'VAL 82' in cooperation with firm Renault. This simulator includes the simulation of timed Petri nets but its input language is identical to that of the programming console. This simulator has been written in PASCAL and implemented on the programming console. It must be pointed out that this simulator is based on the same software as that of the controller emulator. The main difference is that the interpreted Petri net token player is driven by an event scheduler and not directly by input ports and a real-time clock. Therefore the simulation is very accurate and reliable because it is almost the realization itself.

3.3. Controller

3.3.1. Outlines of the controller

i) operation cycle

A programmable controller operation can be synchronous or asynchronous. In the first case a periodic pulse of the real-time clock defines a fixed cycle called operation cycle. The execution of the programs contained in the controller have to be punctuated by this cycle. In the case of a Petri net based programmable controller after each real-time pulse the input values are modified and the interpreted Petri net have to evaluate until a stable marking i.e. a marking for which no transition is fireable (enabled with the attached condition true). The program controlling the evolutions of the interpreted net is called the token player. The execution of the token player until a stable marking defines the execution cycle that has to be included in the operation cycle in absence of malfunction. Consequently at each cycle there is an idle time (figure 4.a).

In the case of asynchronous operation there is no real-time clock and operation cycles are merged into one. No idle time appears and the response time is better but the behaviour of the controller is not regular in relation to the controlled process (figure 4.b). Sometimes this is considered as a drawback.

We have chosen the synchronous operation because of its regularity. The idle times appearing at each cycle can be utilized to execute test sequences and so to improve the security of the controller. As a matter of fact security is an important requirement when industrial process control is concerned.

ii) emulation principles

The execution of the token player can be decomposed into the three following steps:

Step 1 : Establish a list of the transitions enabled by the current marking, go to step 2.
Step 2 : Consider the next transition of the list, if the list is
ended (a stable state has been reached) execute the actions attached to the places, wait for the next real-time clock pulse and then go to step 1, else go to step 3.

Step 3: Check the condition associated with the transition, if it is false go to step 2, else fire the fireable transition (evolution of the current marking), execute the actions associated with it (evolution of the data part of the controller) and go to step 1.

3.3.2. Software structure

i) Data structure

The Petri net structure is stored by means of lists in order to save memory space. In order to obtain the list of the transitions enabled by the current marking without scanning all the transitions of the net (a very slow procedure for large systems) a list of output transitions is attached to each place. In this way a list of transitions that have at least one input place containing a token can be directly obtained from the list describing the marking (list of the marked places with their token loads). In order to complement the description of the Petri net graph a list of input places and a list of output places are associated with each transition. An arc of the graph can appear in more than one list. However it has been shown 'SAM 80' that an optimization algorithm can be developed in order to avoid any redundancy in the graph storage.

The interpretation of the Petri net is made up of the boolean conditions attached to the transitions and of the actions to be executed. Boolean conditions are transformed into trees and actions are stored in tables.

It is the programmable console that translates the high-level description of the interpreted net into the adequate lists, trees and tables and provides them to the controller.

ii) Execution

After each real-time clock pulse an operation cycle is initiated. This cycle consists in emulating the interpreted Petri net until a stable state is reached, taking into account the new input values. This emulation has to be terminated before the occurrence of the next real-time clock pulse. It is therefore necessary to avoid the evaluation of every firing condition and of every boolean expression. The data structure we have chosen allow to consider only the transitions that have at least one input place containing a token. It is only for these transitions that the firing conditions are evaluated (step 1 of the emulation). It must be pointed out that neither the places nor the transitions have to be enumerated during this step i.e. the execution time of this step depends only of the number of places that are marked concurrently and does not depend of the size of the interpreted Petri net specifying the control algorithm.

During step 3 only boolean conditions attached to enabled transitions (i.e. transitions likely to be fired) are evaluated. Consequently the execution time of the emulation algorithm does not depend of the size of the net at all. In fact it depends of the degree of concurrency and of the complexity of the boolean conditions. Moreover the boolean conditions associated with the transitions are evaluated optimally: each variable is evaluated only once
and all the variables are only evaluated in the worst case.

3.3.3. Hardware of the controller

The programmable logic controller prototype has the following hardware structure (figure 5):
- A processor card based upon a Zilog 80A microprocessor with a 4Mhz clock. The emulator program is stored in a 32K ROM on this card. The real-time clock is programmed by means of an INTEL 8253/5 counter/timer.
- A 65K dynamical RAM card where the data utilized by the emulator are stored.
- A serial RS232 interface card is necessary to load the interpreted Petri net description from the programmination console. This interface is also used for the interaction with the human operator, if desired (current marking and input/output variable display, for example).
- A card allowing the acquisition of 64 TTL inputs from the industrial installation (an interface is required between this card and the actual sensors). These TTL inputs are updated at each real-time clock pulse.
- A card allowing 64 TTL outputs. An interface is necessary between this card and the actuators.

4. DISCUSSION

One may wonder whether the complexity of the emulation algorithm is not too great for simple systems. The evaluation of the prototype has shown that a simpler procedure is only better for nets that have less than 15 places. At the present time, the execution cycle of the prototype is of 50ms. This could be reduced by coding directly the emulation algorithm in assembly language. With a 64K micro-computer a Petri net of 100 transitions and 100 places can be emulated with an interpretation involving 100 input/output variables.

An other point that have to be discussed is the choice of the level of abstraction with which the model is utilized. As a matter of fact a given control mechanism can be depicted by a binary Petri net, or by a Petri net with weights attached to the arcs (see section 2.1.) or a coloured Petri net 'PET 81'. With a more abstract model the description will be more concise, less memory will be required by the data but the emulation algorithm will be more complex and slower. Binary Petri nets (i.e. all the weights are equal to 1) are frequently sufficient. However in some cases of interconnected systems as, for example transportation systems in flexible manufacturing workshops, coloured Petri nets are useful because it is necessary to identify the components being transported. It must be pointed out that the hardware of the controller is not involved and that the level of abstraction adaptation will simply require software changes.

REFERENCES


TAF 79 M. E. Tafazzoli: "Réalisation d'un interpréteur matériel de réseau de Petri, application à la commande d'un système multiprocesseur", Thèse de Docteur de 3e cycle, Université de Nice, 1979.

TEL 74 "Etude des automatismes séquentiels", Notice Technique Télémécanique 60010 F 74.

TOU 76 L. Tourres: "Une méthode nouvelle d'étude des systèmes logiques et son application à la réalisation d'automatismes programmés", Revue Générale d'Electricité, mars 1976.


VAL 82 R. Valette: "Écriture d'un simulateur à événements discrets basé sur l'utilisation des réseaux de Petri" Rapport du contrat 03.03.021/LAAS 11 avec le GIE RENAULT Recherche Innovation, 1982.

---

**Figure 1**: Functional decomposition of a controller

---

**Diagram**

- **Human Operator**
- **Control Park**
  - **Actions**
  - **Events**
  - **Controller**
- **Data Park**
  - **Actuators**
- **Production Systems**
  - **Sensors**
FIGURE 2: An Example: A concrete production station
a. Control part

b. Data part

FIGURE 3: Petri net of the example
**FIGURE 4**: Operation cycles

A. SYNCHRONOUS OPERATION

B. ASYNCHRONOUS OPERATION
NODOS
p1:PLACE 1;
p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15:PLACE;
t1,t2,t3,t4,t5,t6,t7,t8,t9,t10:TRANSITION;
t11,t12,t13,t14,t15,t16,t17,t18:TRANSITION;
t19,t20,t21:TRANSITION;

ARCOS
PATH=(p1,t1,p2,t2,p3,t3); PATH=(t3,p4,t4,p5,t7,p9,t10); PATH=(t3,p5,t5,p7,t8,p10,t10); PATH=(p5,t6,p8,t9,p10); PATH=(t10,p11,t11,p13,t13,p16,t16,p18,t17,p19,t18,p20,t19,p21); PATH=(t10,p12,t12,p14,t14,p15,t15,p17,t16); PATH=(p21,t20,p3); PATH=(p21,t21,p1),(t1,p22,t21);

VAR
N,n:INTEGER;
FT3,FT5,FT10,FT60:BOOLEAN;
DCY,VA1,VC1,VC2,VA1,VC1,TPO,TPO,MPX,MTX,MT,MT:BOOLEAN;
a1,a2,cl,fc1,cl,fc2,za,sc:BOOLEAN;
ENTRY a1='11',a2='12',cl='13',fc1='14',c2='15',fc2='16',
za='17',sc='18',DCY='21';
OUTPUT VA1='31',VA2='32',VC1='33',VC2='34',VA1='35',VA1='36',
TPO='37',TPO='38',MT='41',MT='42',MTX='43';
TIMER FT3='03',FT5='05',FT10='10',FT60='60';

INTERPRETATION
ACTION t2 = BEGIN
READ(N);
N:=0;
END;
ACTION t3 = n:=n+1;
ACTION t10 = START TIMER FT5;
ACTION t11 = START TIMER FT10;
ACTION t14 = START TIMER FT3;
ACTION t16 = BEGIN
START TIMER FT10;
START TIMER FT60;
END;
ACTION t18 = START TIMER FT10;
ACTION p4 = VA1;
ACTION p7 = VC1;
ACTION p11 = VA2,TPO;
ACTION p15 = VC1;
ACTION p20 = VID;
CONDITION t1 = DCY;
CONDITION t5 = cl;
CONDITION t7 = a2;
CONDITION t9 = fc2;
CONDITION t12 = FT5;
CONDITION t14 = zc;
CONDITION t17 = FT1O;
CONDITION t19 = FT1O;
CONDITION t21 = (n=N);

TABLE 1 : Program corresponding to the specifications of figure 3.a
FIGURE 5: Hardware structure