24th COBEM - 2017

24th ABCM International Congress of Mechanical Engineering
December 3-8, 2017, Curitiba, PR, Brazil

COBEM-2017-0418
IMPLEMENTATION OF SIGNAL INTERPRETED PETRI NETS USING C LANGUAGE IN ARDUINO

Matheus Ungaretti Borges
Eduardo José Lima II
Universidade Federal de Minas Gerais (UFMG), Av. Antônio Carlos, 6.627 - Campus Pampulha 31270-901 Belo Horizonte – MG
matheusungaretti@gmail.com,
eduardo@demec.ufmg.br

Abstract. *Nowadays, automation is an important study area that seems every day more useful to control industries processes involving machinery and robots and even home processes to control illumination and sound systems, for example. With the advent of electronics in the last decades, its costs have been reduced and, as a consequence, its usage has grown and so its efficiency, productivity and safety. For that, there is the development, software programming, and implementation on the system for solving a task. In this study the focus will be the definition of Signal Interpreted Petri Nets (SIPN) and how it may be used to model this task. It will be proposed a methodology to convert the SIPN into C language to be implemented in a microcontroller.*

Keywords: *Automation, Signal Interpreted Petri Nets, Arduino, Microcontrollers.*

1. INTRODUCTION

Petri net (PN) is a useful tool designed for modelling discrete event systems (DES). Its theory was developed in 1962 by C. A. Petri, according to Zhou (1998). This use is an organized way to describe a range of events. Murata (1989) defines a PN as an 5-uple represented by PN=(P, T, F, W, M_0), where:

$P=\{p_1, p_2, \ldots, p_n\}$ is a finite set of places
$T=\{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions
$F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs
$W=F \rightarrow \{1, 2, 3, \ldots\}$ is a weight-function
$M_0=P \rightarrow \{0, 1, 2, 3, \ldots\}$ is the initial marking, $P \cap T=\emptyset$ and $P \cup T \neq \emptyset$

In this model, Murata (1989) used the concept of conditions and events, which are related to places and transitions, respectively. A transition is a connection of certain number of input (pre-conditions) and output (post conditions) places. The initial marking represents the distribution of tokens on the PN. A token in a place means the conditions associated with that place are true. It can also indicate a quantity of available resources, and in this case one place can have k tokens.

Frey (2000) went further and defined a SIPN, which is a specific type of PN that contains more information, as a 9-uple SIPN=(P, T, F, M_0, I, O, φ, ω, Ω), where the first four elements were already described in an ordinary PN. Other elements can be described as follows.

I a set of logical input signals with $|I|>0$
O a set of logical output signals with $I \cap O=\emptyset$, $|O|>0$
φ a mapping associating every transition $t_i \in T$ with a firing condition $\varphi(t_i)$ = Boolean function in I
ω a mapping associating every place $p_i \in P$ with an output $\omega(p_i) \in (0, 1, -)$, where (-) means 'don't care'
Ω the output function combines the output ω of all marked places Ω: $m \rightarrow (-, 1, 0, c, r_0, r_1, c_0, c_1, c_{01})$. The combined output can be undefined (-), one (1), zero (0), contradictory (c), redundant zero or one (r_0, r_1), or a combination of contradiction and redundancy (c_0, c_1 c_{01}).

Transitions could be interpreted as the flow of tokens through the net. This means that when a transition is fired, a token is removed from each pre-place and put on each post-place that are connected to it. However, there are four rules that must be satisfied to activate its firing process.

1. *A transition is enabled, if all its pre-places are marked and all its post-places are unmarked.*
2. *A transition fires immediately, if it is enabled and its firing condition is fulfilled.*
3. *All transitions that can fire and are not in conflict with other transitions fire simultaneously.*

4. The firing process is iterated until a stable marking is reached (i.e. until no transition can fire anymore). Iterated firing is interpreted as simultaneous. This also means that a change of input signal values cannot occur during the firing process.

After a new stable marking is reached, the output signals are recalculated by applying Ω to the marking. (Frey, 2000)

After all these definitions, Frey (2000) developed a methodology to convert a SIPN to ladder language. This methodology defines input and output signals of the PLC for each transition and event. Each place p_i is assigned a Boolean function value: TRUE for marked and FALSE for unmarked. In Frey (2000), for a general non-safe SIPN it is obligatory to test post-places within firing conditions, whereas, considering a safe SIPN, which each place could have one token at maximum, it is unnecessary to test post-places in the transitions simplifying the program. Thus, satisfying all conditions the transition fires, unmarking all pre-places and marking all post-places. Frey (2000) considers this constraint as the number of tokens at each place is represented as a contact (true or false). In our work, this number can be represented as an integer variable in C language, then this constrain will not be used.

Recent work (de Mello, *et al.*, 2012) a transcription tool from Petri nets to PLC programming languages (IEC 61131-3) was developed. There are five languages: Sequential Function Chart (SFC), Ladder Diagram (LD), Function Block Diagram (FDB), Instruction List (IL) and Structured Text (ST). Despite of the existence of a structured language within IEC 61131-3 and its similarities with Arduino's C programming language, there is(are) essential difference(s) between them. Just to mention one: scan cycle routine is absent in Arduino's C language.

Recently, Moreira *et al.* (2009) and Manhães and Moreira (2014) shown a conversion from SIPN to Ladder Diagram dividing it in small command blocks: initialization, to represent system's initial marking; transition, to symbolize firing conditions of each transition; action, to represent which actions will be executed in each place. In our methodology, a similar structure has been used because it is an organized way to describe a SIPN. Furthermore, scan cycle routine was implemented.

2. METHODOLOGY

The methodology of this paper will consist on developing a methodology to convert a SIPN model to a C program to be implemented in a microcontroller detailing hardware and software. As a case study, a pneumatic system with two cylinders (Fig. 1) will be used to demonstrate the applicability of this methodology inspired in Frey's work in industrial systems. There are seven input signals: four end-course sensors, a start button and two inductive sensors (one for each cylinder). And there are four output signals: cylinder A advance and return and cylinder B advance and return. It will be used to separate metallic blocks from other materials by using inductive sensors.

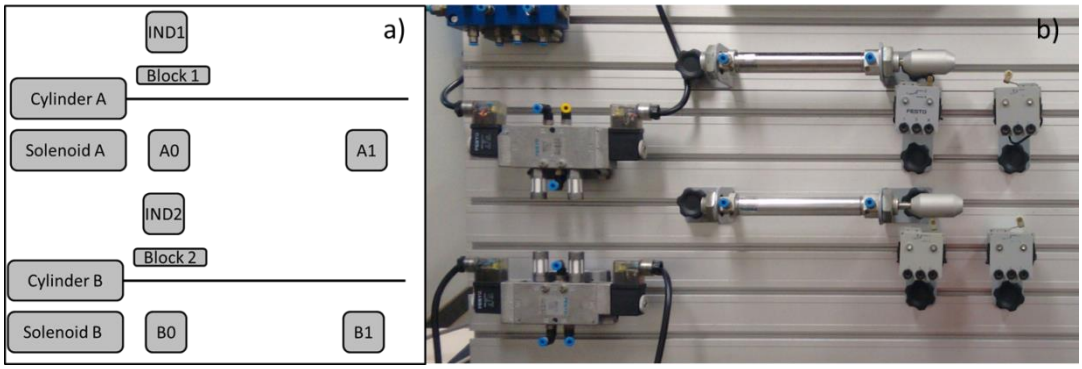

Figure 1 – a) System's scheme; b) System's devices

When the blocks are put in front of the cylinders, inductive sensors will respond a Boolean sign (true or false) to the system, informing if both blocks are metallic or not. If the first condition was true and the start button was pressed, both cylinders will advance simultaneously to collect the blocks together and return one at a time, whereas, in any other situation, they will advance separately and return simultaneously. In the end of this process, it will only restart when new blocks were put in their initial positions for new analysis and the firing conditions were satisfied again.

Arduino was chosen to implement SIPN instead of PLC, because it is a low-cost user-friendly microcontroller, which makes it accessible for university alumns and laboratories with small investment. Furthermore, other important hardware components used were switches to control when each solenoid's side will receive the electrical signal to command pneumatic system outputs (cylinders A and B advance or return). Figure 2 shows these parts connected.

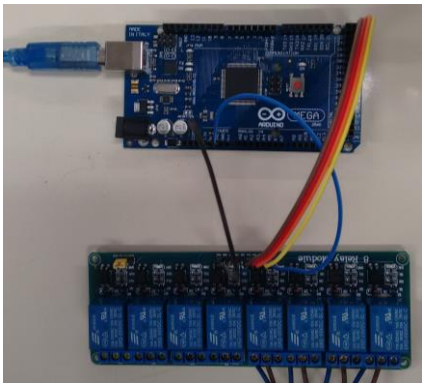

Figure 2 – Arduino and switches connection

Using the 9-uple vector SIPN definition of introduction section, the system (Fig. 3) will be develop as follows.

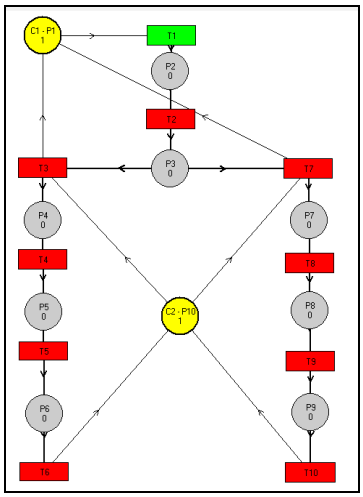

Figure 3 – SIPN model – software by Lima II (2002)

3. RESULTS AND DISCUSSION

In this section, the system will be presented in details, showing the main aspects and results of the conversion from SIPN to C language for Arduino.

Arduino's software for programming the script in C language basically consists of two functions: setup(), which is responsible for initializing the program every time Arduino's started or restarted; and loop(), which runs continuously after setup() is executed once. However, in this study case, two more functions will be necessary: ReadInputs() and WriteOutputs(). When input signals are read, they are copied to a mirror of inputs, which is used during the program's execution. So, the function WriteOutputs() record data on a mirror of outputs and just at the end of program's execution this data is transferred to an output vector to be executed. In the end of this cycle, it starts again from loop(). This is called scan cycle. Its implementation was necessary in Arduino's program because, differently from PLC, this structure was not ready in C language.

The program begins with command lines (Fig. 4) to declare as global variables some important features of the system such as number of inputs and number of outputs. Each input and output should be addressed to a pin of microcontroller. And then, they should be defined as input or output within setup() function (Fig. 5), whereas in PLC inputs and outputs are already defined. Another essential characteristic is initial marking, which is specific for each SIPN. To store initial marking values, a vector of integers was created meaning that each place could receive a positive integer number of tokens.

```
const int numinputs=8;
const int numoutputs=8;
int input[numinputs]; //I0.0, I0.1, ..., I0.7
int output[numoutputs]; //Q0.0, Q0.1, ..., Q0.7
char inputpin[]={23,25,27,29,31,33,35,37}; // set of logical inputs I
char outputpin[]={22,24,26,28,30,32,34,36}; // set of logical outputs O
int P[10+1]={0,1,0,0,0,0,0,0,0,0,1}; // Initial marking M0
```

Figure 4 – Command lines to declare variables

```
void setup() {
  int i;
  for (i=0; i<numinputs; i++){
    pinMode(inputpin[i],INPUT_PULLUP);
  }
  for (i=0; i<numoutputs; i++){
    pinMode(outputpin[i],OUTPUT);
    output[i]=0;
  }
  WriteOutputs();

}
```

Figure 5 – setup() function

For model's easily understanding, transitions and places of SIPN will be represented by small blocks of commands (Fig. 6 and Fig. 7) within loop() function. Each one of them will present a conditional structure *if*, with their respective criteria. On transitions case, for example, it will be verified if determined place is marked and if its activation conditions are being satisfied. If it is true, this transition will be activated unmarking its pre-place and marking its post-place. On places case, the condition of a marked place will activate which outputs that will be written, and, afterwards, which commands will be executed.

```
//Transition t10
if((P[9]==1) && (input[0]==1) && (input[2]==1)){
P[9]=0;
P[10]=1;
}
```

Figure 6 – Command blocks for transitions

```
//Place p4
if (P[4]==1){
output[0]=1;
output[1]=0;
output[2]=1;
output[3]=0;
}
```

Figure 7 - – Command blocks for places

As previously mentioned, ReadInputs() and WriteOutputs() functions (Fig. 8) should be called within loop() function, respectively, to begin and to finish it. So, it will happen what is called scan cycle, and which time the program pass through loop() function values will be refreshed.

```
void ReadInputs()
{
  int i;
  for (i=0; i<numinputs; i++){
    if(digitalRead(inputpin[i])==HIGH)
    {
      input[i]=0;
    }
    else
    {
      input[i]=1;
    }
  }
}

void WriteOutputs()
{
  int i;
  for (i=0; i<numoutputs; i++){
    digitalWrite(outputpin[i],output[i]);
  }
}
```

Figure 8 – ReadInputs() and WriteOutputs() functions

ReadInputs() function has other particularity, when an input pin receive HIGH (TRUE) as Boolean value it was forced to receive 0 (zero) instead because it is an ACTIVE-LOW command.

4. CONCLUSION

The focus of this paper is developing a methodology to convert SIPN to C language to be implemented in Arduino. Arduino is a low-cost user-friendly microcontroller, whose usage has been increasing in a wide range of industries and residences. Its software programming is flexible and effective, but modelling SIPN directly in C language can be tricky. Our methodology provides a straightforward graphical way of making this conversion, maintaining the formalism of SIPN's definition.

The methodology was presented through a specific example. However, we took care to explain its procedure in a generic way that renders it adaptable to any other SIPN. All SIPN information is described in the program, except for specific logic of transitions and places.

5. REFERENCES

de Mello, A.T.F., Barbosa, M.C., Filho, D.J.S., Miyagi, P.E. and Junqueira, F., 2012. "A Transcription Tool from Petri Net to CLP Programming Languages". In *ABCM Symposium Series in Mechatronicos – Vol.5 Section IV – Industrial Informatics, Discrete and Hybrid Systems, p.781-790*

Frey, G., 2000. "Automatic Implementation of Petri Net Based Control Algorithms on PLC". In *Proceedings of the American Control Conference*, Chicago, USA

Lima II, E.J., 2002. *Uma Metodologia para a Implantação através de CLPS de Controle Supervisório de Células de Manufatura Utilizando Redes de Petri*. Dissertation, Universidade Federal da Bahia, Salvador

Manhães, M.S. and Moreira, M.F., 2014. "Proposta de Utilização de Redes de Petri Interpretadas para Controle na Modelagem e Análise de Sistemas a Eventos Discretos". Graduation work, Instituto Federal Fluminense (IFF), Campos dos Goytacazes

Moreira, M.V., Botelho, D.S. and Hazan, S.S., 2009. "Implementação de Sistemas de Automação Descritos por Redes de Petri Interpretadas para Controle". In *Proceedings of the XXXVII Congresso Brasileiro de Educação em Engenharia - COBENGE 2009*. Recife, Brazil.

Murata, T., 1989. "Petri Nets: Properties, Analysis and Applications". In *Proceedings of the IEEE*, Vol. 77, no. 4, p.541-580

Zhou, M. and TWISS, E., 1998. "Design of Industrial Automated Systems Via Relay Ladder Logic Programming and Petri Nets". *IEEE transactions on systems, man, and cybernetics – part C applications and reviews*, Vol. 28, no. 1, p. 137-150

6. RESPONSIBILITY NOTICE

The author(s) is (are) the only responsible for the printed material included in this paper.