Embedding Time Petri nets

Maurice Comlan, David Delfieu, Médésu Sogbohossou, Antoine Vianou

Embedding Time Petri nets

Maurice Comlan *†, David Delfieu*, Médésu Sogbohossou† and Antoine Vianou†
*Laboratoire des Sciences du Numerique de Nantes
1, rue de la Noë, BP 92101 44321 Nantes Cedex 3
Email: comlan@irccyn.ec-nantes.fr, delfieu@irccyn.ec-nantes.fr
†Ecole Polytechnique d'Abomey-Calavi
01 BP 2009 Cotonou, Benin
Email: sogbohossou_medesu@yahoo.fr, avianou@yahoo.fr

***Abstract*—This paper presents a tool (PN2A) which embeds Time Petri Nets (TPN) to Arduino micro-controller architecture. PN2A imports TPN and generates Arduino sketches, which can be then compiled and uploaded to a micro-controller architecture. Some transitions (resp. places) of the transition set (resp. place set) can be assigned to pins of the micro-controller. Theses two types of transitions generates a new firing semantics combining weak and strong semantics. Embedded, the TPN becomes partially non-autonomous and can be defined as a microcontroller Synchronised Time Petri net (mSTPN).**

I. Introduction

Petri nets are a modeling language covering a wide domain: control systems, communication protocols, distributed systems. Several extensions even enlarge its application to Artificial Intelligence (Predicate-Transition PN), Object Programming with Coloured PN, or real-time with temporal extentions (T-TPN, P-TPN, ...). This paper focuses on Time Petri Nets (TPN), particularly adapted to model real-time concurrent systems.

Generally the modeling of a system with Petri nets is made only on the control part. It can be, already, a challenge because it implies to have a good knowledge of the system. From this model generally, the state space is computed and properties like boundedness, reachability, liveness, temporal properties can be verified. When it is necessary to test performance constraints or implementation aspects, the environment is modelized and coupled to the application in a global model. This approach necessites to have a good knowledge of the environment. In that case, the size of the global model increases and it can be more difficult to compute the state space.

The idea of this paper is to embed the modelling on a microcontroller, *challenged* to its real environment. The structure of the TPN is translated in an incident matrix and enabling and firing functions are defined. In addition, one originality of the appoach is the possibility of a "partial association": a subset of places and transitions can be associated to the pins of a microcontrollers. The environment is external and physical : actuators, sensors, motors, leds, human interactivity ... The modelling can be simulated in a real context.

The TPN is transformed into an Arduino Sketch through a code generation step (see Figure 1) for a predefined target architecture. Actually, these architectures are limited to Arduino cards. The sketch of the TPN contains a matrix

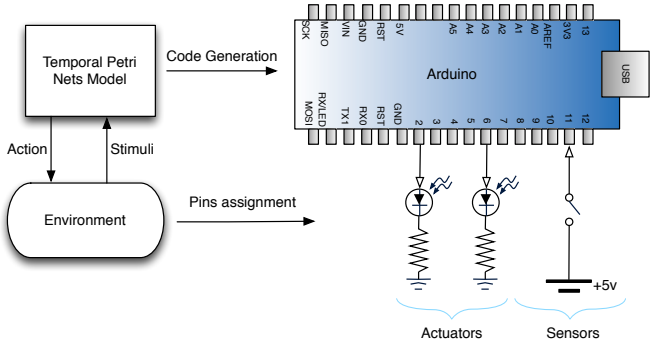

Fig. 1. Model and its environment

structure, functions computing the enabling set and the firing of transition and a timer handler that ages enabled transitions.

Embedding a TPN on a microcontroller requires to set up two definitions that constitute the major contributions of this paper:

- The definition of an embedded TPN, connected to its environment (see definition 2 of section III-C). Connecting a TPN with a real environment, defines a non-autonomous Time Petri net : *a microcontroller Synchronised Time Petri net* ($msTPn$). The definition of an assignment application specifies that subset of transitions or place can be associated to pins. As large microcontrollers can contain upto several dozens of general purpose Input/Output pins, a large range of TPN can be projected to microcontroller cards becoming cheap automatism systems.
- A token player that describes how evolve the $msTPN$ on the microcontroller. This token player is defined througth the description of the firing semantics (definition 3 of section IV).

Section II presents different code generation approaches from Petri nets and how the methodology presented in this paper can be situated. Section III introduces definitions about PN, TPN and msTPN. The section IV presents the firing semantics, the next section describes the tool while the last section illustrates the approach by an example with metrics about execution times and memory occupation.

II. Discussion and related works

Many studies discuss about code generation from Petri Nets. A classification of theses approaches can be found in [9]: Centralized approach, totally centralized and hybrid approach. Centralized approaches implement a "token player" investigating every transition to evaluate their firability. Filters have be founded to decrease the complexity of the algorithm, however thoses filters does not preseve parallelism.

Lee G., H. Zandong and J. Lee: ([7]) present a centralized approach to generate Ladder diagrams from Petri nets. They limit their work to safe and deterministic Petri nets (Control Petri Net). In [8], Ferrarini proposes to synthetize logic controller from Petri nets, but a lot of restrictions are imposed on the Petri nets : The whole Petri nets must be covered by place invariant and transition invariant and the net must be live.

In totally decentralized approach [14], each place and transition are implemented as a processes. This approach preserves parallelism but the overhead of time and memory due to the task scheduler ruins performance. Richta and R. Ko ([16]) propose distributed approach to embed Workflow Petri nets specification on a microprocessor architecture. Concurrency is obtained by processes and subprocesses. The initial model is transformed through several steps into a bytecode interpreted by a Petri nets Virtual Machine which is part of a Petri nets dedicated operating system. This approach is more sophisticated (use of virtual machine) than the approach presented in this paper however, the memory amount is very important for even small modelizations. The authors admit that the SRAM memory of the ATmega328 chip (2kB) is a strong limitation and they propose in future works to use a Raspberry.

In hybrid approaches, the idea is to partition, if possible, a net into a set of components that can be executed concurrently. Those components are "sequential state machines" that can be implemented as processes. Those components can be indentified by the computing of place invariants. If every place invariant does not correpond to a state machine, a state machine corrresponds to a place invariant. Extentions ([6]) using colored invariant can be used to select the right place invariants.

We place our works in a "token player" approach : All the transitions of the net are tested at each evolution step of the net. The token player stands for a scheduler. Moreover, this approach deals with Time Petri nets, without restrictions. This work translates a Time Petri nets into a light structure, in term of memory (see table I), as an Arduino sketch directly downloadable, without operating system, on a pre-determined microcontroller card. We aim to propose a fast, automatic and low-cost code generation procedure.

III. Basic definitions

A. Petri Net

A Petri Net (PN) [13] $\mathcal{N} = < P, T, Pre, Post >$ is a tuple with: P, a finite set of places, T, the finite set of transitions, $P \cup T$ are nodes of the net; ($P \cap T = \emptyset$), and $Pre : (P \times T) \longrightarrow \mathbb{N}$, the Pre-condition function defining incoming arcs (and their valuations) between a place and a transition of $\mathcal{N}$ and $Post : (T \times P) \longrightarrow \mathbb{N}$, the Post-condition relation defining outgoing arcs (and their valuations) between a transition and a place of $\mathcal{N}$.

The pre-set (resp. post-set) of a transition t is denoted ${}^\bullet t = \{p \in P \mid Pre(p,t) > 0\}$ (resp. $t^\bullet = \{p \in P \mid Post(t,p) > 0\}$). A marking of a Petri net $\mathcal{N}$ is a mapping $M : P \longrightarrow \mathbb{N}$. A transition $t \in T$ is *enabled* in M iff: $\forall p \in {}^\bullet t,\ M(p) \geq Pre(p,t)$. This is denoted: $M \xrightarrow{t}$. Firing of an enabled transition t leads to the new marking M' ($M \xrightarrow{t} M'$): $\forall p \in P,\ M'(p) = M(p) - Pre(p,t) + Post(t,p)$. $Enabled(M) = \{t \in T,\ \text{s.t. } M \xrightarrow{t}\}$. The initial marking is denoted M_0.

B. Time Petri nets

Numerous extensions have been proposed to take into account temporal specifications. The two majors extensions are Timed Petri nets [10] and Time Petri nets (TPN) [11]. In Timed Petri nets a numerical value is associated to a transition which represents the firing duration. An equivalent mode is proposed by [12]: p-time Petri nets where durations are associated to places.

Concerning TPNs other extensions have been introduced to enrich expressive power : Scheduling TPN, inhibitors/reset Arcs TPN, and some other ones. All these extensions use Stopwatch and are generically named *SwTPNs*. In this class [3] state reachability is undecidable, but this paper is restricted to simple TPN, where this property holds and which gives a satisfying expressive power in most of usual control applications.

Definition 1: A Time Petri net *is a tuple* $< \mathcal{N}, m_0, I_s >$ *in which* $\mathcal{N}$ *is a Petri net and* $I_s : T \to Q^+ \times Q^+ \cup \infty$ *is called the Static Interval Function.*

For every transition t is associated a static interval $[efd(t), lfd(t)]$ where $efd(t)$ is the earliest firing date of t, and $lfd(t)$ is the lastest firing date of t.

The *enabling date* of a transition is the date of the last firing date which has enabled t. At this date, the local clock associated to the transition is reset and begin to be aged. A transition t, for which time clock value belongs to its static interval is *firable*. Time elapsing is generally considered in a continuous way, and those clocks are defined on $\mathbb{R}^+$. In this paper, time elapsing is generated by a timer, thus, clock variables will be defined on $\mathbb{N}^+$.

Two semantics can be considered ([5]) : in a strong semantics, after $lfd(t)$, the transition t will be unavoidable. In a weak semantics, this transition can expire if its clock exceeds $lfd(t)$.

C. Microcontroller synchronised Time Petri net

This work proposes to associate marked places to the activation of pre-defined output pins (eventually connected to actuators, like in Figure 1) wheras transitions can be associated to positive or negative edge on predefined input pins.

Let's define the assignment of a TPN to a microcontroller. A transition can be associated, to positive or negative edge on an associated pin defined as an input. A marked place can

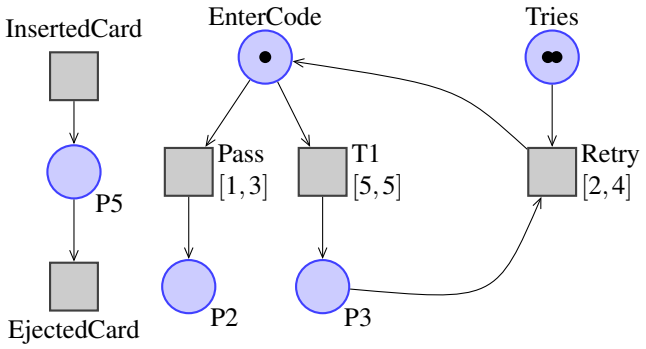

Fig. 2. Identification System

provoke a high or a low level on an associated pin that is defined as output. It is important to note that the assignation of transitions and places to the pins of the microcontroller is not complete. The user must defines the transitions and the places that will be associated to the pins of the micro-controller.

We introduce the term of *sensitized*. A transition will be sensitized if it is activated by an pre-defined external event: a positive edge or a negative edge. Associate a place to a pin means that a high or a low level will be applied on this pin when the place is marked.

Notational : Let's note T^a, a subset of T constitued by a set of transitions associated to some pins of the microcontroller, and P^a, a subset of P constitued by a set of places associated to some pins of the microcontroller. Let's note $Pins()$ the set of pins of a microcontroleur. $Pins()$ defines for each pin a tension level (High, Low).

Definition 2: A microcontroller synchronised Time Petri net (*or* msTPN) *is a tuple* $< \mathcal{N}, \mathcal{A}, P^a, T^a >$ in which $\mathcal{N}$ *is a Time Petri net*, T^a (resp. P^a) a subset ot transitions (resp. places) of T (resp. P) that are associated to a pin of the microcontroller and $\mathcal{A}$, *is the assignment application:* $\mathcal{A} : P^a \cup T^a \rightarrow < Pin, Level >$. This application makes correspond a node $n \in P^a \cup T^a$ to a pair (pin, L) where *pin* is a pin number, and L a logical level.

If $n \in P^a \cup T^a$, then $\mathcal{A}(n) \rightarrow pin$ denotes the associated pin and $\mathcal{A}(n) \rightarrow L$ denotes the associated tension level of the pin associated to the node n. $\mathcal{A}$ is an application, $\mathcal{A}(n)$ refers to only one pair (pin, L). Two definitions folllows for the previous:

- In a msTPN, a transition t becomes urgent, if this transition is not associated to a pin ($t \notin T^a$) and when its local clock reaches its last firing date. We call $Urgent$ the set of urgent transitions.
- A transition t is senzitised if, on the pin $\mathcal{A}(t) -> pin$, is observed a level correponding to $\mathcal{A}(t) -> level$.

IV. Semantics

A. Example

The example (Fig. 2) modelizes the identification system of a cash dispenser. A user has 3 tries for beeing identified (place P_2). If he fails (place P_3), two tokens in place *Tries*, give him two additional chances. Temporal intervals allows to implement a watchdog. T_1 has not been associated to a pin, if the user has be inactive until the date 5, the transition T_1 is forced. The place P_5 models the use of the cash dispenser. *Pass* and *Retry* have been associated to input pins, connected to switches. While *EnterCode*, *Tries*, *insertedCard* and *EjectedCard* are places associated to output pins connected to leds.

The firing of *Pass* is conditionned to the occurence of an event on an input pin of the microcontroller, to its local clock and to the marking of the place *enterCode*. Whereas, for the transition T_1, the firing condition is bounded to its local clock and the marking of *enterCode*.

Initially, *enterCode* contains one token, *Pass* and T_1 are enabled, and theirs clocks are aged. If a positive edge occurs before date 5, *Pass* will be fired otherwise T_1 will become urgent and will be fired.

The concept of urgency forces to consider for T_1 a strong semantics. Thus it is relevant to consider that a strong semantics is associated to *non connected* transition. By opposite, for connected transitions a weak semantics is necessary: Concerning *Retry*, this transition is enabled if there is enought token in incomming places (*EnterCode* and *Tries*), and if the expected event of the associated pin has occured and if its clock belongs to the interval $[2,4]$. If the expected event does not occur, then firing is not forced, and the tokens are implied in input places becomes dead. For this type of transition (connected transition) a weak semantics will be adopted.

B. Firing semantics

In a *msTPN*, the firing of a connected transition is conditionned to three conditions i) its enabling condition, ii) the value of its local clock and iiii) an edge signal (positive or negative) observed on its associated pin. While a non connected transition, the two first conditions are necessary.

The semantics of a microcontroller synchronised Time Petri net can be stated as a synchronised timed transition system which is defined in the following definition (Definiton IV).

Notational : Let's note $v(t)$ the function that defines the temporal valuation of a transition t. v_0 the initial value of all the clock vector associated to the transitions. $< M, v > \xrightarrow{\theta} < M, v' >$ expresses the firing of t from the marking M, with M' and v' respectively the new marking and the new clock vector.

Definition 3 (Semantics of a msTPN): The semantics of a msTPN $\mathcal{N}$ is defined by the transition system:

$$< Q, \{q_0\}, \Sigma, \rightarrow >$$

- $Q = \mathbb{N}^{|P|} \times \mathbb{N}^{|T|}$
- $q_0 =< M_0, v_0 > \in Q$
- $\Sigma = T$
- $\rightarrow \subseteq Q \times (T \cup \mathbb{N}) \times Q$

– The temporal transitions are such that:

$$< M, v > \xrightarrow{\theta} < M, v' > \textbf{ iff}$$

$$\begin{cases} \forall t_k \notin T^a,\ M \geq^{\bullet} t_k \\ \textbf{if } (v(t_k)+\theta \geq lfd(t_k)) \ \wedge (v(t_k) < lfd(t_k)) \ \wedge (t_k \notin T^a) \\ \qquad Urgent += t_k \\ v'(t_k) := v(t_k)+\theta \end{cases}$$

– If $t_i \in Urgent$:

$$< M, v > \xrightarrow{t_i} < M', v' > \ \textbf{iff}$$

$$\begin{cases} \begin{cases} M \geq^{\bullet} t_i \\ v(t_i) \leq lfd(t_i) \end{cases} \\ M' := M -^{\bullet} t_i + t_i^{\bullet} \\ \forall t_k \in T, v'(t_k) := \\ \qquad \begin{cases} 0 \textbf{ if } (t_k \in newlyEnabled(M')) \\ \textbf{else } v(t_k) \end{cases} \end{cases}$$

– else

$$< M, v > \xrightarrow{t_i} < M', v' > \ \textbf{iff}$$

$$\begin{cases} \begin{cases} M \geq^{\bullet} t_i \\ efd(t_i) \leq v(t_i) \leq lfd(t_i) \\ senzitised(t_i) \end{cases} \\ M' := M -^{\bullet} t_i + t_i^{\bullet} \\ Pins(\mathscr{A}(t_i) \rightarrow pins) = \ \mathscr{A}(t_i) \rightarrow level \\ \forall t_k \in T, v'(t_k) := \\ \qquad \begin{cases} 0 \textbf{ if } (t_k \in newlyEnabled(M')) \\ \textbf{else } v(t_k) \end{cases} \end{cases}$$

In this definition T^a (see section 2) allows to distinguish connected and not connected transitions ($t \notin T^a$). The first item defines the $Urgent$ set, it is important to note that a connected transition cannot belong to $Urgent$. The second defines the firing of urgent transition, and the third, the firing of not urgent transition. Let's remark that if $v(t)$ exceeds $lfd(t)$ ($(t \notin T^a)$ this transition is not anymore firable, until it becomes newly enabled.

C. Algorithm

In practice, the activating of the msTPN is implemented by the following algorithm:

The algorithm is a loop and the time cycle can be adjusted by a delay (line 7). The line 3 indicates that urgent transitions are fired in priority. In line 4, every urgent transitions have been fired, and all other transitions are inspected. Their enabling and "senziting" conditions are evaluated. When a transition is fired, as expressed below, the table firable and senzitised are updated and the associated clock is reset.

In parallel of this algorithm, an interruption preempts this algorithm every θ unit of time and ages clock variables of enabled transition:

$$\forall t_i \in T, \ \textbf{if } enabled(t_i) \ \textbf{then } v(t_i) += \theta$$

Every transition that reaches $lfd(t)$ is added to $Urgent$:

$$if((Horloge[t] < efd[t]) \wedge (Horloge[t]+\theta >= lfd[t]))$$
$$Urgent = Urgent \ \cup \ t;$$

while *true* **do**
1. Read the input pins of the microcontroller ;
2. Update vector *sensitized* and *enabled*;
3. **while** $Urgent \neq \emptyset$ **do**
   t:=pop($Urgent$);
   if $(enabled(t) \wedge firable(t) \wedge (pinTransition[t] = None))$ **then**
   $m := m - Pre(p,t) + Post(t,p)$;
   $firable[t] := False$;
   $sensitized[t] := False$;
   $v[t] := 0$;
   end
   end
4. **foreach** $t \in T$ **do**
   if $(enabled(t) \wedge sensitized(t) \wedge firable(t))$ **then**
   $m := m - Pre(p,t) + Post(t,p)$;
   $firable[t] := False$;
   $sensitized[t] := False$;
   $v[t] := 0$;
   end
   end
5. For all newly enabled transition t, $v[t] := 0$;
6. Digital pins corresponding to associated places are activated;
7. A delay completes the cycle;

end

V. Practical Implementation

In the example (see section 2), *Pass*, *Retry* are associated to positive edges on two predetermined pins, while *EnterCode* and *Tries* are associated to leds (see the figure 4 in appendix). Other transitions or places are not associated. T_1, is not associated, T_1 is an internal transition that illustrates a watchdog.

A. Testing

The testing of the model has enabled to asset the time intervals and the time cycle. With two tokens in place *enterCode*, when activating the transition *Pass* with a switch, a bounding phenomena has lead to empty this place with only one action on the switch. This problem has been solved by stating the efd of *Pass* to 1.

It has been also interesting to compare with the simulation mode of Romeo. In this mode, Romeo updates cloks variables when transitions are selectionned by the user, there is no a real-time clock that ages transitions. In the initial state of fig 2, if we let the time elapses, after date 5, T_1 does not become urgent ! In a simulation mode, if T_1 is activated by the user, then clocks of enabled transitions are updated (+5), whenever the date of this activation took place. In the execution of a *msTPN*, T_1 is fired exactly at date 5.

Embedding the TPN really confronts the model to its environment. Finally this implementation has allowed to fit the initial intuition of the modelisation.

B. Complexity and metrics

The greater complexity of the algorithm is the computing of $Enabled(t)$. The algorithm contains a double loop in $|P| * |T|$ and loops in $|T|$. Then the complexity is polynomial in order of $a.n^2 + b.n + c$ with $n = max(|P|, |T|)$.

For an Atmega2560 with a 16 MHz crystal oscillator, table I presents the execution times and memory amount of the embedded TPN correponding to the example. The third column give execution times of the algorithm (see section IV-C) from line 1 to 5. This last result has been obtained by visual observation of pulses on an oscilloscope (precision of 1 μs). Memory amounts have been given by the compilation step. Column 3 represents the size of the code, whereas column 4 represents the dynamic allocation of memory. The size of the code and the amount of dynamic memory increase linearly, without surprise, proportionnaly to the size of matrices. Whereas, the execution time grows more significantly: As expected, the execution time increases in a polynomial order.

For the last test, 25 places and 25 transitions, the execution time rises to 4 *ms*, it corresponds near to the maximum number of input/ouput of the atmega2560. It is important to note that every transition may not be associated to a pin of the microcontroller.

TABLE I
EXECUTION TIMES AND MEMORY SPACE

Complexity	*Time*	*Prog. Space*	*Dyn. Memory*
$\|P\| = 5$ $\|T\| = 5$	150 μs	4.9 kb	592 bytes (5%)
$\|P\| = 10$ $\|T\| = 10$	180 μs	5.3 kb	1.2 kb (7%)
$\|P\| = 15$ $\|T\| = 15$	840 μs	5.9 kb	2.1 kb (15%)
$\|P\| = 20$ $\|T\| = 20$	2.5 ms	6.7 kb	3.3 kb (40%)
$\|P\| = 25$ $\|T\| = 25$	4 *ms*	7.7 kb	4.8 kb (59%)

The tests of complexity have been realized from the initial example: the increasing sizes have been produced by putting in concurrency n duplications of the basic model.

VI. TOOL

PN2A [1] is a software where binary executables are available for Windows (64 bits), Mac OS (64 bits) and Linux environments. Input files describing the TPN can be edited by two PN editors: Tina and Romeo. The parser, the interface and the algorithm have been written in Lisp.

PN2A proposes an IDE (see the figure 3 in appendix) constitued by a main window where the user chooses a microcontroller card, and defines the assignment between places, transitions and pins. The user can adjust the delay of the cycle (cf IV-C) and the time unit. Pressing the button "Generate Arduino sketch" generates the code downloadable directly through Arduino IDE.

A. Code generation

An Arduino sketch (a project) is organized around a repertory containing all the implied files. A file with the same name as the directory (with *ino* extention) is the main file with at least, a *loop* and a *setup* functions. *setup* contains the initializations while *loop* is called infinitely and will include the algorithm IV-C. The code generation is realized from a set of static files (independent of the TPN) and a "dynamic file" which is stated from the parsing of the TPN.

[1]PN2A is freely available at http://pn2A.rts-software.org/

B. Parsing TPN files

Romeo produces xml files, while Tina produces text files. The parsing of those files produces the matrices: *Pre*, *Post*, M_0, *Arcs*, $Alpha$ and *Beta* that defines the structure of the TPN:

- *Pre* and *Post* are respectivley the pre-condition and the post-condition matrices
- M_0 is the initial marking vector
- *Arc* gives the valuations of the arcs incomming or outgoing of transitions
- $Alpha$ and *Beta* represents respectivelly the earlier firing date and the latest firing date of the transitions.

When the upper bound, lfd is the infinite, it is represented by the value $INT32\_MAX$ $(2^{31} - 1)$.

C. Static files and generated files

The main file of the arduino sketch includes many static files (independant of the models) and generated files by the parsing:

- Generated file containing structure declaration (described in previous section VI-B)
- A file contains the algorithm IV-C.
- A static file contains different functions : *enabled*, *fire*, *newlyEnabled*, *findUrgent*, ...
- A file contains the specifications attached to the preselectionned Arduino card and the interruption routine specific for the microntroller.

a) Interrupt routine: This interruption routine is called at every 1ms ("ageing timer"). This interrution updates the clocks of every enabled tansition.

b) Setup routine: The setup procedure initiates the "ageing timer" and computes the incidence matrix C from *Pre* and *Post*, and initiates the input pins and the outputs pins of the microcontroler and initiates the serial communication.

CONCLUSION

This paper proposes a tool which allows to embed a Time Petri net into an Arduino card. This work needed the definition of a new semantics based on a combination of weak and strong semantics. The implementation of a model can valid (or not) performance constraints of the model relatively to a target architecture. Checking constraints on a prototype allows also to uncover constraints that might not have been foreseen in this initial specification. Time constraints can be asset more precisely.

In perspective, it could be possible to assign fragment of non blocking code to place or transition. Moreover, other types firing semantics can be tested and implemented. It also is possible to developp monitoring, via the serial communication.

For the problem of complexity, computing place invariant or unfolding techniques, could bring additional high level informations allowing to reduce the number of transitions to inspect, in the computing of enabled transition, improving the speed of the algorithm.

Furthermore, other boards based on new microcontrollers can be added. The development of Arduino boards is dymanic, recently, various boards have been proposed: Arduino Due, *M*0, 101, or Yún. These new boards are architectured on powerfull microcontroller(ARM core, x86, Atheros, ...) with higher frequencies (upto 400 mhz). These improvments promise better perfomances and use of bigger TPN.

References

[1] Attiogbe, C.: Semantic Embedding of Petri Nets into Event B. International IM_FMT, Dusseldorf, 2009.

[2] Berthomieu, B., Lime, D., Roux, O. H., Vernadat F.: Reachability problems and abstract state spaces for time Petri nets with stopwatches. Discrete Event Dynamic Systems 17, 133-158, 2007.

[3] Cerone A. and Schettini M.: Time-based expressivity of time Petri netds for system specification Therorical Computer Science, 216,(1-2) : 1-53, 1999.

[4] Kordon F.: Prototypage de systèmes parallèles partir de rèseaux de Petri colorés, application au langage Ada dans un environnement centralisé ou rparti Phd Thesis, Université Pierre et Marie Curie, Paris VI, 1992.

[5] Lee G., H. Zandong and J. Lee: Automatic generation of ladder diagram with control Petri Nets Journal of Intelligent Manufacturing, 15 (2), 2004.

[6] L. Ferrarini: An Incremental Approach to Logic Controller Design with Petri Nets ArticleinIEEE Transactions on Systems Man and Cybernetics 22(3): 461 - 473, 1992.

[7] C. Girault and R. Valk: Petri nets for system Engineering, a guide to Modeling, Verification and Applications Springer, chapter 21 : 432-470 - 1998.

[8] Ramchandani, C.: Analysis of Asynchronous Concurrent Systems by Petri Nets. Cambridge: Massachusetts Institute of Technology. PhD Thesis, 1974.

[9] Merlin, P. M. and Farber, David J.: Recoverability of Communication Protocols–Implications of a Theoretical Study. IEEE Transactions on Communications, vol.24, no.9, 1036-1043, 1976.

[10] Sifakis, J.: Use of Petri Nets for Performance Evaluation. H. Beilner and E. Gelenbe, Measuring, Modeling and Evaluation of Computer Systems, 75-93, 1977.

[11] Petri, C.A.: Kommunikation mit Automaten. Bonn: Institut fur Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

[12] Taubner, D: On the implementation of Petri nets. Article in Advances in Petri Nets, Rozenberg, Grzegorz, Springer Berlin Heidelberg, pp 418-439, 1988.

[13] Delfieu, D. , Comlan M., Sogbohossou M.: Unfolding of time Petri nets for quantitative time analysis. Sixth International Conference on Advances in System Testing and Validation Lifecycle, 21-27, 2014.

[14] Tomáš Richta, V. J. and R. Kočí (2013).: Petri nets-based development of dynamically reconfigurable embedded systems. In D. Moldt and H. Rlke (Eds.), Petri Nets and Software Engineering. International Workshop, PNSE'13, Milano, Italy, Proceedings. Volume 989 of CEUR Workshop Proceedings, pp. 203–217, 2013.

Appendix

The next figure, that illustrates globally the approach, includes four frame areas : One, for the edition of the Time Petri nets, a second for *PN2A* (pin association and code generation), a third for Arduino IDE and the last one for he serial monitor that allows to supervise the automaton.

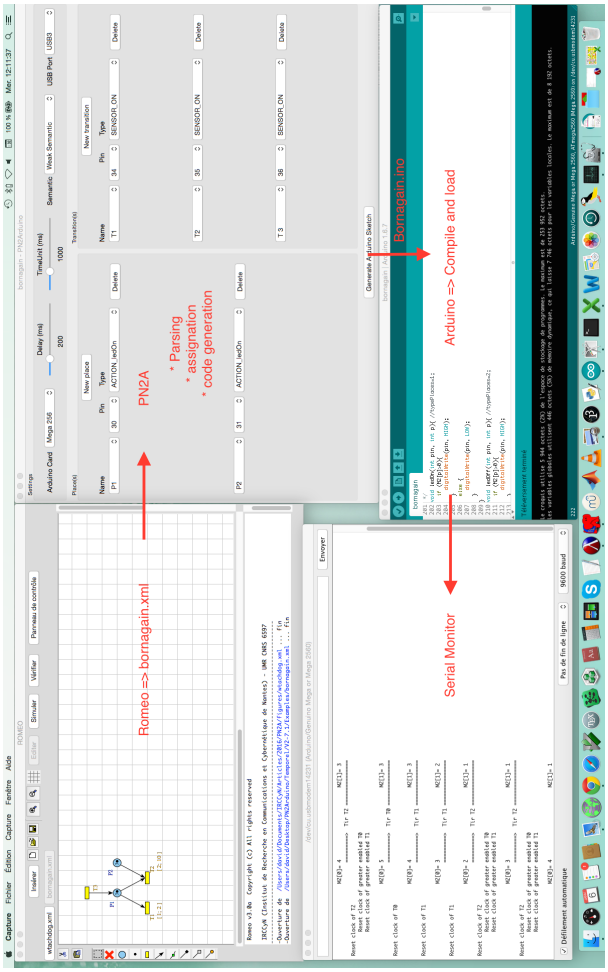

Fig. 3. From the edition to the execution step

In the next figure a practical lab is connected to an Arduino card (Atmega 2560). Leds, bistable switches provides interactions to the TPN.

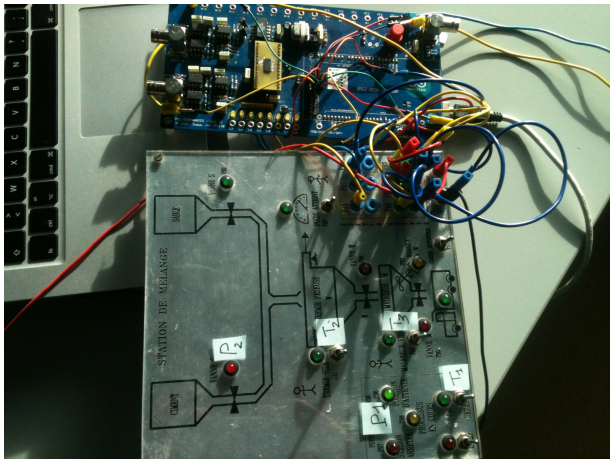

Fig. 4. Arduino card