Petri Nets to Arduino (PN2A) Embedding Time Petri Nets into a Microcontroller Architecture

David Delfieu, Maurice Comlan

Petri Nets to Arduino (PN2A)
Embedding Time Petri Nets Into a Microcontroller Architecture

David Delfieu and Maurice Comlan

Abstract This paper presents a tool which embeds Time Petri Nets into Arduino micro-controller boards. Time elapsing is mimicked by a timer that is aging a set of clock variables. Embedding a network makes it not autonomous. Model simulations are now truly dependent and time-dependent of external stimuli on micro-controller pins. Indeed, firings of transitions are now also conditioned to predefined events observed on the pins. This paper defines the semantics, the algorithm of the token player, describes the code generation and the use of the tool.

Keywords Embedded systems · Time Petri Nets · Firing semantics · Code generation · Microcontroller

1 Introduction

Embedded systems are increasingly complex; they run on mono or multiprocessors, parallel or distributed architectures and can be submitted to synchronisation, concurrency, or communication constraints. By their complexity, such systems are difficult to tackle and it is necessary to have reliable modeling tools, for one hand, to design and specify them and for another hand, to check that a set of properties is established. For example, prevention of deadlock, to lay down that the capacity of a buffer is never exceeded, ... The objective is to ensure the safety and reliability of the system and the modeling will attest the conformity of the behavior to the requirements. Formal approaches produce modeling of the behavior of systems from which, proves can be lead. As a modeling is an abstract and approximated

David Delfieu
Laboratoire des Sciences du Numérique de Nantes 1, rue de la Noë, BP 92101 44321 Nantes Cedex 3
E-mail: david.delfieu@univ-nantes.fr

Maurice Comlan
Ecole Polytechnique d'Abomey-Calavi 01 BP 2009 Cotonou
E-mail: comlan@hotmail.fr

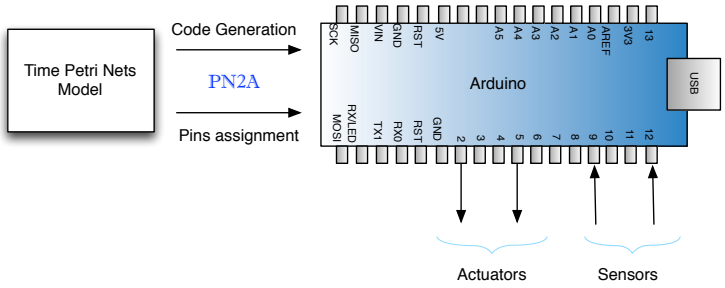

Fig. 1 Model and its environment

representation of a real system, each modeling technics will have its own merits and weaknesses.

Petri nets (PN) [**?**] constitute one of the well-known formalisms to model in a compact and explicit way the concurrency and the synchronisation between the dynamic components of discrete-event systems. Petri nets are a modeling language covering a wide domain: control systems, communication protocols, distributed systems. The properties of reachability, deadlock or boundedness are critical properties and Petri Nets are particularly an appropriate modeling tool because those properties, although expensive, are generally decidable. Several extensions even enlarge its application to Artificial Intelligence, Object Programming with Colored PN, or real-time with temporal extensions. This paper focuses on Time Petri Nets (TPN), particularly adapted to model real-time concurrent systems. In these models gigantic state space models are computed. To keep a certain compactness a graph of zones is computed. To each zone corresponds a set of variables and time inequalities. The passage of time is event driven. It is the user who acts on the transitions that makes time evolve. We are more dealing with a logical time rather than a real time.

Generally the modeling of a system with Petri nets carries only on the control part. It is often, 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 comes necessary to test performance constraints or implementation aspects, the environment is modelled and coupled to the application in a global model. Therefore it is necessary to know very precisely the environment and the target architecture.

This paper describes a tool $PN2A$, which allows the modeling of a system using Time Petri net embedded in a microprocessor whose external environment interacts with the model. This embedding may allow to test external equipments, or may reveal unexpected problems or artefacts (response time, electromagnetic interference, . . .). The general idea can be summarized by the following Figure (1). This figures describes the association connecting the modeling with the physical peripherals: Actuators, sensors, motors, leds, human interactions. The modeling then can be simulated in a real context. A TPN is first edited with Romeo [**?**] or Tina [**?**]. PN2A can open a Romeo or Tina project and parses the TPN into a structure. Then $PN2A$ may define an assignation between pins and transitions

or places. Then a code generation step produces enabling and firing functions. A timer handler is added to the code to deal with temporal aspects. An Arduino project is generated, ready to be uploaded to the target (see Figure 1).

Section 2 presents different code generation approaches from Petri nets and how the approach, presented in this paper, can be situated. Section 3 recalls basics definitions and propose a way to define the embedding of a TPN on a microcontroller board. The next section presents the firing semantics, and the last section describes the tool and illustrates the approach by an example with metrics about execution times and memory occupation.

2 Discussion and related works

Many studies discuss about code generation from Petri Nets. [**?**] proposes a classification of theses approaches between totally centralized, centralized and hybrid approach. In totally decentralized approach [**?**], 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 [**?**] 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 byte code interpreted by a Petri nets Virtual Machine which is part of a Petri nets dedicated operating system. This approach is much more sophisticated (use of virtual machine), than the approach presented in this paper, however the memory amount is very important for even small modeling. 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.

Centralized approaches are implementing "token players". At each step there is an evaluation of the firability of every transition. Lee G., H. Zandong and J. Lee [**?**] 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 [**?**], Ferrarini proposes to synthesize logic controller from Petri nets, but a lot of restrictions are imposed on the Petri nets : The whole Petri nets must be covered by a place invariant and a transition invariant and the net must be alive. They consider Petri nets without parallelism.

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 identified by the computing of place invariants.

This work is part of the centralized approaches. A token player is implemented which stands for a scheduler. Moreover, this approach deals with general Time Petri nets with parallelism. Unsafe TPN with non free choice conflicts are considered. In this approach Time Petri nets are translated into an incidence matrix and a set of clock variables. In this approach, there is no need to implement subprocesses. The parallelism is implicit within the matrix structure and the firing equations (defined in the following Section 3.1). The generated code is directly uploadable to the target and executable without any operating system.

3 Basic definitions

3.1 Petri Net

A Petri Net (PN) [**?**] is a structure (P, T, F) where P is a finite set of places, T, a finite set of transitions, (where $P \cap T = \emptyset$), and $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs*. $pre : (P \times T) \longrightarrow \mathbb{N}$, the Pre-condition function defining incoming arcs and their valuations, between a place and a transition and $post : (T \times P) \longrightarrow \mathbb{N}$, the Post-condition relation defining outgoing arcs (and their valuations) between a transition and a place. Places and transitions are called nodes. The pre-set of a node x is denoted ${}^\bullet x = \{y \in P \cup T \ : \ (y, x) \in F\}$ and the post-set of a node x is denoted $x^\bullet = \{y \in P \cup T \ : \ (x, y) \in F\}$.

A marking of a Petri net $\mathcal{N}$ is a mapping $M : P \longrightarrow \mathbb{N}$. A marked Petri Net is a tuple $< \mathcal{N}, M_0 >$ where $\mathcal{N}$ is a Petri net and M_0 is the initial marking. A transition $t \in T$ is *enabled* in a marked Petri Net M iff: $\forall p \in {}^\bullet t, \ M(p) \geq pre(, t))$. This is denoted: $M \xrightarrow{t}$. Firing of an enabled transition t leads to the new marking M' ($M \xrightarrow{t} M'$): $\forall p \in P, \ M'(p) = M(p) - pre(p, t) + post(t, p)$ Thus the following set can de defined: $Enabled(M) = \{t \in T, \text{ s.t. } M \xrightarrow{t}\}$. The next section introduce several temporal extensions of Petri Net.

3.2 Time and Petri nets

Extensions have been proposed to take into account temporal specifications: In Timed Petri nets a duration is associated to a transition while for time Petri nets a temporal interval is associated to transitions. Concerning Time Petri nets, time can be associated to places, arcs or transitions: P-Timed Petri nets (P-TPN) [**?**], Arc-Timed Petri Nets (A-TPN) [**?**] and Time Petri nets (T-TPN) [**?**,**?**] are the principal variants.

In term of expressivity, on a theoretical point of view, P-TPN et T-TPN are not comparable, A-TPN being more expressive [**?**]. But in term of conciseness T-TPN are much more concise than P-TPN or A-TPN. In term of language they are all equivalent. Concerning T-TPN other extensions have been introduced to enrich expressive power: Scheduling TPN, inhibitors/reset Arcs TPN, and some other ones. In theses extensions, the 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 marked Time Petri net *is a tuple* $(\mathcal{N}, M_0, I_s)$ *in which* $(\mathcal{N}, M_0)$ *is a marked Petri net and* $I_s : T \to Q^+ \times Q^+ \cup \infty$ *is called the Static Interval Function.*

For every transition t associated to a static interval $[efd(t), lfd(t)]$: $efd(t)$ defines the earliest firing date of t, while $lfd(t)$ is the latest firing date of t.

The *enabling date* of a transition is the date of the last firing 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 R^+. In this paper, time elapsing is generated by a timer, thus, clock variables will be defined on N^+.

With time Petri Nets, two semantics can be considered [**?**]: In a strong semantics, after $lfd(t)$, the firing of the transition t will be unavoidable. In a weak semantics, this transition can expire if its clock exceeds $lfd(t)$ and the tokens of incoming places will die. An in-depth discussion about semantics will be tackled in section 4.1.

3.3 Microcontroller synchronized Time Petri net

The external environment of the microcontroller constitutes (see Figure 1), the real environment of the model. Embedding a model therefore requires the definition of an association between the nodes of the TPN and the set of actuators or sensors that defines the real environment of the micro-controller . The evolution of control in a Petri net is depicted by the set of tokens and their evolutions. Thus it is proposed to associate marked places to predefined output pins whereas transitions can be assigned to predefined input pins.

Let's define this assignation: A transition can be assigned to the high or the low level of an assigned pin. As well, a marked place will produce a high or a low level on an assigned pin.It is important to note that the assignation of transitions and places to the pins of the microcontroller can be partial. In the Figure 2, the read nodes ($EnterCode$, $Pass$, , . . .) are assigned to pins while the blues ones (T_1, P_2, . . .) are not .

Let's note T^a (resp. P^a), the set of transitions $t \in T$ (resp. $t \in P$) such as t (resp. p) has been assigned to a pin of the micro-controller and $Pins$ the set of pins of a micro-controller .

Definition 2 A microcontroller synchronized Time Petri net ($mstpn$) is defined by the *tuple* $< \mathcal{N}^t, \mathcal{A}(), T^a, P^a >$ in which $\mathcal{N}^t$ *is a Time Petri net*, T^a (resp. P^a) a subset of T (resp. P) which are assigned to a pin of the microcontroller by the the assignment application $\mathcal{A}() : P^a \cup T^a \rightarrow < Pin\ number,\ Level >$.

If $n \in P^a \cup T^a$, let's note $\mathcal{A}(n) \rightarrow pin$ the assigned pin and $\mathcal{A}(n) \rightarrow L$ the associated tension level of the assigned pin to the node n. The next definitions arise from the previous:

- In a msTPN, a transition t can become *urgent*, if this transition is not assigned to a pin ($t \notin T^a$) and when its local clock reaches its latest firing date.
- A transition t belonging to T^a is *sensitized* if the level observed onto $\mathcal{A}(t) \rightarrow pin$ matches to $\mathcal{A}(t) \rightarrow L$.

4 Semantics

The example (Figure 2) models the identification system of a cash dispenser. In this Figure red transitions $Pass$, $Retry$ $InsertedCard$, $EjectedCard$ are assigned to input pins, while red places $EnterCode$, and $InCard$ are places assigned to output pins. The blues nodes are not assigned. A user has 3 tries for being identified (to reach the place P_2). If he fails (place P_3), the tokens in place $Tries$ give

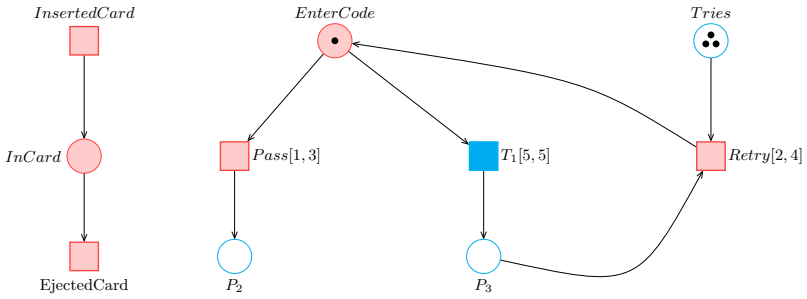

Fig. 2 Cash Dispenser

him additional chances to retry. Reds nodes are assigned to pins while blue ones are not. The temporal intervals allow to implement a watchdog. T_1 monitors the transition *Pass*. if the transition *Pass* occurs into $[1, 3]$ the control goes in P_2 else, if the elapsing of time exceeds 5, *Retry* is fired and the control enters again the place *EnterCode*. It is interesting to note that from now on the firing of $Pass$ is determined by three conditions: The occurrence a positive edge on its input pin, its local clock and the marking of the place *enterCode*. Whereas, the firing condition of T_1 is only bounded to its local clock and the marking of *enterCode*.

If it sounds pertinent to consider a strong semantics in the firing of T_1, what is the associated semantics to the firing of $PASS$? Consider the case, where the clock variable associated to $Pass$, reaches its $lfd()$, should we consider that the token in P_3 should die, should be urgently fired, or remains available for T_1 ? The following section discusses on the semantics to apply to these two types of transitions.

4.1 Discussion about Firing semantics

In this discussion, we analyze the firing semantics for assigned and non assigned transitions.

Non assigned transitions

When the designer of the model has chosen not to assign T_1 (Figure 2), he therefore, has decided that the firing condition of T_1 is only conditioned to its clock variable. In that case, it is interesting to consider a Strong Semantics and to force the firing at $efd(T_1)$. In the opposite case, the set of tokens belonging to ${}^\bullet T_1$ should stay in the net, becoming unavailable, cluttering the net.

Assigned transitions

For assigned transitions, we examine a weak semantics. In the simplified example of the Figure 3, the only assigned node (PIN_6) is T_1. T_2 acts as a watchdog on T_1.

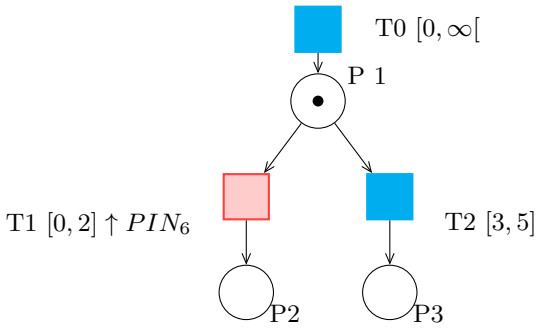

Fig. 3 Discussion about firing semantic

Let's consider two scenarii: First, T_0 is fired at date 0 and 3, T_2 and a positive edge on PIN_6 occurs at the date 5. In this scenario, a first token appears in place P_1 at date 0, a second token is produced at date 3. At this date, the clock variable of the first token is 3 and the second token has its clock variable at zero. At date 5, T_1 is fired with the older token in P_1 while T_2 is fired with the other token. In the second scenario, we consider the same temporal sequence except that the positive level on PIN_6 does not occur: At the date 5, there is two tokens in the place P_1. But in this case, only T_2 is fired and the older token will stay in the net.

In a weak semantics, the expiration of a delay, before the occurrence of the associated event, signifies that internal conditions (time elapsing) prevails over external conditions. Thus, weak semantics brings an interesting expressive power. For example, tokens can be assimilated to fresh products, submitted to cold chain integrity, and an upper firing date determines the validity of the cold condition on a process. If the date has passed, the products (tokens) becomes unusable for some transitions or for all.

On the contrary, this expressive power brings a great complexity: clocks must be attached to each token. Moreover, the enabling conditions must take into account the multiple and different clocks of several tokens. Some tokens will be usable while other not. Moreover, dead tokens can accumulate, occupying memory and slowing down the tests. Tests on examples with dead or unusable tokens have shown that the induced complexity impacts significantly the response time of the system. We propose to use in this paper a strong semantics for the two type of transitions. Moreover, this semantics allows to attach clock variables to transitions rather than tokens.

4.2 Definition of the firing semantics

In a $msTPN$, the firing of a connected transition depends on three conditions: i) its enabling condition, ii) the value of its local clock and iiii) an edge signal (positive or negative) observed on its assigned pin. For a non connected transition, the first two conditions are necessary.

The semantics of a microcontroller synchronized Time Petri net can be stated as a synchronized timed transition system which is defined in the following definition (Definition 4). In this semantics clock variables are attached to transition

Definition 3 (Clock variables) For every transition t in T, v associates a positive integer representing a time elapsing. $v : T \rightarrow \mathbb{N}$, vector defining all the clock variables. v_0 : null vector, allows to initialize every clock variables. $< M, v > \xrightarrow{\theta} < M, v' >$ expresses the firing of t from the marking M, with M' and v' respectively the new marking and the new clock vector.

A transition is said *newly enabled* when it has just became enabled. The vector $newlyEnabled : T \rightarrow \mathbb{B}$ is a boolean vector defining if all the transitions that are newly enabled. Initially, all the values of this vector are false. When a transition is *newly enabled* there is a reset of its clock variable. If this transition is enabled again by additional tokens, the clock variable is not modified and the transition can be fired with any subset of tokens of its preset. Tokens are therefore undifferentiated. This choice allows to prevent the use of clock variables on tokens.

Definition 4 (Semantics of a msTPN) The semantics of a msTPN $\mathcal{N}$ is defined by the transition system: $< Q, \{q_0\}, \Sigma, \rightarrow >$

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

  - Computing of $Urgent$:

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

  - Urgent transitions:

$$< M, v > \xrightarrow{t_i} < M', v' > \ \textbf{iff} \begin{cases} \textbf{if } \begin{cases} M \geq^{\bullet} t_i \\ t_i \in Urgent \end{cases} \\ \textbf{then } M' := M -^{\bullet} t_i + t_i^{\bullet} \end{cases}$$

  - Non urgent transitions, assigned or not:

$$< M, v > \xrightarrow{t_i} < M', v' > \ \textbf{iff} \begin{cases} \textbf{if } \begin{cases} M \geq^{\bullet} t_i \\ efd(t_i) \leq v(t_i) \leq lfd(t_i) \\ (t_k \notin T^a) \ \| \ (sensitized(t_i) == True) \end{cases} \\ \textbf{then } M' := M -^{\bullet} t_i + t_i^{\bullet} \end{cases}$$

  - Computing newlyEnabled and reset of clocks:

$$\begin{cases} \forall t_k \in T, \ \ newlyEnabled(t_k) = (M' > {}^{\bullet}(t_k)) \ \wedge \neg(newlyEnabled(t_k)) \\ \\ v'(t_k) := \begin{cases} 0 \ \textbf{if } (newlyEnabled(t_k)) \\ \textbf{else } v(t_k) \end{cases} \end{cases}$$

In this definition, the first item defines the $Urgent$ set. The second defines the firing of an urgent transition. This item defines the new marking and modifies the newlyEnabled vector. Moreover it reset the clock variable of a transition if it newlyEnabled.

The third item differs in its firing condition (upper parenthesis). In that case, it adds the condition $(t_i \notin T^a) \parallel (sensitized(t_i) == True)$. This condition is true when either $(t_i \notin T^a)$, either if $(t_i \in T^a)$ then the condition $(sensitized(t_i) == True)$ must be true.

5 The tool

PN2A[1], is a software where binary executables are available for Windows, Mac OS and Linux environments. Input files describing the TPN can be edited by two PN editors: Tina[2] and Romeo [3].

We illustrate the use of the tool on the previous example (Figure 2). *Pass*, *Retry* have been assigned to switches, while *EnterCode* and *Tries* are assigned to leds (see Figure 7). Other transitions or places were not assigned. T_1 implements a watchdog. The edition of the example is made on Romeo (Figure 4) and saved on an xml file (cdRevue.xml):

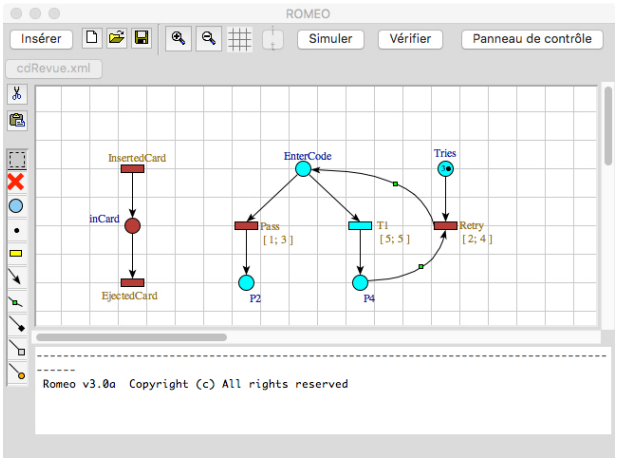

Fig. 4 Edition

PN2A proposes an IDE constituted by a main window where the user chooses a micro-controller card, and defines the assignment (Figure 5) between places, transitions and pins. The user can adjust the delay of the cycle and the time unit. Pressing the button "Generate Arduino sketch" generates the code downloadable directly through Arduino IDE. The edited file is open on PN2A and an the assignation process is made:

[1] Freely available at pn2A.rts-software.org/
[2] Freely available at www.laas.fr/tina/
[3] Freely available at romeo.rts-software.org/

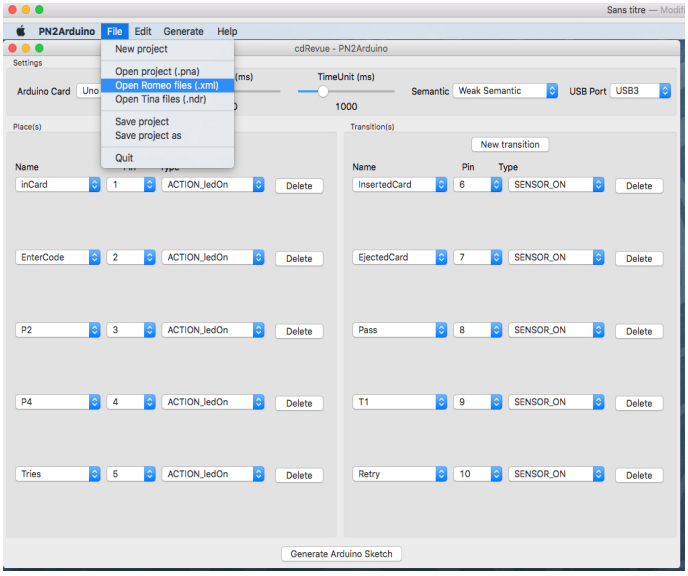

Fig. 5 Opening a romeo file in PN2A

Romeo produces xml files, while Tina produces text files. The parsing of those files produces the matrices: pre, $post$, M_0, F, and $efd()$ and $lfd()$ that defines the structure of the TPN. The code generation of the example produce an Arduino sketch (see cdRevue.ino in Figure 6):

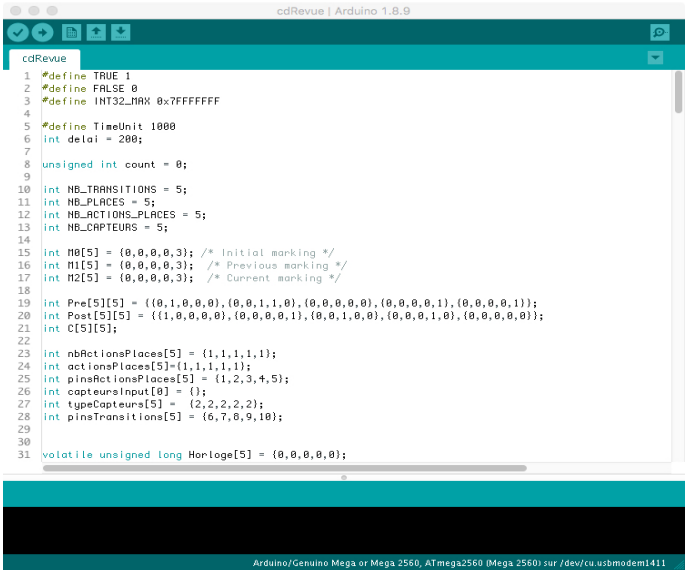

Fig. 6 Opening the generated code in Arduino

Testing the model with switches and leds in Figure 7:

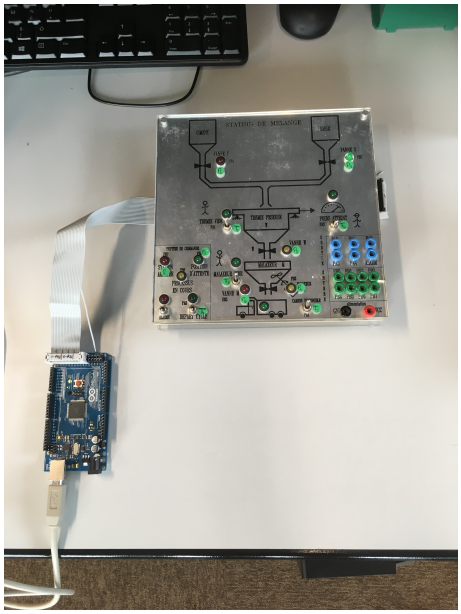

Fig. 7 Test of the model on an Arduino card

5.1 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 clocks variables when transitions are selected by the user, there is no a real-time clock that ages transitions. In the initial state of Figure 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 the real date 5.

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

5.2 Complexity and metrics

The greater complexity of the algorithm is the computing of $Enabled(t)$. The algorithm contains intricate loops in $|P| * |T| * |T^a|$ and a loop in $|T^u|$ (urgent transitions). Then the complexity is polynomial in order of $a.n^3 + b.n + c$ with $n = max(|P|, |T|)$.

For an Atmega2560 with a 16 MHz crystal oscillator, table 1 presents the execution times and memory amount of the embedded TPN corresponding to the example. The third column give execution times of the algorithm (see the algorithm 1) 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, proportionally 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/output of the Atmega2560. It is important to note that every transition may not be assigned to a pin of the micro-controller .

Table 1 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.

6 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 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 of firing semantics can be tested and implemented. It also is possible to develop monitoring, via the serial communication. For the problem of complexity, off-line computing of 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 micro-controllers can be added. The development of Arduino boards is dynamic, recently, various boards have been proposed: Arduino Due, $M0$, 101, or Yún. These new boards are based on powerful micro-controller (ARM core, x86, Atheros, ...) with higher frequencies (up to 400 MHz). These improvements promise better performances and use of bigger TPN.