Interpreted Synchronous Extension of Time Petri Nets - Definition, Semantics and Formal Analysis

Karen Godary-Dejean, Hélène Leroux, David Andreu

▶ **To cite this version:**

Karen Godary-Dejean, Hélène Leroux, David Andreu. Interpreted Synchronous Extension of Time Petri Nets - Definition, Semantics and Formal Analysis. Discrete Event Dynamic Systems, 2022, 32, pp.27-64. 10.1007/s10626-021-00347-z . lirmm-03454630

HAL Id: lirmm-03454630
https://hal-lirmm.ccsd.cnrs.fr/lirmm-03454630

Submitted on 29 Nov 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Discrete Event Dynamic Systems
Interpreted Synchronous Extension of Time Petri Nets

Definition, Semantics and Formal Analysis
--Manuscript Draft--

Manuscript Number:	
Full Title:	Interpreted Synchronous Extension of Time Petri Nets Definition, Semantics and Formal Analysis
Article Type:	Manuscript
Keywords:	Critical embedded systems; Implementation of Petri nets; Synchronous Time Petri nets semantics; Bisimulation
Corresponding Author:	Karen GODARY DEJEAN, Ph.D. University of Montpellier: Universite de Montpellier Montpellier, Languedoc-Roussillon FRANCE
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	University of Montpellier: Universite de Montpellier
Corresponding Author's Secondary Institution:	
First Author:	Karen GODARY DEJEAN, Ph.D.
First Author Secondary Information:	
Order of Authors:	Karen GODARY DEJEAN, Ph.D.
	Hélène LEROUX, Ph.D.
	David ANDREU, Ph.D. HdR
Order of Authors Secondary Information:	
Funding Information:	
Abstract:	Our work is integrated into a global methodology to design synchronously executed embedded critical systems. It is used for the development of medical devices implanted into human body to perform functional electrical stimulation solutions (used in pacemakers, deep brain stimulation...). These systems are of course critical and real time, and the reliability of their behaviors must be guaranteed. These medical devices are implemented into a programmable logic circuit in a synchronous way, which allows efficient implementation (space, consumption and actual parallelism of tasks execution). This paper presents a solution that helps to prove that the behavior of the implemented system respects a set of properties, using Petri nets for modeling and analysis purposes. But one problem in formal methods is that the hardware target and the implementation strategy can have an influence on the execution of the system, but is usually not considered in the modeling and verification processes. Resolving this issue is the goal of this article. Our work has two main results: an operational one, and a theoretical one. First, we can now design critical controllers with hard safety or real time constraints, being sure the behavior is still guaranteed during the execution. Second, this work broadens the scope of expressivity and analyzability of Petri nets extensions. Until then, none managed in the same formalism, both for modeling and analysis, all the characteristics we have considered (weights on arcs, specific test and inhibitor arcs, interpretation, and time intervals, including the management of effective conflicts and the blocking of transitions).

Powered by Editorial Manager® and ProduXion Manager® from Aries Systems Corporation

Manuscript

Click here to view linked References

Click here to access/download;Manuscript;GodaryDejean_JDEDS.pdf

DEDS manuscript No.
(will be inserted by the editor)

Interpreted Synchronous Extension of Time Petri Nets

Definition, Semantics and Formal Analysis

Karen Godary-Dejean · Hélène Leroux · David Andreu

Received: date / Accepted: date

Abstract Our work is integrated into a global methodology to design synchronously executed embedded critical systems. It is used for the development of medical devices implanted into human body to perform functional electrical stimulation solutions (used in pacemakers, deep brain stimulation...). These systems are of course critical and real time, and the reliability of their behaviors must be guaranteed. These medical devices are implemented into a programmable logic circuit in a synchronous way, which allows efficient implementation (space, consumption and actual parallelism of tasks execution). This paper presents a solution that helps to prove that the behavior of the implemented system respects a set of properties, using Petri nets for modeling and analysis purposes. But one problem in formal methods is that the hardware target and the implementation strategy can have an influence on the execution of the system, but is usually not considered in the modeling and verification processes. Resolving this issue is the goal of this article. Our work has two main results: an operational one, and a theoretical one. First, we can now design critical controllers with hard safety or real time constraints, being sure the behavior is still guaranteed during the execution. Second, this work broadens the scope of expressivity and analyzability of Petri nets extensions. Until then, none managed in the same formalism, both for modeling and analysis, all the characteristics we have considered (weights on arcs, specific test and inhibitor arcs, interpretation, and time intervals, including the management of effective conflicts and the blocking of transitions).

Karen Godary-Dejean
LIRMM UMR 5506, Univ. Montpellier, CNRS, Montpellier, France.
E-mail: karen.godary-dejean@umontpellier.fr
ORCID ID : 0000-0002-5835-021X

Hélène Leroux
Lycée International François 1er, Fontainebleau, France.

David Andreu
LIRMM UMR 5506, Univ. Montpellier, CNRS, Montpellier, France.
NEURINNOV company `http://neurinnov.com/`

Keywords Critical embedded systems · Implementation of Petri nets · Synchronous Time Petri nets semantics · Bisimulation

1 Introduction

1.1 Contexte

In the context of critical embedded systems, such as avionics, automotive and medical, it is imperative to prove that requirements are met, whether they are regulatory or normative requirements, or performance ones. To achieve this, design and development methodologies are rigorous and testing is as thorough as possible. In most cases, these critical systems must be certified before industrial and commercial operation. Within the framework of this certification, the potential contributions of formal methods are studied, not as a substitute for testing at this stage, but as a complement. This is the case, for example, with RTCA DO-333, ”Formal Methods Supplement to DO-178C and DO-278A”. The use of formal methods includes, for example, the use of theorem proving, model checking, or abstract interpretation.

Our work is part of such an approach, i.e. to contribute to the proof of satisfaction of requirements, more particularly those related to performance and reliability. In other words, we are interested in building a tool-based methodology that helps to prove that the behavior of the implemented system respects a set of formally validated properties. This work therefore falls within the scope of the HILECOP (High-Level Hardware Component Programming) methodology, which was initially designed to assist in the development of safe active implantable medical devices (Andreu et al (2009); Leroux et al (2015)). These devices are implanted into human body to perform FES (Functional Electrical Stimulation) solutions which are being successfully used in an increasing number of applications, including pacemakers, deep brain stimulation, pain control and hearing restoration.

Therefore, the designed systems are submitted to usual constraints like dimensions and energy consumption, as well as more specific ones as electrical safety, intrinsic reliability, without omitting the need to deal with architectural and programming complexity, or communication needs. Mostly, these systems are critical and real time as the reliability of their behaviors must be guaranteed, including in a temporal point of view: the stimulation must be turned on and off at very specific times. Also, beyond the medical field, this methodology could be useful for any system with similar constraints: avionics, automotive, spatial, etc.

The HILECOP methodology deals with the complexity of digital systems thanks to a component modular approach as well as the use of a formal language (Petri nets) for modeling the behavior and composing the components. Formal languages offer the possibility of using formal validation methods to deal with the critical constraints of the targeted systems. Formal methods are complementary to more usual methods (as test and simulation), providing more complete validation results at an earlier step of the design process. Finally, the embedded and real time constraints are managed by selecting a specific execution target: FPGA (Field-Programmable Gate Arrays) or ASIC (Application Specific Integrated Circuit), which allows efficient implementation in terms of space and consumption, and time efficiency thanks to the actual parallelism of tasks execution.

The very critical aspect of our system incites us to use formal methods, especially model checking. Model checking (Baier and Katoen (2008)) is based on the exhaustive enumeration of all the reachable states of a model of the system. It is then possible to verify specific properties (safety, liveness, ..) to validate the system behavior. But the usual issue of the model checking approach is that properties are verified on a model, not on the system itself. The programming step can introduce errors. This problem is managed in the HILECOP methodology, as the programming step is done with an automatic generation of the VHDL code corresponding to the model (Andreu et al (2008); Leroux et al (2015)). Another problem resides in the hardware target, which can has an influence on the execution of the system as it is usually not considered at the modeling level. To be consistent, the model and the validation method must finely consider the implementation constraints imposed by the hardware target and the implementation strategy.

Thus, taking into account implementation and execution constraints into the modeling and validation steps is the goal of this article. We show here that our methodology allows to formally verify some properties on the system model, which remain guaranteed at the execution step. The next section presents the implementation problematic and choices we have done for modeling and analysis purpose.

1.2 Implementation and execution issues

1.2.1 Interpretation problematic

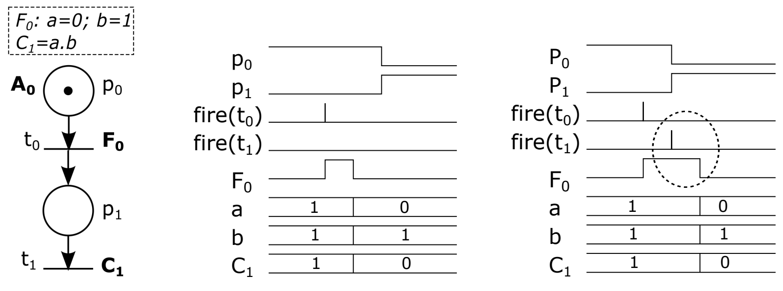
In the HILECOP methodology, Petri nets (PN) are used for the design of industrial systems. The designed model has a real signification: it is linked with the real world as it will be implemented on industrial products. For example for the design of an implanted medical controller for FES (Andreu et al (2009)), the PN places will represent the states of the controller, and are directly associated to actions executed by the neural stimulator. Transitions between these states are linked with events or variables of the system environment (for example sensor values or external commands). Then the behavior of the system, and therefore the evolution of the model, depend on the environment events and the variable values. This specific use of PN can not be ignored in a global methodology where the correct behavior of the implemented system must be guaranteed by the analysis of its designed model. Thus, we have to integrate the **interpretation** characteristics into the formal definition and semantics of our modeling formalism.

In our context, interpretation is composed of conditions, continuous actions and impulsive actions, which handle signals and variables coming from the system environment. Conditions allow to restrict the evolution of the PN model with signals or variables. In the form of a logical expression, a condition is associated with a transition, and then this transition is fired only if the associated condition is true. This introduces a particularity into the semantics of the Petri nets: to be fired, a transition must not only be enabled by the marking of its input places, but it must also take into account its associated condition. Actions allows to handle signals or variables. Continuous actions are associated to places and are executed as long as one of its associated places is marked. Impulsive actions are associated to transitions, and are executed once when one of their associated transitions is fired. Incrementing a counter is a typical example of such impulsive actions. An example of such a PN, named an Interpreted PN (IPN), is given in figure 1a, with a continuous action A_0 associated to the place p_0, an impulsive action F_0 associated to the transition t_0 which set the internal variables a to 0 and b to 1, and a condition $C_1 = a.b$ associated to the transition t_1.

1.2.2 Synchronous implementation problematic

In programmable logic devices (such as FPGA) context, the asynchronous implementation of PN is well-known (for example in Uzam et al (2009); Wegrzyn et al (2014)). But the addition of interpretation makes it difficult as VHDL is not executed sequentially but combinatorially. Indeed, impulsive actions could handle internal variables and then modify the condition values. Yet it is necessary to guaranty that the signals are stables to have a deterministic behavior, but there is no (automatic) way to exactly know when the impulsive actions are finished (Leroux et al (2015)).

For example for the interpreted PN of figure 1a, the correct behavior should be the one of figure 1b: t_0 is fired, which starts the impulsive action F_0; at the end of F_0, a is set to 0; in the same time the markings of the places p_0 and p_1 are evolving; then, when t_1 becomes enabled (i.e. when p_1 is marked), the evaluation of the condition C_1 is done: C_1 is now $false$, preventing the firing of t_1. But if the execution time of F_0 is longer than the evolution of the marking, the decision of firing t_1 could be taken before the stabilization of the C_1 value, leading to an unexpected behavior (figure 1c).



(a) IPN **(b)** Correct behavior **(c)** Incorrect behavior

Fig. 1 Incorrect behavior because of interpretation in asynchronous execution

One solution is to implement the interpreted PN in a synchronous way. The principle is that the evolution of the PN is driven by a clock. In our approach, the two clock edges are used, and the following steps concerning the evolution of the model states are repeated at each clock cycle (figure 2):

- On the falling edge of the clock ①: evaluation, from the current state, of which transitions have to be fired (i.e. they are enabled because of the marking, and firable depending on their condition values).
- On the rising edge of the clock ③: update of the marking depending on the previously fired transitions.
- Periods ② and ④ are necessary for the transmission and the stabilization of the signals and the variable values.

Continuous and impulsive actions are performed on stable states:

- Continuous actions are activated on the falling edge of the clock ①, once the marking of the places is stable, and are maintained as long as their associated places are marked.
- Impulsive actions are triggered on the rising edge ③ following the firing of the transitions they are associated to, and must end before the next falling edge (their maximum execution time is the half-period ④).

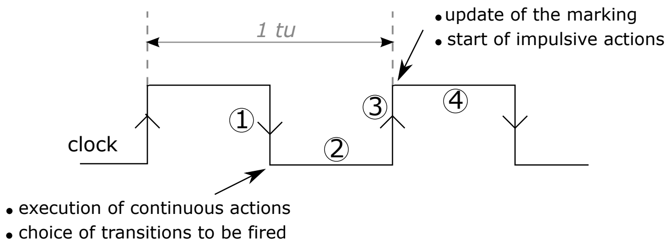

1 tu
• update of the marking
• start of impulsive actions
clock
①
②
③
④
• execution of continuous actions
• choice of transitions to be fired

Fig. 2 Principle of the synchronous implementation of IPN

The synchronous implementation in such a parallel execution target (FPGA or ASIC) specifically manages the parallelism: all the transitions firable at the same clock tick will be fired simultaneously, as shown figure 3: if t_0 and t_1 are firable at the initial state (i.e. C_0 and C_1 are `true`), then they are both fired on the first falling edge.

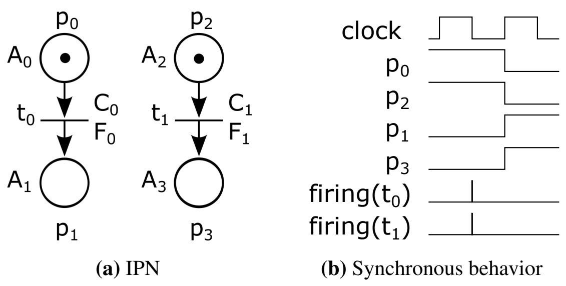


Fig. 3 Simultaneous firing of transitions in synchronous execution

But this behavior could be problematic in a Petri net, in case of conflicts. Informally speaking, a conflict is a situation in which a token could be used by several transitions at the same time. In synchronous execution, this could induce inconsistencies as one shared token could be used to fire two different transitions at the same time, potentially leading to an unacceptable behavior in case of a choice structure. Figure 4 illustrates this problem: transitions t_0 and t_1, which are in conflict, are both firable, and then are simultaneously fired in synchronous execution. It leads to the marking of both places p_1 and p_2, which is not the expected behavior of such a classical choice structure in Petri nets. Thus the synchronous execution of a Petri nets model must manage the simultaneous firing of the firable transitions while considering the conflict problem.

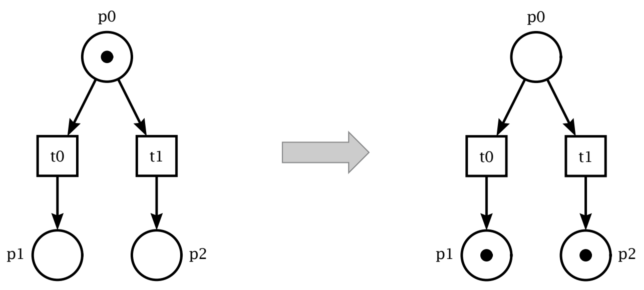


Fig. 4 Problem of an effective conflict in synchronous execution

1.3 State of the art of Petri nets formalisms

We have quickly present earlier the classical Petri nets, as well as their extension with interpretation. But in our context, the model must reflect the whole behavior of our targeted system. Thus it is necessary to integrate into the modeling formalism all the needed characteristics. We also have to study the analysis capacity of the formalism, as we want to use model checking techniques for properties verification.

The use of Petri nets is a natural choice as the actual parallelism of the FPGA target fits with the actual parallelism of PN. PN are a well-known formalism in the discrete event systems and control synthesis communities. Time Petri nets (TPN) (Merlin (1974)), a temporal extension of PN for quantitative time in which transitions are associated with firing intervals, are often used for modeling and analysis of real time systems (Girault and Valk (2013)). But we have seen that our need of determinism, as well as our specific implementation target, make it necessary to execute the Petri nets in a synchronous way, which does not fit with the classical asynchronous hypothesis of PN and TPN formalisms. Furthermore, the formalism we need must deal with interpretation, as the designed model is linked with the real world by variables and signals. We thus have to consider others PN extensions to deal with our constraints.

Few works deal with the formal definition and analysis of interpreted Petri nets. In the work done on SPIN (Signal Interpreted PN, Frey (2002)), as well as the one on CIPN (Control Interpreted PN, Grobelna and Adamski (2011)), interpretation is handled like the classic way in controller programming: it is associated as input with transitions and as output with places. But, in these cases, the execution hypothesis is that the transition firing is instantaneous, which is not the case in our context. Furthermore, both SPIN and CIPN solutions do not manage conflicts in a deterministic way, and do not allow quantitative time representation.

The interpretation could also be seen as synchronization: in the synchronized Petri nets formalism defined in (David and Alla (2010); Moalla et al (1978)), synchronization is considered as the association of transitions with event occurrences. Similarly, in Basile et al (2020), the input/output interpretation are associated with event occurences, mixing Interpreted and Synchronized Petri nets. Traditionally used for logic controllers specification and synthesis (Devillers and Van Begin (2006); David and Alla (2010)), synchronized PN have been used in various application domains, for example testing (Pocci et al (2016)) or fault diagnosis and control (Chen et al (2013)). But synchronized PN have some differences with our implementation choices, especially as they make the hypothesis that the firing execution time of transitions is instantaneous, and so do not consider the execution time for the associated impulsive actions. This could lead to problem in case of too long execution time as illustrated figure 1, and also could lead to instantaneous multiple firing of transitions (see David and Alla (2010)). A solution could be to combine timed PN and synchronized PN as in Huang et al (2018); Elidrissi et al (2020). But timed PN are different from temporal PN. The semantics describes in these articles uses timed transitions to represent a fixed duration of firing, and the decision of firing is done without considering the temporal information. Thus we could not represent a firing interval, which could be useful to represent the behavior of real time systems. Finally, Synchronized PN and Timed Synchronized PN also make the hypothesis that *two independent events never occur simultaneously* (David and Alla (2010)), which simplifies the problematic of simultaneous firings, reducing the problem to transitions associated to the same event (Pocci et al (2016)).

Furthermore, synchronized PN are a more general definition than needed in our context, where the events associated with transitions are the clock edges. In our context, clock edges are the only events that could trigger the firing of transitions, we do not need to represent more events. Thus the synchronous semantics we need seems closer to the discrete-time semantics of TPN (Popova (1991); Magnin et al (2008, 2009); Knapik et al (2010)) where time can not elapse more then $1tu$ (time unit) at a time. But this semantics is only a discrete interpretation of the time firing intervals of the TPN, while keeping the asynchronous firing hypothesis. Thus, the simultaneous firing of transitions is not managed.

So, the only way to be really close from our implementation choices is definitively to use the specific synchronous semantics of Petri nets. Unfortunately, few works deal with synchronous Petri nets, and even less with a precise and formal description of their corresponding behavior. In Hilal and Ladet (1993), Synchronous Petri nets (SynPN) are based on the traditional assumptions as for synchronous languages, which means that "*the firing cycle duration is considered as null*". But we think that this assumption is not realistic, as the marking evolution as well as the diffusion of the variable values are not instantaneous in a circuit. In Ribeiro and Fernandes (2007), the authors define synchronous interpreted Petri nets, named SIP-nets, in a close way of our implementation. But they do not consider the conflict problem, nor the quantitative time representation. Furthermore, their semantics is also simplified by the hypothesis of safe PN (no reachable marking can contain more than one token in any place).

The last formalisms that we must study are the generalized and extended extensions of Petri nets, which are quite common tools used to increase the expressive power of modeling. The theoretical definition of generalized Petri nets (i.e. it is possible to have several tokens in one place and weight on arcs) is well-known, but ultimately not often effectively used for the programming of logical controllers, where the assumption of safe Petri nets is currently made. In the same idea, the extension of Petri nets with test (or read) arcs and inhibitor arcs is common, even if they could limit the analysis possibilities (Busi (2002); Berthomieu et al (2007a); Ivanov et al (2014)). But these extensions have only been developed for asynchronous PN.

To resume, none of the existing Petri nets-based formalisms includes all the characteristics necessary to finely represent the reality of the hardware implementation we use. So we have to define a new formalism allowing all together the expression of interpretation, synchronous and parallel execution, quantitative time and expressiveness facilities.

1.4 Goal of this article

This article tries to answer to an industrial and concrete need: the modeling, analysis and synthesis of digital architectures for critical real-time embedded systems. For this purpose, we define a new extension of Petri nets, we named *Generalized Extended Interpreted Synchronous Priority Time Petri Nets: GEISPrT PN*. This formalism includes all the desired characteristics coming from our applicative context: expressiveness (generalized PN, with inhibitor and test arcs), interpretation (with the consideration of the duration of the signals and variables evolution as non-zero), deterministic synchronous execution (synchronized on clock edges, with simultaneous transitions firing and deterministic conflict management thanks to priorities), and quantitative time (representation of time constraints). This article presents in details the formal definition and semantics of the GEISPrT PN. It also presents some model transformation rules which allow to guarantee that the behavior of this formalism is included into the behavior of more classical (asynchronous) Petri nets, classically named TPN and more precisely in this article named GET PN (Generalized Extended Time Petri Nets). Thanks to that, some properties of a GEISPrT PN model could be verified with existing TPN analysis tools.

This article completes earlier works: a first definition of the formalism and the semantics of GEISPrT PN has been given in Leroux et al (2013, 2015)[1], but here these definitions have been refined with the consideration of the clock falling edge

[1] In these articles the GEISPrT PN formalism has been named IPrTPN.

events (expressing the synchronous aspect) and the explicit management of the interpretation. This semantics is therefore more precise, which was necessary to perform the formal proof of the semantics behavior preservation. One novelty of this article is indeed the formal proof of the inclusion of the GEISPrT PN behavior into the GET PN one (i.e. the ”classical” Petri nets), which is the necessary condition to obtain some verification results with existing analysis tools.

To simplify the explanation, the article is composed iteratively, first dealing with GEIS PN without conflict nor time, then introducing the conflict management and finally adding time intervals. Section 2 introduces the basis of our formalism: the Generalized Extended Interpreted Synchronous Petri Nets, first supposing that they are without conflict. We address the problematic of the analysis of this formalism in sections 3 and 4. The management of simultaneous firable transitions which are in conflict is added in section 5 both into the semantics and the analysis. And we finally study the addition of the quantitative time in section 6.

2 Generalized Extended Interpreted Synchronous Petri Nets

As presented in introduction, our context leads us to enhance the basic Petri Nets formalism with many characteristics that have to be considered for modeling but also for implementation and analysis purposes. It is then necessary to formally precise definition and semantics of such a formalism. For now, we only consider in this section interpretation and synchronous execution, without quantitative time consideration into the designed model. We also make the hypothesis in this section that our model does not has conflict: the firing of one transition could not prevent the firing of a simultaneous firable one. We integrate as a basis of our formalism specific characteristics on the arcs: weight could be associated to arcs, and test and inhibitor arcs are allowed. We think that these elements are essential to increase the modeling opportunities for industrial designers, and they are yet included in existing analysis tools.

2.1 Formal definition of GEIS PN

Definition 1 (GEIS PN) Let $\mathscr{C}$ the set of conditions, $\mathscr{F}$ the set of impulsive actions and $\mathscr{A}$ the set of continuous actions. A Generalized Extended Interpreted Synchronous Petri Net (GEIS PN) without conflict is a tuple $< P, T, Pre, Pre_t, Pre_i, Post, m_0, C, F, A, clk >$ where:

- $< P, T, Pre, Pre_t, Pre_i, Post, m_0 >$ is a generalized extended Petri net (David and Alla (2010)). P is the set of places, T the set of transitions and m_0 is the initial marking. $Pre, Pre_t, Pre_i, Post : T \to P \to \mathbb{N}$ are respectively the precondition, test, inhibition and postcondition functions[2].
- $C : T \to \mathscr{C} \to \{-1, 0, 1\}$ is the condition function. $\forall t \in T, \forall c \in \mathscr{C}, C(t)(c) = 1$ means that condition c is associated to t; $C(t)(c) = -1$ means that the negation

[2] For simplification, when the context does not lead to confusion, we will simplify the notation $Pre(t)(p)$ into $Pre(t)$, and respectively for the functions Pre_t, Pre_i and $Post$.

of the condition c is associated to t; and $C(t)(c) = 0$ means that condition c is not associated to t.
- $F : T \rightarrow \mathscr{F} \rightarrow \mathbb{B}$ is the impulsive action function. $\forall t \in T, \forall f \in \mathscr{F}, F(t)(f) = 1$ means that function f is associated to t, otherwise $F(t)(f) = 0$.
- $A : P \rightarrow \mathscr{A} \rightarrow \mathbb{B}$ is the continuous action function. It is defined on the same principle as F.
- $clk \in Clk$ is the clock signal that synchronizes the PN. The set of clock events is $Clk = \{\uparrow clk, \downarrow clk\}$, with $\uparrow clk$ the rising edge and $\downarrow clk$ the falling edge of the system clock.

Definition 2 (GEIS marking, state, enabled and firable) The *marking* of the GEIS PN is defined by the function $m : P \rightarrow \mathbb{N}$ such that $\forall p \in P,\ m(p)$ is the number of tokens in the place p. The definition of a transition *enabled* by a marking m, noted $t \in en(m)$, is:

$$t \in en(m) \ \Leftrightarrow\ \big(m \geq Pre(t) + Pre_t(t)\big) \wedge \big(m < Pre_i(t)\big)$$

The instantaneous value of a condition (i.e. its real value at each instant) is defined with the function $val : \mathscr{C} \rightarrow \mathbb{B}$. The value used to manage the system evolution (i.e. the one used on the following edge) is an image of this instantaneous value at the moment we read it, defined with the function $cond : \mathscr{C} \rightarrow \mathbb{B}$. The execution of the impulsive or continuous actions is defined with the function $ex : \mathscr{F} \cup \mathscr{A} \rightarrow \mathbb{B}$. Thus, the *state* of a GEIS PN is defined with $s = (m, cond, ex)$.
In GEIS PN, the definition of a *firable* transition t from the state $s = (m, cond, ex)$, noted $t \in firable_{GEIS}(s)$, is:

$$\begin{aligned} t \in firable_{GEIS}(s) \ \Leftrightarrow\ & t \in en(m) \\ & \wedge \big(\forall c \in \mathscr{C} \mid C(t)(c) = 1, cond(c) = 1\big) \\ & \wedge \big(\forall c \in \mathscr{C} \mid C(t)(c) = -1, cond(c) = 0\big). \end{aligned}$$

The explicit representation of the association of the negation of a condition to a transition leads to a quite complex representation of the firable expression. It seams useless at this point, as the negation could be directly integrated in the logic expression itself. But this information could be interesting for conflict resolution (see section 5). Nevertheless, when this information is not essential, we assume that we always have conditions such as $C(t)(c) = 1$.

2.2 Semantics of GEIS PN

Based on these definitions for GEIS PN with the hypothesis of no conflict, we can now formally define their semantics.

Definition 3 (GEIS semantics) The semantics of a GEIS PN without conflict $< P, T, Pre, Pre_t, Pre_i, Post, m_0, C, F, A, clk >$ is the transition system $< S, s_0, \longrightarrow >$ where:

- S is the set of states.

- $s_0 = (m_0, o, o)$ is the initial state where o is the zero function. At the initial state, we have $\downarrow clk = 1$.
- $\longrightarrow \subseteq S \times (Clk \times T^*_\varepsilon) \times S$ is the state transition relation, with T^* the set of finite sets of transitions on T and $T^*_\varepsilon = T^* \cup \varepsilon$. This relation is defined as follows:
  - *Falling transition:* we have $s = (m, cond, ex) \xrightarrow{\downarrow clk, \varepsilon} s' = (m, cond', ex')$ iff $\downarrow clk = 1$ and:
    1. $\forall a \in \mathscr{A}, \exists p \in P \mid A(p)(a) = 1 \wedge m(p) \neq 0 \Rightarrow ex'(a) = 1$, otherwise $ex'(a) = 0$ (update of the execution function for continuous actions)
    2. $\forall c \in \mathscr{C}, cond'(c) = val(c)$ (update of conditions values)

    Let $fired(s') \subseteq T^*$ be the set of transitions that will be fired from s'. At this point we can determine $fired(s')$ depending on $firable_{GEIS}(s')$:
    1. $\forall t \in firable_{GEIS}(s'), t \in fired(s')$ (firable transitions will imperatively be fired)
    2. $\forall t \notin firable_{GEIS}(s'), t \notin fired(s')$ (transitions not firable are not fired)
  - *Rising transition:* we have $s \xrightarrow{\uparrow clk, fired(s)} s'$, with $s = (m, cond, ex)$ and $s' = (m', cond, ex')$, iff $\uparrow clk = 1$ and:
    1. $m' = m - \sum\limits_{t \in fired(s)} Pre(t) + \sum\limits_{t \in fired(s)} Post(t)$ (update of markings)
    2. $\forall f \in \mathscr{F}, \exists t \in fired(s) \mid F(t)(f) = 1 \Rightarrow ex'(f) = 1$, sinon $ex'(f) = 0$ (update of the execution function for impulsive actions)

Example 1 (GEIS PN execution) We illustrate in figure 5 the evolution of the GEIS PN of figure 3. At the initial state e_0, the initial marking is $m_0 = p_0 p_2$ and the values of all the conditions and the continuous or impulsive actions are nil (resp. C_i, A_i and F_i). At the initial state we also have $\uparrow clk = 1$, so the next state transition is a falling transition leading to state e_{00}. The falling transition sets the continuous actions values depending on the marking: p_0 and p_2 are marked thus $ex(A_0) = ex(A_2) = 1$. It also set the conditions values, reading the external value of the signals: for this example, we suppose that both conditions C_0 and C_1 are true. For that reason, transitions t_0 and t_1 are firable in e_{00}, and then will be fired during the next rising transition: we have $\uparrow clk = 1 \wedge fired(e_{00}) = firable_{GEIS}(e_{00}) = \{t_0, t_1\}$. This transition leads to state e_1, and the update of the marking is: $m_1 = p_1 p_3$. There is also an update of the impulsive action values: as the fired transitions are t_0 and t_1, so $ex(F_0) = ex(F_1) = 1$.

Now the semantics of the formalism of GEIS PN without conflict is known, and it precisely respects the implementation constraints of our hardware target. But we also have to translate this formalism into an analyzable one to allow the verification of properties in a formal way.

3 Analysability of GEIS PN

3.1 State of the art

The analysability of the GEIS PN could be considered with three problems to solve: the analysis method must consider the extension for expressiveness (weights, inhibitor and test arcs), the synchronous behavior, and the interpretation influence. The

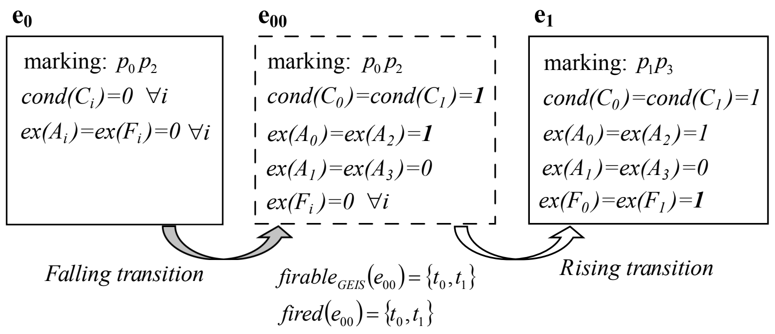


Fig. 5 Evolution of the GEIS PN of figure 3

first point is not a problem as many validation methods and tools allow formal analysis of generalized and extended Petri nets (Berthomieu et al (2004); Gardey et al (2005)).

The second point is more complex, as there is no tool allowing to analyze Petri nets with our synchronous evolution. A classical synchronous evolution could eventually be represented into a discrete semantics, as in Popova (1991); David and Alla (2010). The analysis of discrete time Petri nets has been studied several times, and methods have been proposed for computing the state space and for verifying logical properties (Popova (1991); Magnin et al (2008, 2009); Janowska A. (2013)). There is no deep study on the equivalence of the analysis results between discrete and synchronous behaviors. But we have seen that some specific behaviors are not considered in discrete time, as for example the simultaneous firing of transitions. Furthermore, discrete time TPN analysis methods still have combinatory explosion problem, and there are more optimization methods and efficient tools for the dense time TPN (Gardey et al (2005); Berthomieu et al (2004)).

Another track to follow is the analysis of synchronized PN. The reachability set of a synchronized PN model can easily be computed supposing some simplifying hypotheses: the model is supposed to be safe and/or deterministic (without effective transition conflicts), as in Devillers and Van Begin (2006); Chen et al (2013); Pocci et al (2016). In that case, the reachability set and the language - the set of feasible transition firing sequences - are included into the ones of the underlying ordinary PN (Chen et al (2013)). But we do not accept this simplification. Moreover, synchronized PN make the hypothesis of the instantaneous propagation of the interpretation signals, which could be a problem depending on the calculation time performances of the hardware target. We illustrate this problem on an example figure 1.

Finally, about the third problem, it is well-known that interpretation could not be analyzed on a model as the value of interpretation variables could not be known a priori. To be exhaustive in the property verification process (for the validation of safety properties for example), the only solution is to consider all the possible values of the interpretation and then to over-estimate the real reachable state set. For example if an input variable is a binary one, we could not know a priori if its value will be equal to 0 or 1 at the execution moment. We thus must consider both the values into the analysis process, to verify all the possible real behaviors.

To conclude, there is no analysis method nor analysis tool which are perfectly suitable to our needs. Thus we choose to use the well-known analysis possibilities of the more classical time Petri nets, and we propose a method to take into account the interpretation and the synchronous characteristics into the analysis process thanks to specific time intervals on transitions. We then describe transformation rules from GEIS PN to GET PN (Generalized Extended Time Petri Nets), which is a formalism that could be analyzed with existing analysis tools (Berthomieu et al (2004), Gardey et al (2005)). We also prove that these analysis results are useful and safe: the real implemented behaviors are included into the analyzed behaviors.

3.2 Formal definition and semantics of GET PN

The formal definition of the GET PN formalism is well-known and could be find in the literature (for example Berthomieu et al (2007b); Boyer and Roux (2008); Reynier and Sangnier (2009)). We just give here the basic definitions to fix the notation used in the rest of the article.

Let $\mathbb{I}^+$ be the set of non empty real intervals with non negative integer endpoints. $\forall i \in \mathbb{I}^+$, $\downarrow i$ is its lower bound and $\uparrow i$ its upper bound (could be $+\infty$). To simplify notations, we define that for any $i \in \mathbb{I}^+$, $\theta \in \mathbb{R}^+$, $i - \theta$ corresponds to $[\downarrow i - \theta, \uparrow i - \theta]$.

Definition 4 (GET PN) A generalized extended time Petri net (GET PN) is a tuple $< P, T, Pre, Pre_t, Pre_i, Post, m_0, Is >$ where :

- $< P, T, Pre, Pre_t, Pre_i, Post, m_0 >$ is a GE PN with P the places, T the transitions, m_0 the initial marking and $Pre, Pre_t, Pre_i, Post : T \to P \to \mathbb{N}^+$ the precondition, test, inhibition and postcondition functions.
- $Is : T \to \mathbb{I}^+$ is the static interval function.

Definition 5 (GET marking, state, enabled, newly enabled and firable) The *marking* and *enabled* definitions are the same as for the GEIS PN. A *state* of a GET PN is a pair $s = (m, I)$ in which m is the marking and I is the interval function defined as $I : T \to \mathbb{I}^+$. It associates a time interval with every transition enabled at m.

It is also necessary to define the set of newly enabled transitions: a transition k is said to be *newly enabled* by the firing of the transition t ($t \neq k$) from the marking m, noted $k \in \uparrow en(m,t)$, iff k is enabled by the new marking $m - Pre(t) + Post(t)$ but was not by $m - Pre(t)$. The marking $m - Pre(t)$ is considered because the tokens consumed by the firing of t could temporarily disable a transition before adding the tokens of $Post(t)$. t could also be newly enabled itself if it is still enabled by the new marking. We have:

$$\begin{aligned} k \in \uparrow en(m,t) \Leftrightarrow\ & k \in en(m - Pre(t) + Post(t)) \\ & \wedge\ \big[\big(k = t\big) \vee \big(k \notin en(m - Pre(t))\big)\big] \end{aligned}$$

Finally, in GET PN, a transition is *firable* if it is enabled since enough time to respect its time interval:

$$t \in firable_{GET}(s) \Leftrightarrow t \in en(m) \wedge \downarrow I(t) = 0$$

Definition 6 (GET semantics) The semantics of a GET PN $< P, T, Pre, Pre_t, Pre_i, Post, m_0, Is >$ is the timed transition system $< S, s_0, \longrightarrow >$ where:

- S is the set of states.
- $s_0 = (m_0, I_0)$ is the initial state where m_0 is the initial marking and I_0 is the static interval function Is, restricted to the transitions enabled at m_0.
- $\longrightarrow \subseteq S \times (T \cup \mathbb{R}^+) \times S$ is the state transition relation defined as follows:
  - *Discrete transition:* we have $(m, I) \xrightarrow{t} (m', I')$ iff $t \in T$ and:
    1. $t \in firable_{GET}(m)$
    2. $m' = m - Pre(t) + Post(t)$
    3. $\forall k \in T$, if $k \in \uparrow en(m), I'(k) = I_s(k)$, else $I'(k) = I(k)$.
  - *Continuous transition:* we have $(m, I) \xrightarrow{\theta} (m, I')$ iff $\theta \in \mathbb{R}^+$ and:
    1. $\forall t \in T,\ t \in en(m) \Rightarrow \theta \leq \uparrow I(t)$
    2. $\forall t \in T,\ t \in en(m) \Rightarrow I'(t) = I(t) - \theta$

All the necessary definitions of the GEIS and GET PN formalisms have now been given, we can now specify the transformation rules to represent a GEIS PN model into a GET PN one.

3.3 Transformation rules from GEIS to GET PN

The aim is to translate a GEIS PN $N =< P, T, Pre, Pre_t, Pre_i, Post, m_0, C, F, A, clk >$ into a GET PN $N' =< P', T', Pre', Pre'_t, Pre'_i, Post', m'_0, Is' >$.

- First the PN structure is kept, thus: $P' = P$, $T' = T$, $Pre' = Pre$, $Pre'_t = Pre_t$, $Pre'_i = Pre_i$, $Post' = Post$ and $m'_0 = m_0$.
- Then Is' is defined to reflect the synchronous constraints and the interpretation possibilities. We establish that $1tu$ represents a whole clock period. The synchronous implementation requires that transitions can not be fired in less than $1tu$ thus: $\forall t \in T', \downarrow Is'(t) = 1$. In the other hand, interpretation could indefinitely prevent the firing of a transition, if its associated condition remains false. Thus if a transition in N does not has condition it will be fired in $1tu$, else it can be fired at any time, possibly never:
  - $\forall t \in T, \forall c \in \mathscr{C}, C(t)(c) = 0 \Rightarrow \uparrow Is'(t) = 1$
  - $\forall t \in T, \exists c \in \mathscr{C} \mid C(t)(c) \neq 0 \Rightarrow \uparrow Is'(t) = +\infty$

  These basic transformations are illustrated in figure 6.
- Continuous and impulsive actions of the interpretation are not considered here because they do not directly influence the system execution. They indirectly modify the condition values, then they are ever considered as we have included all the possible conditions values in the GET PN model.

These transformation rules allow to obtain, from a GEIS PN which represents the implemented behavior, a corresponding GET PN. Now we must show that the analysis of this GET PN provides pertinent (guaranteed) and interesting (sufficient, useful) validation results.

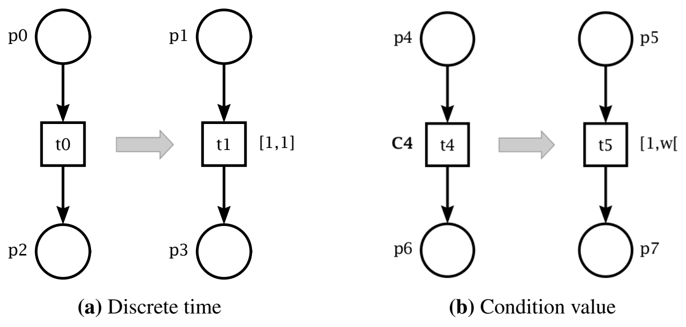


Fig. 6 Transformation rules from GEIS PN to GET PN

4 Analysis results relevance

4.1 Discussion on analysis results

It is of course not possible to precisely describe the behavior of a GEIS PN into a GET PN. First, the interpretation could only be verified considering all the possible values of the interpretation variables, which is a superset of the execution scenarios. Indeed, as the variables could depend on each other, some of the configurations are not realistic. For example, if $a = b - 5$, and if a and b are independently associated to the conditions of two different transitions, the scenario with $a = 12$ and $b = 0$ is not possible in the real world, but will be analyzed. As a result of the interpretation, the analyzed behaviors are therefore a superset of the state space of the implemented real behaviors (see figure 7).

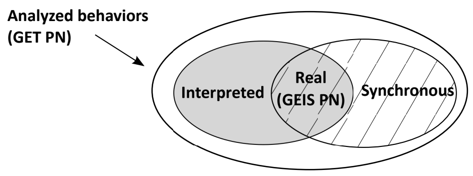
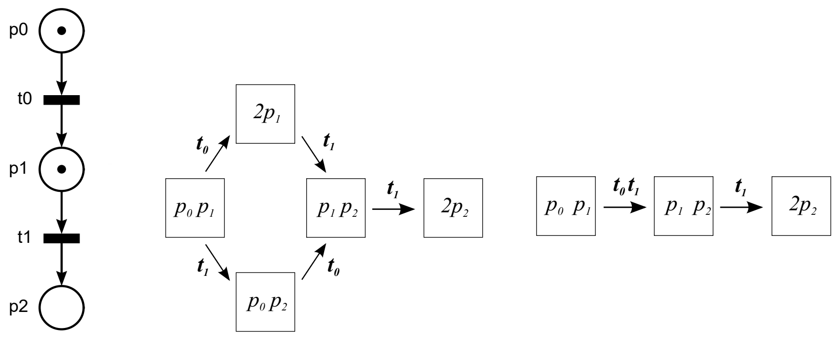


Fig. 7 Inclusion of the real implemented behaviors into the analyzed ones

Second, we could guarantee that the synchronous behavior is included into the asynchronous one, but they are not equivalent. Figure 8 gives in 8a an example of a Petri net, then its state graphs for an asynchronous execution in 8b, and for a synchronous one in 8c[3]. In this example, transitions t_0 and t_1 are concurrent: they are both firable from the initial marking p_0p_1. With synchronous execution, they are simultaneously fired and then the next marking is p_1p_2. But with asynchronous execution, the transitions must be fired one after the other: either t_0 is fired first then t_1, or the

[3] These graphs are not complete: only the relevant marking information have been represented. The continuous GET transitions, as well as the rising GEIS ones, have not been detailed.

contrary. Thus it exists two intermediate markings $2p_1$ and p_0p_2 which do not exist in the synchronous state space.



(a) PN model **(b)** Asynchronous state graph **(c)** Synchronous state graph

Fig. 8 Synchronous vs. asynchronous state space

This example clearly shows that the asynchronous hypothesis produces more states than the synchronous one. The question now is: are all the synchronous behaviors are included in those asynchronous ? We will prove in the following that this inclusion is true, thus proving the relations given in figure 7.

The inclusion of the real behaviors (with the GEIS PN semantics) into the analyzed ones (from the GET PN model) has several consequences on the analysis possibilities, depending on the properties we want to verify (Baier and Katoen (2008)). For example, invariance[4] or safety[5] properties could be guaranteed: if they are satisfied by all the GET PN reachable states, so they are verified by the GEIS PN ones. However, if such a property is not satisfied on the GET PN states, it can still be in the GEIS PN ones, but we can not verify it. On the contrary, the verification results for liveness[6] or reachable[7] properties are irrelevant for the GEIS PN if the answer is ”yes”, as the state satisfying the property could be one of the over-estimated - so unreal - states. However, if these properties are never verified in the GET PN state space, they will not either in the GEIS PN one.

Our goal is to automatically translate the designed model (GEIS PN) into an analyzable model (GET PN) while ensuring the behavior inclusion. But our goal is also to minimize the unrealistic analyzed scenarios, i.e. all parts in figure 7 that are not included in the grey-hatched zone. Indeed, the closer are the analyzed and implemented models, the more interesting are the validation results. The translation method given in section 3.3 respects these two aims. In the domain of formal analysis, model transformations are usual, for example to reduce the verification complexity by the refinement of a complex model with an abstracted one. Refinement relations such

[4] Invariance property: holds for all reachable states
[5] Safety property: something bad never happens
[6] Liveness: a good thing will happen in the future
[7] Reachability: one specific state could be reach

as language inclusion, timed strong or weak simulation or bisimulation are useful to prove trace inclusion, conservation of behaviors and preservation of properties. In the following sections, we prove the inclusion of the GEIS PN behaviors into the GET PN ones thanks to our transformation rules, using the timed simulation equivalence relation.

4.2 Formal definition

Timed simulations (either weak or strong) are powerful relations leading to several interesting conclusions on analysis results. These methods have been used to prove the behaviors conservation of refinements with Timed Automata (Frehse (2006), Fares et al (2013)) and to make a comparison of the expressiveness between several semantics of Time Petri Nets (Bérard et al (2005, 2013)) and Timed Automata (Berthomieu et al (2006); Balaguer et al (2012)). We can also note that the weak timed simulation implies the language inclusion: $S_1 \preceq_W S_2 \Longrightarrow \mathscr{L}(S_1) \subseteq \mathscr{L}(S_2)$ (Baier and Katoen (2008); Bérard et al (2013)). It also has been introduced as a sufficient condition for trace inclusion (Fares et al (2013); Frehse (2006)).

We remind here useful definitions adapted from the above mentioned articles:

Definition 7 (Weak Timed Simulation)
Let $S_1 = (Q_1, q_0^1, \Sigma_\varepsilon, \longrightarrow_1)$ and $S_2 = (Q_2, q_0^2, \Sigma_\varepsilon, \longrightarrow_2)$ be two transition systems over the alphabet Σ and $\preceq$ be a binary relation over $Q_1 \times Q_2$. $\Sigma_\varepsilon = \Sigma \cup \varepsilon$ with ε the silent letter and the empty word. Let $\longrightarrow_{i,\varepsilon}$ be the weak transition relation allowing ε-transition over $\longrightarrow_i$, $i \in \{1,2\}$. The relation $\preceq$ is a weak timed simulation relation of S_1 by S_2 iff:

- $q_0^1 \preceq q_0^2$;
- if $q_1 \xrightarrow{a}_{1,\varepsilon} q_1'$ with $a \in \Sigma_\varepsilon \cup \mathbb{R}_{\geq 0}$ and $q_1 \preceq q_2$ then $\exists q_2 \xrightarrow{a}_{2,\varepsilon} q_2'$ such that $q_1' \preceq q_2'$;

A transition system S_2 weakly simulates S_1 if there is a weak timed simulation relation of S_1 by S_2. We write $S_1 \preceq_W S_2$ in this case.

Definition 8 (GEIS PN run) The synchronous implementation (see section 1.2.2) imposes that a falling transition is necessarily followed by a rising one, thus a GEIS PN run is at least composed of one couple $\{f, r\}$. Therefore, the $\downarrow clk$ and $\uparrow clk$ signals are the representation of the evolution of the time, and the total duration of one falling (f) followed by one rising (r) transitions represents one time unit. We will note $Duration(\{f, r\}) = 1$. The implementation also imposes that no more than $1tu$ can flow by. Indeed, if the conditions associated with all the enabled transitions are false, no transition is firable, but the values of the conditions will be re-evaluated each time unit. In that case, an empty rising transition is fired with no transition fired.

Thus, a run ρ of length n in the GEIS PN semantics is a finite sequence of alternating rising (r) then falling (f) transitions such as:

$$\rho = s_0 \xrightarrow{f_0} s_0' \xrightarrow{r_0} s_1 \ \ldots\ s_{n-1} \xrightarrow{f_{n-1}} s_{n-1}' \xrightarrow{r_{n-1}} s_n$$

We also write $s_0 \xrightarrow{f_0 r_0 \cdots f_{n-1} r_{n-1}} s_n$. We define $Untimed(\rho) \in T^*$ as the concatenation of the transitions fired in the falling transitions of ρ, and $Duration(\rho) \in \mathbb{N}^*$ the representation of the time elapsed during this run. Thus for the run ρ we have:

$$Untimed(\rho) = fired(s'_0) fired(s'_1) \ldots fired(s'_{n-1})$$

and

$$Duration(\rho) = \sum_{i=0}^{n-1} Duration(\{f_i, r_i\}) = n$$

The set of the GEIS PN runs is noted Run_{GEIS}.

Definition 9 (GET PN run) A run ρ of length n in the GET PN semantics is a finite sequence of alternating continuous ($\theta_i \in \mathbb{R}$) and discrete ($t_i \in T$) transitions such as:

$$\rho = s_0 \xrightarrow{\theta_0} s'_0 \xrightarrow{t_0} s_1 \ \ldots \ s_{n-1} \xrightarrow{\theta_{n-1}} s'_{n-1} \xrightarrow{t_{n-1}} s_n$$

We also write $s_0 \xrightarrow{\theta_0 t_0 \cdots \theta_{n-1} t_{n-1}} s_n$. The word $Untimed(\rho) \in T^*$ is obtained by the concatenation $t_0\, t_1 \ldots t_{n-1}$ of the transitions $t_i \in T$ fired during the discrete transitions, and we have $Duration(\rho) = \sum_i |\theta_i|$ with $|\theta_i|$ the duration of the continuous transition θ_i. Unlike GEIS PN runs, a GET PN run can consist of a single transition. The set of GET PN runs is noted Run_{GET}.

Definition 10 (Equivalence of runs) Two runs are equivalent if their respective *Untimed* and *Duration* values are the same: the same transitions are fired during the same amount of time. We formally note: $\forall \rho_s \in Run_{GEIS}, \forall \rho_a \in Run_{GET}$:

$$\rho_s \approx^{run} \rho_a \iff Untimed(\rho_s) = Untimed(\rho_a) \wedge Duration(\rho_s) = Duration(\rho_a)$$

Definition 11 (Equivalence of states) Let N_s the synchronous GEIS PN semantics and N_a the asynchronous GET PN semantics. We consider the equivalence relation $\approx$ over states $s_s = (m_s, cond, ex)$ of N_s and $s_a = (m_a, I)$ of N_a such that: s_s and s_a are two equivalent states ($s_s \approx s_a$) iff :

- $m_a = m_s$: they have the same markings;
- the interval function I of s_a uses the values of the static interval function which were generated from s_s respecting the translation rules given in section 3.3.

4.3 Inclusion proof: *GEIS PN* $\subset$ *GET PN*

To well understand this section, it is very important to distinguish transitions from models than transitions from semantics. For that, we remind that semantics transitions are either named continuous and discrete transitions for the asynchronous semantics, or rising and falling transitions for the synchronous one.

Proposition 1 *For any Petri net $N_s =< P,T,Pre,Pre_t,Pre_i,Post,m_0,C,F,A,clk >$ with the GEIS PN semantics $Sem_{GEIS} = (S_s, s_{s0}, \longrightarrow_s)$, it exists a Petri net $N_a =< P_a,T_a,Pre_a,Pre_{ta},Pre_{ia},Post_a,m_{0a},Is >$ with the GET PN semantics $Sem_{GET} = (S_a, s_{a0}, \longrightarrow_a)$ which weakly timed simulates N_s. Hence the GET PN semantics weakly simulates the GEIS PN semantics: $Sem_{GEIS} \leq_w Sem_{GET}$.*

Proof
We consider that N_a was generated from N_s respecting the transformation rules of section 3.3. By definition of these rules, the structure of the Petri net, as well as the initial marking, are conserved thus $N_a =< P,T,Pre,Pre_t,Pre_i,Post,m_0,Is >$.

The initial states of the two semantics respect the conditions of definition 11 so they are equivalent: $s_{a0} \approx s_{s0}$. Then we can consider that it exists $s_s = (m_s, cond, ex)$ a state of Sem_{GEIS} and $s_a = (m_a, I)$ a state of Sem_{GET} which are equivalent: $s_s \approx s_a$. The weak timed similarity of the semantics will be established if $s_s \approx s_a$ and for any $s_s \xrightarrow{\rho_s}_s \overline{s_s}$ then it exists $s_a \xrightarrow{\rho_a}_a \overline{s_a}$ with $\rho_s \approx_{run} \rho_a$ and $\overline{s_s} \approx \overline{s_a}$. We have to prove this equivalence for all the three possible types of GEIS PN runs: *(1)* no transition is fired, *(2)* only one transition is fired and *(3)* several transitions are fired. For each case, we will prove that it exists in the GET PN semantics an equivalent run, and we will prove that this run leads to a final state equivalent to the final one of the GEIS PN semantics. So, the inclusion $Sem_{GEIS} \subset Sem_{GET}$ will be proved, with the hypothesis established in this section that the Petri nets have no conflict.

(1) No transition fired: Let consider a GEIS PN run ρ_{s0} such as: $s_s \xrightarrow{\rho_{s0}}_s \overline{s_s}$, with $Untimed(\rho_{s0}) = \emptyset$ and $Duration(\rho_{s0}) = n \in \mathbb{N}^*$. We have $\rho_{s0} = s_s \xrightarrow{f_0}_s s'_s \xrightarrow{r_0}_s s_{s1} \dots s_{sn-1} \xrightarrow{f_{n-1}}_s s'_{sn-1} \xrightarrow{r_{n-1}}_s \overline{s_s}$ with $r_i = (\uparrow clk, \emptyset)\ \forall i$. This run can has any duration $n \in \mathbb{N}^*$, indeed it could be possible to infinitely repeat an alternation of f_i and r_i without firing a transition.

Existence of an equivalent run ρ_{a0} : The run ρ_{s0} corresponds to states for which all the enabled transitions are not firable[8]. According to definition 2, this corresponds to transitions associated to conditions whose values prevent the firing. Thus, as all these enabled transitions are associated to a condition in N_s, the translation rules impose that these transitions have the following time interval in N_a: $\forall t_i \in en(m_a), Is(t_i) = [1, +\infty[$ (as in the right of figure 6). As the upper bound is infinite, there is no obligation to fire these transitions: it exists a run where time could elapsed without firing any transition. Hence a continuous transition θ is possible from s_a with any duration, including all the values of $\mathbb{N}^*$: $s_a \xrightarrow{\theta}_a \overline{s_a}$. Thus we have proved, for any GEIS PN run with no transition fired, the existence of an equivalent GET PN run:

$$\forall \rho_{s0} \in Run_{GEIS} \mid Untimed(\rho_{s0}) = \emptyset,\ \exists \rho_{a0} \in Run_{GET} \mid \rho_{a0} \approx^{run} \rho_{s0}$$

[8] We do not detailed the case where no transition is enabled, because it represents a blocking state without discrete possible evolution, but the proof is also relevant in this case.

Equivalence of the final states $\overline{s_s} \approx \overline{s_a}$: According to the GEIS PN semantics, the falling transitions f_i do not change the marking. Furthermore, for all the rising transitions r_i of the run ρ_{s0} no transition is fired, so there is no marking change either. Therefore we have $m_s = m_{si} = m'_{si} = \overline{m_s}\ \forall i$. We also remind that $s_s \approx s_a$ thus we have $m_s = m_a$. Let consider now ρ_{a0} a GET PN run equivalent to ρ_{s0}. As ρ_{a0} consists of only one continuous transition, there is no marking change and $\overline{m_a} = m_a$. By hypothesis, we consider that N_a was generated from N_s respecting the desired transformation rules. So the equivalence of markings is sufficient to establish the equivalence of states. Thus, we have $\overline{s_s} \approx \overline{s_a}$.

(2) Only one transition fired: Let consider a run ρ_{s1} such as: $s_s \xrightarrow{\rho_{s1}}_s \overline{s_s}$, with $t_1 \in T$, $Untimed(\rho_{s1}) = \{t_1\}$ and $Duration(\rho_{s1}) = 1$. We have $\rho_{s1} = s_s \xrightarrow{f}_s s'_s \xrightarrow{r}_s \overline{s_s}$ with $r = (\uparrow clk, t_1)$. This case corresponds to a transition which is enabled and firable in s'_s: $t_1 \in firable_{GEIS}(s'_s)$. Thus t_1 is either without condition or associated to a condition which is true in s_s, as in the example of figure 9a. Furthermore, the firing of only one transition means that (for a PN without conflict) either only this transition is enabled, or the other enabled transitions are not firable because of their conditions: $\forall t_i \in en(m'_s) \mid t_i \neq t_1 \Rightarrow t_i \notin firable_{GEIS}(s'_s)$. An example of such a run is given in figure 10, if the condition C is false so only t_1 is fired.

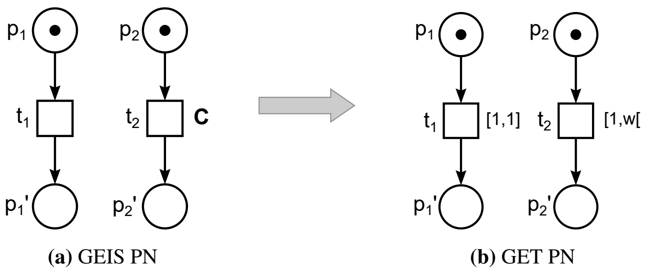


Fig. 9 Example of a GEIS PN and its corresponding GET PN

Existence of an equivalent run ρ_{a1} : Following the transformation rules, the transitions of N_s are transformed in N_a in temporal transitions: $Is(t_1) = [1,1]$, t_1 being without condition in N_s, and $Is(t_2) = [1,+\infty[$, t_2 being associated to a condition in N_s. Thus, from s_a, it is necessary to first execute a continuous transition θ with $|\theta| = 1$: $s_a \xrightarrow{\theta}_a s'_a$. According to the GET PN semantics, we then have $\downarrow I'_a(t) = \downarrow I_a(t) - |\theta| = 0,\ \forall t \in en(m_a)$. Then all the transitions enabled by m_a become firable in s'_a, including t_1: it is now possible to execute a discrete transition corresponding to the firing of t_1. Therefore it exists a run $\rho_{a1} = s_a \xrightarrow{\theta}_a s'_a \xrightarrow{t_1}_a \overline{s_a}$ in the GET PN semantics with $Duration(\rho_{a1}) = 1$ and $Untimed(\rho_{a1}) = \{t_1\}$, which is equivalent to ρ_{s1}. Such a run is given in figure 11.

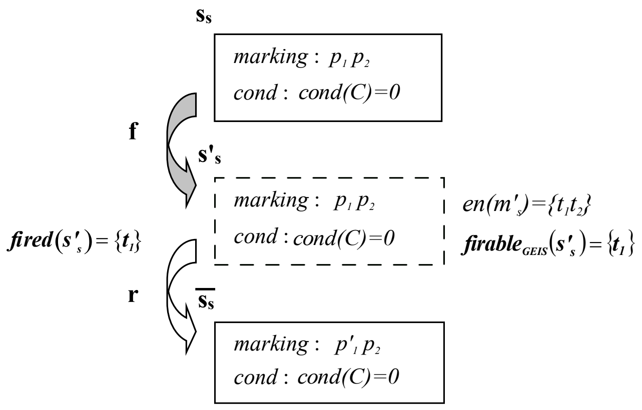


Fig. 10 Run ρ_{s1} of the GEIS PN of figure 9a if C is false

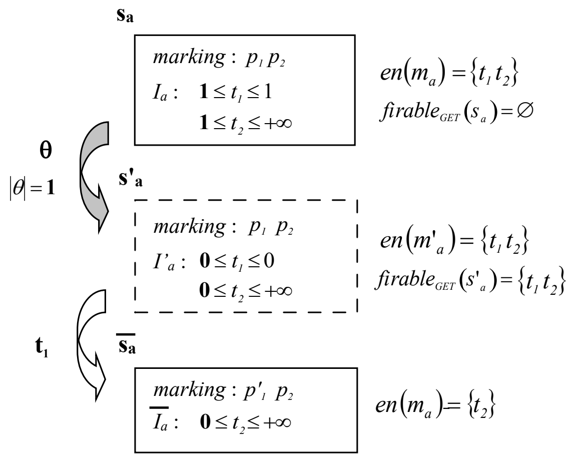


Fig. 11 One run ρ_{a1} of the GET PN of figure 9b

Equivalence of the final states $\overline{s_s} \approx \overline{s_a}$: In the GEIS PN semantics, the firing of the falling transition f does not change the marking: $m'_s = m_s$. The only marking modification of the run ρ_{s1} is done during the rising transition r which corresponds to the firing of t_1. Thus the final marking of ρ_{s1} is: $\overline{m_s} = m_s - Pre(t_1) + Post(t_1)$. According to the GET PN semantics, the continuous transition θ does not modify the marking: $m'_a = m_a$. On the contrary, the discrete transition t_1 set the marking to $\overline{m_a} = m_a - Pre(t_1) + Post(t_1)$. As $m_s = m_a$, the markings $\overline{m_s}$ and $\overline{m_a}$ are the same and we have $\overline{s_s} \approx \overline{s_a}$.

This proof has been done when the fired transition (t_1) has no associated condition. But the same proof can be done with one associated condition, for example in figure 9, firing t_2 if C is true and t_1 not firable (p_1 not marked).

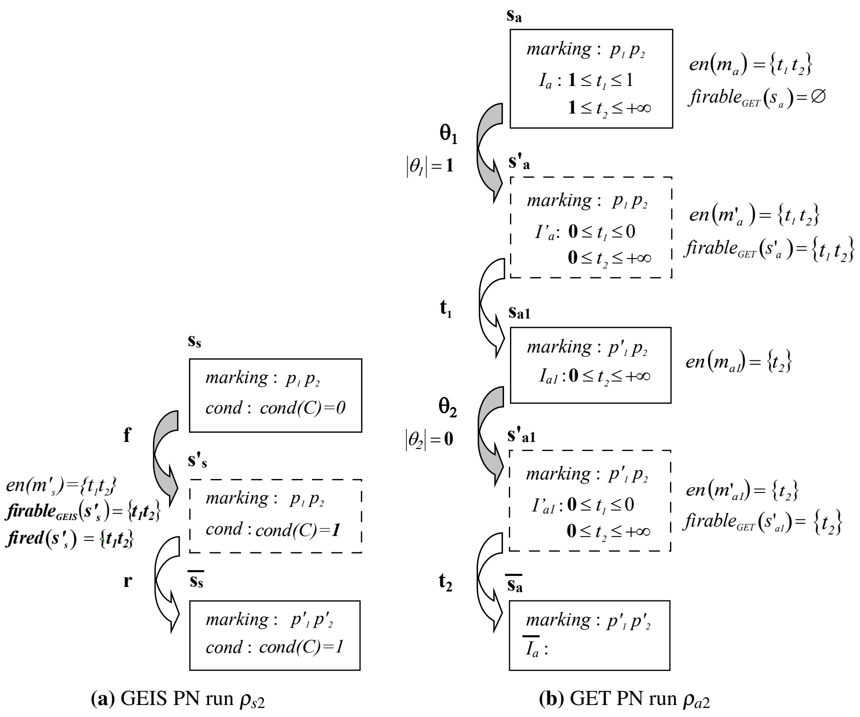
(3) Several transitions fired: This proof will be done by induction: first for the firing of two transitions, then we discuss on the general case of n transitions.

Let consider a run ρ_{s2} such as: $s_s \overset{\rho_{s2}}{\rightsquigarrow}_s \overline{s_s}$, with $Untimed(\rho_{s2}) = \{t_1, t_2\}$ and $Duration(\rho_{s2}) = 1$. We have $\rho_{s2} = s_s \xrightarrow{f}_s s'_s \xrightarrow{r}_s \overline{s_s}$ with $f = (\downarrow clk, \varepsilon)$ and $r = (\uparrow clk, \{t_1, t_2\})$. This corresponds to the case where transitions t_1, t_2 are enabled, and they are the only firable ones: either they do not have condition, or their conditions have true values in s_s, and all others enabled transitions are not firable. This could be the case in figure 9 if C is true: both t_1 and t_2 are firable, and then fired in the same rising transition. This run is shown in figure 12a, and its equivalent GET PN run ρ_{a2} in figure 12b.

Existence of an equivalent run ρ_{a2} : To respect the synchronous behavior, all the transitions of N_s have been transformed in N_a in temporal transitions with $\downarrow Is(t) = 1$. Thus, no transition is immediately firable. So it is necessary to first execute from s_a ($s_a \approx s_s$) a continuous transition θ_1 with $|\theta_1| = 1$: $s_a \xrightarrow{\theta_1}_a s'_a$. The marking is modified but the time intervals is decremented: $\forall t \in en(m'_a), \downarrow I'(t) = 1 - |\theta_1| = 0$. Then all the enabled transitions become firable, included t_1 and t_2. Thus we can execute a discrete state transition corresponding to the firing of the transition t_1: $s'_a \xrightarrow{t_1}_a s_{a1}$. First, according to the GET PN semantics, we know that there is no new firable transition: the time intervals are not modified by a discrete transition, excepted for the transitions newly enabled in s_{a1}. But in that case we would have: $\forall k \in \uparrow en(m_{a1}, t_1), \downarrow I_{a1}(k) = 1$ thus $k \notin firable_{GET}(s_{a1})$. Second, as we supposed that N_s is without conflict, the firing of t_1 does not prevent the firing of the other firable transitions. In particular, t_2 is still firable. As a GET PN run is an alternation of continuous and discrete transitions, it is now necessary to fired a continuous transition, which can be instantaneous ($|\theta_2| = 0$): $s_{a1} \xrightarrow{\theta_2}_a s'_{a1}$. This transition does not change either the label or the firing intervals, so we can fire a discrete transition with t_2: $s'_{a1} \xrightarrow{t_2}_a \overline{s_a}$. We finally have the complete run $\rho_{a2} = s_a \xrightarrow{\theta_1}_a s'_a \xrightarrow{t_1}_a s_{a1} \xrightarrow{\theta_2}_a s'_{a1} \xrightarrow{t_2}_a \overline{s_a}$ with $Duration(\rho_{a2}) = |\theta_1| + |\theta_2| = 1$ and $Untimed(\rho_{a2}) = \{t_1, t_2\}$. Thus we have $\rho_{a2} \approx_{run} \rho_{s2}$.

Equivalence of the final states $\overline{s_s} \approx \overline{s_a}$: In ρ_{s2}, f does not change the marking: $m_s = m'_s$ and r changes the marking respecting the GEIS PN semantics: $\overline{m_s} = m_s - Pre(t_1) - Pre(t_2) + Post(t_1) + Post(t_2)$. In ρ_{a2}, the continuous transitions does not modify the marking: $m'_a = m_a$ and $m'_{a1} = m_{a1}$. For the discrete transition corresponding to the firing of t_1, we have $m_{a1} = m'_a - Pre(t_1) + Post(t_1)$. Likewise, we have $\overline{m_a} = m'_{a1} - Pre(t_2) + Post(t_2)$. Then we finally have $\overline{m_a} = m_a - Pre(t_1) + Post(t_1) - Pre(t_2) + Post(t_2)$, which is equal to $\overline{m_s}$ as $m_a = m_s$. Thus, the final states are equivalent: $\overline{s_a} \approx \overline{s_s}$.

Generalization to n fired transitions: If more than two transitions are firable in the GEIS PN semantics without conflict, they are all fired in the same time unit.



(a) GEIS PN run ρ_{s2} **(b)** GET PN run ρ_{a2}

Fig. 12 Equivalent runs with two simultaneously fired transitions

We just proved that for the simultaneous firing of two transitions in synchronous semantics, the firing of the same two transitions is done during the same time unit in the corresponding asynchronous model. This is done adding an instantaneous continuous transition which allows to fire in the same time unit all the transitions initially firable. Then this proof could easily be extended to *n* fired transitions: in GEIS PN, all the firable transitions are fired in one time unit. In GET PN, only one transition is fired at a time, but we can fire several transitions successively adding instantaneous continuous transitions between them.

All the other possibilities of GEIS PN runs could be proved combining the runs considered above. At the end, it is always possible to find an equivalent GET PN run. And we also prove that equivalent runs lead to equivalent states. This formally proves the proposition 1, meaning **our translation rules guarantee the inclusion of the behavior of the initial GEIS PN into the one of the generated GET PN, if the models are without conflicts**. This proves that for any Petri net with the GEIS PN semantics, it exists a Petri net with the GET PN semantics which weakly timed simulates it. Hence GET PN semantics weakly simulates GEIS PN semantics[9]:

$$\forall N \in Sem_{GEIS},\ \exists N' \in Sem_{GET} \mid N \leq_W N' \Rightarrow Sem_{GEIS} \leq_W Sem_{GET}$$

It is now necessary to verify that this remains true while adding the management of conflicts in GEIS PN models.

5 Conflicts resolution : GEIS PN with priorities

We previously made the hypothesis, as most of the methods dealing with synchronous or synchronized PN in the literature, that the GEIS PN are without conflict. However, we think that it is necessary to remove this hypothesis, as the expression of conflicts are interesting in a modeling point of view, offering more possibilities and simplicity for the designer. So the conflict problem must be considered, and a method of conflict resolution must be provided to manage conflict problems when needed. Conflict resolution can be done with probabilities, alternated firing, or priorities (David and Alla (2010)). In Leroux et al (2014) we propose a solution based on priorities, describing the conflict problematic, how to detect them, and how to manage conflicts in our synchronous VHDL implementation context.

In this article, we focus on the formal part of our conflict management method: we formally define the conflict concept for GEIS PN, then we present a solution to (automatically) handle it, and the extension of our GEIS PN semantics including this solution. This new formalism will be called GEISPr PN (Generalized Extended Interpreted Synchronous *Priority* Petri Nets). Finally, we show that the conflict resolution does not change the verification possibilities by means of GET PN analysis.

5.1 Conflict definitions

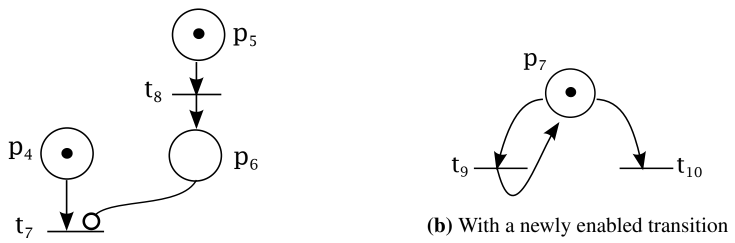
Definition 12 (Structural conflict) A structural conflict in PN traditionally *"corresponds to the existence of a place which has at least two output transitions"* (David and Alla (2010); Chen et al (2013)).

Definition 13 (Effective conflict) But a structural conflict does not necessarily lead to a problematic situation: the real problem is when the concurrent transitions could actually not be fired at the same time. We call it an effective conflict. A simple definition of an effective conflict could be define as in Girault and Valk (2013): *"there is a conflict when two transitions are enabled and the occurrence of one disable the other"*. For classical generalized PN, this definition only implies the consideration of the actual marking: a set of transition T_c sharing an upstream place p are in an effective conflict if they are enabled by a marking m and if the number of tokens in p for m is less than the sum of the weights of the entering arcs of all the transitions of T_c (David and Alla (2010)).

[9] The opposite is not true, as some GET PN models can not be simulated by GEIS PN ones: we do not have equivalence of the semantics.

In our context, these definitions have to be extended considering all the characteristics of our specific GEIS PN formalism:

Generalized The consideration of non-binary Petri nets has already been considered in the previous definition thanks to weight on arcs.
Extended We consider several types of arcs, as inhibitor and test ones, which do not consumed tokens when the associated transition is fired. In that cases, the firing of these transitions does not influence the firing of the others in structural conflict. But dealing with inhibitor arcs, we have a first difference between the asynchronous and the synchronous Petri nets definitions of effective conflicts. For example in figure 13a: in asynchronous semantics, the firing of t_8 prevents the firing of t_7, while in the synchronous semantics they could be fired in the same time as they do not used the same tokens.
Interpreted In our context, because of the interpretation, the set of firable transitions is different than the set of enabled ones, taking into account the condition values. It is thus necessary when we verify if a condition prevents the firing of another, to consider that the transition remains firable (and not only enabled).
Synchronous This is also the case considering the synchronous execution constraint, which imposes that the firing of a transition must be in at least $1tu$. Thereby, even if a transition marks again its input places (thus remaining enabled), all the concurrent downstream transitions of this place do not remain firable because of the $1tu$ minimal time of firing. An example of such a case is given in figure 13b. In that case, in asynchronous semantics t_9 does not prevent the firing of t_{10} at the same time moment, while in synchronous semantics t_{10} will be fired $1tu$ later.



(a) With an inhibitor arc

(b) With a newly enabled transition

Fig. 13 Examples of conflict situations

Thus the definition of effective conflicts must be adapted, not considering the final marking after the firing of t_i, but considering the intermediate marking $m - Pre(t_i)$. For that, we use the concept of newly enabled transitions defined for the GET PN semantics (definition 5).

Definition 14 (Effective conflict in GEIS PN) For a state $s = (m, cond, ex)$ of a GEIS PN model, we define $T_c(t_i, s) \subseteq T$ the set of transitions in effective conflict with $t_i \in \mathit{firable}_{GEIS}(s)$ and with $s \xrightarrow{\mathit{fired}(t_i)} s'$, as the following:

$$t_j \in T_c(t_i, s) \iff \begin{cases} t_j \neq t_i \\ \wedge\, t_j \in \mathit{firable}_{GEIS}(s) \\ \wedge\, \left[t_j \;\in \uparrow en(m, t_i)) \vee\, t_j \notin \mathit{firable}_{GEIS}(s')\right] \end{cases}$$

And we define $T_c(s) \subseteq T$ the set of all the transitions implicated in at least one effective conflict for the state s as:

$$T_c(s) = \{t_i \in T \mid t_i \in \mathit{firable}_{GEIS}(s) \wedge T_c(t_i, s) \neq \emptyset\}$$

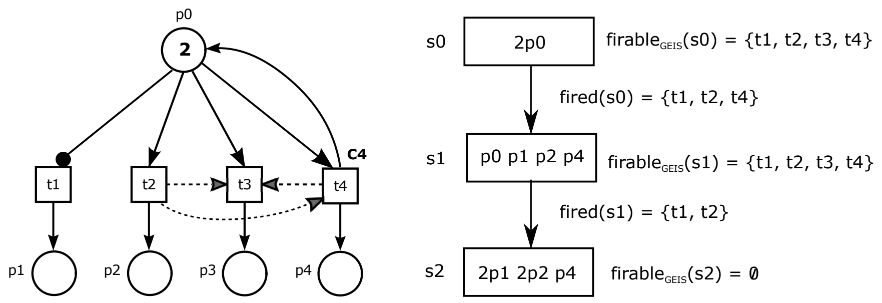
5.2 Method of conflict resolution

Our method of conflict resolution is based on a deterministic resolution of effective conflicts. A static priority is defined between every transition of each structural conflicts. Then, during the execution of the PN, it is checked if the conflict is effective or not in the current state, in order to dynamically determine which transitions must be fired.

Time Petri nets with priority have already been defined in the literature for asynchronous PN (Berthomieu et al (2006, 2007b)). In summary, if two transitions t_1, t_2 are concurrent and if t_1 has priority over t_2, noted $t_1 \succ t_2$[10], so t_1 will be fired before t_2. But, because of the synchronous implementation, the priority principle we need is slightly different from this one. Indeed in our case the priority is used only when the transitions are in an effective conflict. If two transitions are firable but not in an effective conflict, even if there is a priority between them, both must be fired. The principle is just to add, on the falling transition, the consideration of the existence of effective conflicts between transitions. Only in this case the priority are considered, to select all the most priority concurrent transitions which could be fired instantaneously.

Figure 14a gives an example of GEISPr PN with 3 transitions in effective conflict at the initial state (supposing that C_4 is true): $T_c(s_0) = \{t_2, t_3, t_4\}$. t_1 is not included in this set because its firing does not influence the firability of the others. The resolution of this conflict is done with the following priorities: $t_2 \succ t_3$, $t_2 \succ t_4$ and $t_4 \succ t_3$. They are represented in the figure with dotted arcs between transitions. Figure 14b shows the synchronous state graph of this model, only showing the significant states (falling transitions and their intermediate states are not represented). In s_0, if C_4 is true, all the transitions are firable: t_1 will be fired, but it is necessary to use the method of conflict resolution to choose the actually fired transitions of the set $T_c(s_0)$. The marking of s_0 allows to fire the two priority transitions t_2 and t_4, but not t_3. Then $\mathit{fired}(s_0) = \{t_1, t_2, t_4\}$, which leads to a state s_1 from which there is again a conflict resolution, with the same firable transitions but not the same marking in p_0. Then $1tu$ later, $\mathit{fired}(s_1) = \{t_1, t_2\}$ from s_1, leading to the final state s_2. If C_4 is true in s_0, t_3 will never be fired. Now if we consider that C_4 is false in s_0, there is no effective conflict so all the firable transitions $\{t_1, t_2, t_3\}$ are fired from s_0, leading to a final state with $m = p_1 p_2 p_3$.

[10] Graphically, this is represented with an arrow from t_1 to t_2.



(a) GEIS PN model **(b)** Simplified synchronous state graph

Fig. 14 Example of effective conflict resolution

More details on our conflict management method are given in Leroux et al (2014), considering such simple cases but also the asymmetric conflict cases as well as the possibility to have interconnected group of transitions.

The advantage of our method is that it is a dynamic but also deterministic one: the conflict resolution does not suppress a priori the conflicts in a structural way (as for example alternative firing methods), but it is done in a dynamic way only when the conflicts are effective. Our method is a deterministic one, as when an effective conflict occurs, it is always resolved in the same way.

5.3 Definition and semantics of the GEISPr PN

Definition 15 (GEISPr PN definition) The GEIS PN semantics presented section 2.2 must be adapted with the priority management. First we must add the priority concept into the GEIS PN definition: $< P, T, Pre, Pre_t,\ Pre_i, Post, m_0, C, F, A, clk, \succ >$ with $\succ$ the irreflexive, asymmetric and transitive priority relation.

Let $Pr(t)$ be the set of transitions which has priority over $t \in T$:

$$Pr(t) = \{t_i \in T \mid t_i \succ t\}$$

Definition 16 (GEISPr PN semantics) The *enabled* and *firable* functions are the same as for the GEIS PN. The priorities are considered for the calculation of the $fired()$ set of the rising transition, adapting the GEIS PN semantics (definition 3) with the priority management as follow:

- we can determine $fired(s')$ depending on $firable_{GEIS}(s')$:
  1. $\forall t \in firable_{GEIS}(s')$, if $\forall t_i \in Pr(t),\ t_i \notin firable_{GEIS}(s') \Longrightarrow t \in fired(s')$ (firable transitions without firable priority transitions will be fired)
  2. $\forall t \in firable_{GEIS}(s')$, if $[\forall t_i \in Pr(t) \wedge t_i \in firable_{GEIS}(s')]$, $m > Pre(t) + \sum\limits_{t_i} Pre(t_i)$ $\Longrightarrow t \in fired(s')$ (the marking is sufficient to fire t and all the more priority firable transitions in effective conflict with t)
  3. else $t \notin fired(s')$ (in other cases, the transition is not fired)

5.4 Analysis of GEIS with conflict

The existence of priority between two transitions could leads to two different behaviors depending on the existence and the value of conditions on these transitions. For example in figure 4, supposing $t_1 \succ t_0$: if t_1 is firable, i.e. not associated to a condition or if its condition is true, it is fired and t_0 can not be. On the contrary, if t_1 is associated to a false condition, and if t_0 is firable, t_0 is fired. These two different behaviors come from the same model, the only difference is the value of the conditions, which depends on the instantaneous values of the system variables. As we ever said, these values could not be known a priori, therefore **both the behaviors must be analyzed**. That's why we do not used priority in the analyzed model, as the priority in GET PN could prevent the firing of t_0. Thus, we do not need to add anything to the translation rules described section 3.3.

This guaranty that the real behaviors (GEISPr PN) are included into the analyzed ones (GET PN). The proof of the behaviors inclusion (not detailed here) is just an adaptation of the proof of section 4.3, taking into account runs with effective conflicts. As informally explained in the preceding paragraph, for all the possible runs in an effective conflict situation of GEISPr PN, it exists an equivalent run in the GET PN (the same transitions are fired in the same duration), which keeps the proof of the inclusion.

We now have to add the last element to our formalism: the management of quantitative time, to finally have the complete formalism dealing with all the constraints of our context.

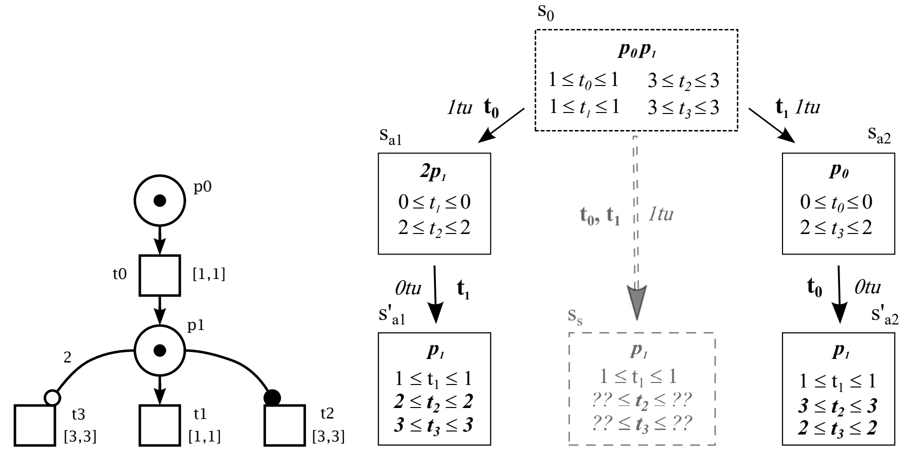
6 Temporal extension: GEISPr Time PN

Adding the time management into the GEISPr PN formalism leads to the formalism we name GEISPrT PN. This is the final formalism, which includes all the constraints implied by our implementation context.

6.1 Problematic of time

6.1.1 Resetting of counters

In a synchronous semantics, to deal with concurrency, it is necessary to finely manage the resetting of the time counter caused by the transient states. An example is given in figure 15a, in which t_0 and t_1 are concurrent, i.e. simultaneously firable but not in effective conflict. In synchronous execution they will be simultaneously fired and then the marking of place p_1 remains equal to 1 token, even if a transient nil marking exists. Now the problem resides in the consideration of the inhibitor and the test arcs of the transitions t_2 and t_3: is it necessary to reset the time counters of these transitions, and how[11]? This problem is closed to the firing semantics problem for TPN developed in Bérard et al (2005), in which the authors proved that the three proposed firing semantics are equivalent for upper-closed intervals. But in synchronous semantics, it is more intricated.



(a) Petri net with time **(b)** Its asynchronous / synchronous execution

Fig. 15 Management of the time counters resetting

For the analysis purpose, it is necessary that GEISPrT PN behaviors are included into the GET PN ones. Yet, for analysis, the execution will be asynchronous (states in dark, in figure 15b): either t_0 is fired first, then t_1 (run ρ_{a1}, at the left of the graph), either the contrary (run ρ_{a2}, at the right of the graph). When t_0 is fired first, the marking of p_1 becomes equal to 2, disabling t_3. Then t_1 is fired making t_3 enabled again, but with its firing interval which has been reset. In the case t_1 is fired first, the new marking is only 1 token in p_0, disabling t_2. Then the firing of t_0 makes t_2 enabled again, but with its firing interval which has been reset. Then the two runs ρ_{a1} and ρ_{a2}, even if they lead to the same marking p_1, do not lead to the same state because of the firing interval function: ρ_{a1} leads to state s'_{a1} with $I'_{a1}(t_2) = [2,2]$ and $I'_{a1}(t_3) = [3,3]$, whereas ρ_{a2} leads to state s'_{a2} with $I'_{a2}(t_2) = [3,3]$ and $I'_{a2}(t_3) = [2,2]$.

In asynchronous execution, it is not possible to observe the resetting of both counters, nor any resetting at all. If we want to guarantee the inclusion of the semantics, the resulting state of the synchronous execution (simultaneous firing of t_2 and t_3) must correspond to one of the asynchronous resulting states. Thus it is necessary to define the management of the counter resetting in the GEISPrT PN semantics following one of the two asynchronous situations (either the resetting of the test arcs, or the inhibitor arcs). The choice we made is the following: **in synchronous execution, the transient marking is always calculated considering first the withdrawal of the tokens**. If a transition is disable by this transient marking, its time counter is reset. In

[11] The same question could be asked when a transition is newly enabled.

the case illustrated figure 15a, the simultaneous firing of $\{t_0,t_1\}$ leads to the transient marking $m_0 - Pre(t_0) - Pre(t_1) = 0$, which disable t_2 as in the asynchronous run ρ_{a2}. This also allows to deal with the resetting of a transition which re-enables itself.

In the implementation point of view, the resetting of the counters will be done in two steps. First on the rising edge, the calculation of the new marking is done, allowing to determinate the newly enabled transitions, and then the ones which must be reset. We manage this by means of a specific *reset* signal associated to each transition. Second, on the falling edge, the time counters values of the firing interval are calculated, including the ones that must be reset.

6.1.2 Firing semantics

Two semantics are conventionally used with regard to the firing of transitions in TPN: strong semantics and weak semantics (Boyer and Roux (2008); Reynier and Sangnier (2009)). Strong semantics defines that if a transition is enabled it is necessarily fired before, or at worst when, the upper bound of its firing interval is reached. Whereas in the weak semantics, no transition is forced to be fired: if its time interval is exceeded, the transition is "disabled" and will become firable again when it will be enabled again. The strong semantics is the most used to describe real-time systems as it allows to model the urgency of events. This is also the semantics used in the TPN analysis tools, and the one used in the GET PN formalism we have presented section 3.2.

Therefore we use the strong semantics, but with a more strict firing rule: **a firable temporal transition is immediately and imperatively fired**. This is consistent with the behavior of the real system, and with our need for determinism. But we have to slightly modify this semantics to take into account the blocking situations.

6.1.3 Blocking situation

The strong semantics forces the firing of a transition when its upper bound is reached. Yet, in case of the association of a condition and a firing interval on the same transition, it could be possible that the upper bound is reached whereas the condition always remains false[12], preventing the firing and provoking a blocking situation. An example is given in figure 16: if c_2 remains `false` during all the duration of $[a,b]$, the transition t_2 must, but can not, be fired when b is reached.

To manage this problem, we adapt the strong semantics introducing the weak semantics concept: in this specific case of blocking, **the transition time counter can overlap its upper bound but the transition could not be fired anymore until being newly enabled**. In our example of figure 16, if t_2 is blocked, the firing of t_1 could empty p_1, disabling and then unblocking t_2. If such an alternative path does not exist, then the blocked transition remains indefinitely blocked.

[12] The time counter is incremented as soon as the transition is enable, independently of the condition value.

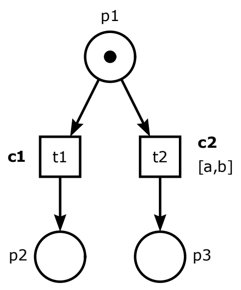


Fig. 16 Example of a potential blocking situation

6.2 Formal definitions and semantics of GEISPrT PN

The basic definitions for the GEISPrT PN semantics are based on the ones given in the sections 2.1 and 5.3. We give here the additions or the changes we have introduce for the quantitative time management in our synchronous semantics.

6.2.1 Definitions for GEISPrT PN

Definition 17 (GEISPrT PN) First we must add the firing interval concept into the GEISPr PN definition (definition 15), leading to the GEISPrT PN $< P, T, Pre, Pre_t, Pre_i, Post, m_0, C, F, A, clk, I_s, \succ>$ with $I_s : T \to I^+$ the static firing interval function. We suppose that $I_s(t) = \emptyset$ for untimed transitions (transitions not concerned by time interval[13]).

Definition 18 (Reset, State, Firable in GEISPrT PN) Let $reset : T \to \mathbb{B}$ be the resetting function, that must be considered into the states. The value of the counter of firing intervals must also be considered, with $I : T \to \mathbb{I}^+$ the dynamical firing function which associates a time interval to every transition enabled at m. We chose to represent the blocking of a transition through its firing interval: $I(t) = \oslash$ means that t is blocked[14]. Thus the *state* of a GEISPrT PN is defined with $s = (m, cond, ex, I, reset)$.

The *firable* definition must integrate the firing interval management: we add to the definition given in definition 2 that a transition t is firable, in addition to being enabled and having its associated condition `true`, iff the lower bound of its firing interval is reached: $0 \in I(t)$. We note $t \in firable_{GEISPrT}(s)$.

Definition 19 (Newly enabled function) We have seen that the notion of newly enabling is important in GEISPrT PN both for the counters resetting and the management of the blocked transitions. A transition $k \in T$ is *newly enabled* by the firing of a set of transitions $T_F \subset T$ from the marking m, noted $k \in \uparrow en(m, T_F)$, iff k is enabled

[13] We could have defined a default interval $I_s(t) = [1,1]$, as in both cases this transition will be fired in $1tu$, but in the implementation point of view there is a difference because untimed transitions are more efficiently implemented regarding energy consumption and silicon footprint.

[14] Note that $\oslash$ is different than the empty interval $[0,0]$ and than no interval at all $I_s(t) = \emptyset$.

by the new marking m', and either $k \in T_F$ or $k \notin T_F$ and k was disabled by the transient marking $m - \sum_{t \in T_F} Pre(t)$.

Formally we have, with $m' = m - \sum_{t \in T_F} Pre(t) + \sum_{t \in T_F} Post(t)$:

$$k \in \uparrow en(m, T_F)$$
$$\Leftrightarrow$$
$$\big[k \in en(m')\big] \wedge \Big[\,(k \in T_F) \vee \big(k \notin T_F \wedge k \notin en(m - \sum_{t \in T_F} Pre(t))\big) \,\Big]$$

6.2.2 Semantics of GEISPrT PN

We can now formally define the semantics of our complete formalism, including all the desired characteristics : the Generalized Extended Interpreted Synchronous Time Petri nets Priority (GEISPrT PN).

Definition 20 (Semantics of GEISPrT PN) The semantics of a GEISPrT PN $\mathcal{N} = < P, T, Pre, Pre_t, Pre_i, Post, m_0, C, F, A, clock, I_s, \succ >$ is the timed transition system $< S, s_0, \longrightarrow >$ where:

- S is the set of states $(m, cond, ex, I, reset)$ of $\mathcal{N}$.
- $s_0 = (m_0, o, o, I_0, o)$ is the initial state where o is the zero function and I_0 is the restriction of I_S to the transitions enabled by m_0.
- $\longrightarrow \subseteq S \times Clk \times S$ is the state transition relation, noted $s \longrightarrow s'$, defined as follows: Let $fired(s) \subseteq T$ the set of transitions fired from state s.
  - *Falling transition:* we have $s = (m, cond, ex, I, reset) \overset{\downarrow clk, \varepsilon}{\longrightarrow} s' = (m, cond', ex', I', reset)$ iff $\downarrow clk = 1$ and:
    1. Updating the execution function for continuous actions is the same as for GEIS PN.
    2. Updating condition values is the same as for GEIS PN.
    3. $\forall t \in en(m), \big(reset(t) = 0 \wedge I(t) \neq \oslash\big) \Rightarrow I'(t) = I(t) - 1$ (normal evolution)
    4. $\forall t \in en(m), reset(t) = 1 \Rightarrow I'(t) = Is(t) - 1$ (reset of the firing interval[15] while unblocking the transition if it was blocked)
    5. $\forall t \in T, \big(reset(t) = 0 \wedge I(t) = \oslash \Rightarrow I'(t) = I(t)$ (a blocked transition remains blocked if it was not newly enabled)
  - The calculation of $fired(s')$ is the same as for the GEISPr PN formalism (definition 16), but using $firable_{GEISPrT}(s')$ (definition 18) instead of $firable_{GEIS}(s')$.
  - *Rising transition:* we have $s \xrightarrow{\uparrow clk, fired(s)} s'$, with $s = (m, cond, ex, I, reset)$ and $s' = (m', cond, ex', I', reset')$, iff $\uparrow clk = 1$ and:
    1. The marking is updated as for GEIS PN.
    2. As well as the execution function for impulsive actions.
    3. $\forall t \in T, t \in \uparrow en(m, fired(s)) \Rightarrow reset'(t) = 1$ else $reset'(t) = 0$ (all the newly enabled transitions will be reset in the next falling transition)

[15] This reset is the consequence of the transitions fired on the previous rising transition, done in the previous time unit. Then we reset *and* subtract 1 to the static interval to be consistent.

4. $\forall t \in T, (t \notin fired(s) \wedge \uparrow I(t) = 0) \Rightarrow I'(t) = \oslash$, else $I'(t) = I(t)$ (blocking of transitions not fired when their upper bound are reached)

6.3 Analysis of GEISPrT PN

6.3.1 Transformation rules

The three characteristics we add into our semantics for the time management (resetting the counters, the "as soon as possible" firing semantics and the blocking semantics) must of course be analyzed. We therefore have to study their inclusion into the GET PN behaviors, and modify the transformation rules presented section 3.3 to translate a GEISPrT PN $N =< P, T, Pre, Pre_t, Pre_i, Post, m_0, C, F, A, clk, I_s, \succ>$ into an analyzable GET PN $N' =< P', T', Pre', Pre'_t, Pre'_i, Post', m'_0, Is' >$ which respects the execution constraints.

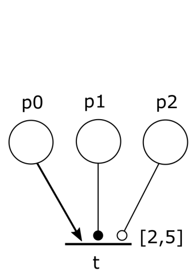
As illustrated section 6.1.1, the resetting of the counters is not a problem, as the chosen semantics leads to an already existing GET behavior. Finally, the blocking situation can not be entirely translated with firing intervals in the GET PN formalism. The initial firing interval must be kept, as the condition could become true anytime during it. But it is also necessary to explicitly add a specific structure (described in the next section) to represent the blocking of the transition.

- $\forall t \in T, \left(\forall c \in \mathscr{C}, C(t)(c) = 0 \wedge I_s(t) = \emptyset\right) \Rightarrow \downarrow I'_s(t) = \uparrow I'_s(t) = 1$ (transitions without condition nor firing interval are fired in $1tu$)
- $\forall t \in T, \left(\exists c \in \mathscr{C} \mid C(t)(c) \neq 0 \wedge I_s(t) = \emptyset\right) \Rightarrow \downarrow Is'(t) = 1, Is'(t) = +\infty$ (transitions with a condition but no firing interval can be fired at any time)
- $\forall t \in T, \left(\forall c \in \mathscr{C}, C(t)(c) = 0 \wedge I_s(t) \neq \emptyset\right) \Rightarrow \downarrow I'_s(t) = \uparrow I'_s(t) = \downarrow I_s(t)$ (transitions with a firing interval but no condition is fired at the lower bound)
- $\forall t \in T, \left(\exists c \in \mathscr{C} \mid C(t)(c) \neq 0 \wedge I_s(t) \neq \emptyset\right) \Rightarrow \downarrow Is'(t) = \downarrow Is(t), \uparrow Is'(t) = \uparrow Is(t)$ (transitions with condition and firing interval can be blocked: the firing interval is kept and a blocking structure will be added)

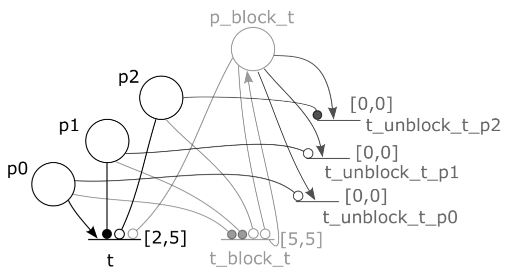
6.3.2 Blocking structure

A specific blocking semantics does not exist in the analysis tools of classical Petri nets. We then have to define a specific structure to explicitly represent the blocking management into the analyzable model. An example is given in figure 17 for the blocking management of the t transition, with the initial GEIS PN model in figure 17a and its transformation in analyzable GET PN in figure 17b.

First we add elements allowing the modeling of the blocking (in light grey in figure 17b) of the targeted transition t: (i) a blocking transition *t_block_t* which has a time interval $[b, b]$ and the same enabling conditions than t: we copy the same input arcs than the ones of t, only switching normal arcs into test ones; and (ii) a place *p_block_t* which prevents the firing of t (after the firing of the blocking transition) thanks to an inhibitor arc. Note that the firing of *t_block_t* does not change the marking of the input places of t which is $p_0 p_1 p_2$. Then we add the unblocking mechanism



(a) Initial model



(b) Analyzable model

Fig. 17 Blocking structure for analysis

(in dark grey, to the right of figure 17b): (iii) for each input place *pi* of *t*, we add a specific unblocking transition *t_unblock_pi* with inverse enabling conditions; if one of the input places *pi* does not satisfy the enabling conditions of *t* anymore (if its marking changes), then *t* will be immediately unblock, firing the related unblocking transitions thus consuming the token of *p_block_t*. For place and simplicity reasons, we do not give the formal definition of this transformation, but it can be found in Leroux (2014).

6.3.3 Behaviors inclusion

The addition of time intervals does not change the proof of the behaviors inclusion in the general case, except in the case of the blocking semantics. Indeed, the added blocking structure modifies the structure of the Petri nets, adding places and transitions. To prove the inclusion of the behaviors, we must show that the inclusion still exists despite of the blocking structure. Thus we have to modify the equivalence relations defined section 4.2.

Let $\mathscr{N}$ be a GEISPrT PN, and $\mathscr{N}'$ be the GET PN generated with the transformation rules defined in 6.3.1. Let P_{block} and T_{block} respectively be the sets of places and transitions added to the GET PN model for the modeling of the blocking of transitions. We have $T' = T \cup T_{block}$ and $P' = P \cup P_{block}$.

Definition 21 (Equivalence of runs) For a run in a GET PN, let $Untimed^{NB}$ be the restriction of $Untimed$ on the initial "non blocking" set of transitions, i.e. on T.

A GET run and a GEISPrT PN run are equivalent if their $Duration$ values are the same, and if their $Untimed$ values contains the same non-blocking transitions of T in the same order of execution. The only difference is the possible interleaving of transitions of T_{Block} in the GEISPrT PN run. We formally note[16]: $\forall \rho_s \in Run_{GEIS}$, $\forall \rho_a \in Run_{GET}$:

$$\begin{aligned}\rho_s \approx^{run} \rho_a \iff & Duration(\rho_s) = Duration(\rho_a) \\ & \wedge Untimed(\rho_s) = Untimed^{NB}(\rho_a)\end{aligned}$$

[16] The definition of GEIS runs is still valid for GEISPrT PN runs.

Definition 22 (Equivalence of states) For a marking m in a GET PN, let m^{NB} the restriction of m on the "non blocking" set of places, i.e. on P.

Let N_s the synchronous GEISPrT PN semantics and N_a the asynchronous GET PN semantics. We consider the equivalence relation $\approx$ over states as: the states s_s of N_s and s_a of N_a are equivalent if the restriction of their markings to the non-blocking places are the same:

$$s_s \approx s_a \Leftrightarrow (m_s = m_a^{NB})$$

Proof

We do not give here the details of the proof, we just give the outlines.

The blocking semantics add 3 possible behaviors in case of a potentially blockable transition: (1) either the transition is fired inside its firing interval, and then not blocked; (2) or its upper bound is reached but the transition is still not firable, then it is blocked, and there is no unblocking possibility: the transition will be blocked forever; (3) or it is blocked but will be unblocked later. To complete the proof of the whole representation of time in GEISPrT PN (and not only the blocking structure), we must add a fourth behavior which corresponds to the "as soon as possible" firing semantics of a temporal transition without associated condition.

The first and fourth behaviors are quite easy to prove, as the continuous consideration of time in GET PN semantics naturally include all the discrete firing instants (Popova (1991); Magnin et al (2009)). For the blocking situations, it is also quite easy to show that the equivalence of the states and of the runs are kept, as (i) the marking of none of the places of P is affected by the firing of the transitions of T_{block}; (ii) the enabling function is only affected by the new blocking place; (iii) the blocking and unblocking transitions in the GET PN are enabled at the same time than the blocking conditions in the GEISPrT PN semantics, leading to the same action.

7 Conclusion

This paper presents a solution to formally design digital architecture (e.g. controllers) for synchronously-executed embedded critical systems. The typical targeted applications are active implantable medical devices (AIMD) implanted into human body to perform FES (Functional Electrical Stimulation). In our context, the digital part of the controller of the AIMD is implemented into a FPGA execution target in a synchronous way. But these works could be used for the design of any critical systems that need high level of dependability guarantee.

The goal of our work is to finely consider at the formal level all the executive constraints imposed by the hardware target and the implementation strategy. The very critical aspect of our system imposes that these constraints must be integrated from the very beginning of the design process, within the analysis step, to guarantee that the verification results remain consistent in all possible cases.

This article first presents our context, focusing on the two scientific key points: expressing and analyzing the interpretation (i.e. the link between the system and the real world) and the synchronous implementation. We present in detail all the execution constraints implied by our context, and explain how these constraints must be considered into the modeling and the analysis purposes. Then sections 2, 3 and 4 present the bases of our work: the GEIS PN (Generalized Extended Interpreted Synchronous Petri Nets) semantics, which is our basic execution semantics; then how to express it into a more classical PN (the GET PN, better known as TPN) semantics; and finally the proof of the inclusion of the GEIS PN semantics into the GET PN one. This inclusion offers the possibility to obtain consistent verification results on our system using existing Petri nets analysis tools.

In sections 5 and 6, in an iterative way, we make our approach more complex to finally integrate all our execution constraints in the final GEISPrT PN semantics, and prove that it is still included into the GET one.

The work presented in this paper has two main results: an operational one, and a theoretical one. First, thanks to these works, we can use the HILECOP methodology to design more critical controllers, with harder safety or real time constraints. We will be sure that the model we designed respects the execution constraints of our hardware target and implementation strategy. HILECOP is currently used to the design of concrete and industrial systems in the medical domain, mainly through the NEURINNOV start-up. Second, this work broadens the scope of expressivity and analyzability of Petri nets extensions. Until then, none managed in the same formalism, both for modeling and analysis, all the characteristics we have considered (weights on arcs, specific test and inhibitor arcs, interpretation and time intervals, including the management of effective conflicts and the blocking of transitions).

Perspectives As a result of this work, we want to go deeper into the analysis problematic. For now, we propose to translate our model in the well-known (GE)TPN formalism, thus using an over-estimated state space of the real one. This implies that many of the analyzed behaviors are unrealistic ones, as shown figure 7. This is a limitation in the properties we want to verify. Liveness and reachability properties are often very useful properties, which could not be guaranteed for now in our analysis solution. It could be interesting to work on the semantics of a specific state space graph for the GEISPrT PN. We thus could adapt existing analysis algorithms to this graph to obtain more pertinent verification results.

Another element to work on is to go further than the modeling step, including the execution language semantics itself. For now, the execution constraints are expressed in the Petri nets semantics GEISPrT PN. Then the model is automatically translated into a specific implementation in VHDL language, assuming that the execution constraints have been well represented in the model, and that they are well preserved by the translation in VHDL. To increase the reliability of the whole methodology, a PhD thesis is ongoing to formally prove that this translation preserves the behaviors and the properties. The aim is so to prove the equivalence between the GEISPrT PN specification and the VHDL code, using a COQ based approach.

8 Glossary and acronyms

AIMD : **Active Implantable Medical Device**
Devices implanted into human body to perform FES solutions.

FES : **Functional Electrical Stimulation**
Application of small electrical charges to nerves and muscles to artificially generate movements. These solutions are useful in an increasing number of applications, including pacemakers, deep brain stimulation, pain control and hearing restoration.

FPGA : **Field-Programmable Gate Arrays**
Specific execution target with real parallelism.

GET PN : **Generalized Extended Time Petri Nets**
Petri nets extension with weight on arcs, test and inhibitor arcs, and time intervals on transitions. This formalism is rarely explicitly named, and is often included in the more simple name TPN.

GEIS PN : **Generalized Extended Interpreted Synchronous Petri Nets**
Petri nets extension with weight on arcs, test and inhibitor arcs, with explicit interpretation elements, and synchronously executed.

IPN : **Interpreted Petri Nets**
Petri nets extension in which some elements of the model are linked with the real world through interpretation elements (events and actions) .

PN : **Petri Nets**
Classical formalism for modeling of discrete event systems.

TPN : **Time Petri Nets**
Petri nets extension with time intervals on transitions.

VHDL : **Generalized Extended Interpreted Synchronous Petri Nets**
Programming language adapted to FPGA execution targets.

HILECOP : **High-Level Hardware Component Programming**
Methodology designed in the INRIA team DEMAR/CAMIN to assist in the development of safe AIMD.

References

Andreu D, Souquet G, Gil T (2008) Petri net based rapid prototyping of digital complex system. In: Proc. of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Montpellier, France

Andreu D, Guiraud D, Souquet G (2009) A distributed architecture for activating the peripheral nervous system. Journal of Neural Engineering 6(2):18

Baier C, Katoen JP (2008) Principles of Model Checking. MIT Press, Cambridge, MA, USA

Balaguer S, Chatain Th, Haar S (2012) A concurrency-preserving translation from time Petri nets to networks of timed automata. Formal Methods in System Design 40(3):330–355

Basile F, Faraut G, Ferrara L, Lesage J (2020) An optimization-based approach to discover the unobservable behavior of a discrete-event system through interpreted petri nets. IEEE Transactions on Automation Science and Engineering 17(2):784–798

Bérard B, Cassez F, Haddad S, Lime D, Roux OH (2005) Comparison of different semantics for time Petri nets. In: Proc. of the 3rd Int. Symposium on Automated Technology for Verification and Analysis (ATVA), Taipei, Taiwan

Bérard B, Cassez F, Haddad S, Lime D, Roux O (2013) The expressive power of time petri nets. Theoretical Computer Science 474:1–20

Berthomieu B, Ribet PO, Vernadat F (2004) The tool tina – construction of abstract state spaces for petri nets and time petri nets. International Journal of Production Research 42(14):2741–2756

Berthomieu B, Peres F, Vernadat F (2006) Bridging the gap between timed automata and bounded time petri nets. In: Proc. of 4th Formal Modeling and Analysis of Timed Systems (FORMATS), Paris, France

Berthomieu B, Lime D, Roux OH, Vernadat F (2007a) Reachability problems and abstract state spaces for time Petri nets with stopwatches. Discrete Event Dynamic Systems 17(2):133–158

Berthomieu B, Peres F, Vernadat F (2007b) Model checking bounded prioritized time petri nets. In: Proc. of the Int. Symposium of Automated Technology for Verification and Analysis (ATVA), Tokyo, Japan

Boyer M, Roux OH (2008) On the compared expressiveness of arc, place and transition time petri nets. Fundamenta Informaticae 88(3):225–249

Busi N (2002) Analysis issues in petri nets with inhibitor arcs. Theoretical Computer Science 275:127–177

Chen Xl, Li Zw, Al-Ahmari AM, El-Tamimi AM, Nasr ESA (2013) Confusion diagnosis and control of discrete event systems using synchronized petri nets. Asian Journal of Control 15(6):1736–1751

David R, Alla H (2010) Discrete, Continuous, and Hybrid Petri Nets. Springer-Verlag

Devillers R, Van Begin L (2006) Boundedness undecidability for synchronized nets. Information Processing Letters 99(5):208–214

Elidrissi HL, Nait-Sidi-Moh A, Tajera A (2020) Modular design for an urban signalized intersections network using synchronized timed petri nets and responsive control. In: Proc. of the 11th International Conference on Ambient Systems, Networks and Technologies (ANT), Warsaw, Poland

Fares E, Bodeveix JP, Filali-Amine M, Garnacho M (2013) An automatic technique for checking the simulation of timed systems. In: Proc. of the 11th Int. Symposium of Automated Technology for Verification and Analysis (ATVA), Hanoi, Vietnam

Frehse G (2006) On timed simulation relations for hybrid systems and compositionality. In: Proc. of the 4th Int. Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS), Paris, France

Frey G (2002) Design and formal analysis of petri net based logic control algorithms. PhD thesis, University of Kaiserslautern

Gardey G, Lime D, Magnin M, Roux OH (2005) Roméo: A tool for analyzing time petri nets. In: Proc. of the 17th Int. Conf. on Computer Aided Verification (CAV), Edinburgh, Scotland, UK

Girault C, Valk R (2013) Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications. Springer Science & Business Media

Grobelna I, Adamski M (2011) Model checking of control interpreted petri nets. In: Proc. of the 18th Int. Conf. Mixed Design of Integrated Circuits and Systems (MIXDES), Gliwice, Poland

Hilal R, Ladet P (1993) Synchronous petri nets: formalisation and interpretation. In: Proc. of the Int. Conf. on Systems, Man and Cybernetics (SMC), Le Touquet, France

Huang Y, Weng Y, Zhou M (2018) Design of regulatory traffic light control systems with synchronized timed petri nets. Asian Journal of Control 20:174– 185

Ivanov S, Pelz E, Verlan S (2014) Small universal non-deterministic petri nets with inhibitor arcs. In: Proc. of the 16th Int. Workshop Descriptional Complexity of Formal Systems (DCFS), Turku, Finland

Janowska A PAZA Penczek W (2013) Using integer time steps for checking branching time properties of time petri nets. In: Transactions on Petri Nets and Other Models of Concurrency VIII, Lecture Notes in Computer Science, vol 8100, Springer, Berlin, Heidelberg, pp 89–105

Knapik M, Penczek W, Szreter M, Pólrola A (2010) Bounded parametric verification for distributed time petri nets with discrete-time semantics. Fundamenta Informaticae 101(1-2):9–27

Leroux H (2014) Méthodologie de conception d'architectures numériques complexes : du formalisme à l'implémentation en passant par l'analyse, préservation de la conformité. application aux neuroprothèses. PhD thesis, Université de Montpellier 2

Leroux H, Godary-Dejean K, Andreu D (2013) Complex Digital System Design: a methodology and its application to medical implants. In: Proc. of the Int. Workshop on Formal Methods for Industrial Critical Systems (FMICS), Madrid, Spain

Leroux H, Godary-Dejean K, Coppey G, Andreu D (2014) Automatic handling of conflicts in synchronous interpreted time petri nets implementation. In: Proc. of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Tampa, FL, USA

Leroux H, Andreu D, Godary-Dejean K (2015) Handling exceptions in petri net-based digital architecture: From formalism to implementation on FPGAs. IEEE Transactions on Industrial Informatics 11(4):897 – 906

Magnin M, Lime D, Roux OH (2008) Symbolic state space of stopwatch petri nets with discrete-time semantics (theory paper). In: Proc. of the 29th Int. Conf. on Application and Theory of Petri Nets and other models of concurrency (ICATPN), Xi'an, China

Magnin M, Molinaro P, Roux OH (2009) Expressiveness of petri nets with stopwatches. discrete-time part. Fundamenta Informaticae 97(1-2):139–176

Merlin PM (1974) A study of the recoverability of computing systems. PhD thesis, Univ. of California

Moalla M, Pulou J, Sifakis J (1978) Synchronized petri nets: A model for the description of non- autonomous sytems. In: Proc. of the 7th Int. Symposium on mathematical foundations of computer science (MFCS), Zakopane, Poland

Pocci M, Demongodin I, Giambiasi N, A G (2016) Synchronizing sequences on a class of unbounded systems using synchronized petri nets. Discrete Event Dynamic Systems 26(1):85–108

Popova L (1991) On time petri nets. Journal of Information Processing and Cybernetics - EIK 27(4):227–244

Reynier PA, Sangnier A (2009) Weak time petri nets strike back! In: Proc. of the 20th Int. Conf. on Concurrency Theory (CONCUR), Bologna, Italy

Ribeiro O, Fernandes JM (2007) Translating synchronous petri nets into promela for verifying behavioural properties. In: Proc. of the IEEE Int. Symposium on Industrial Embedded Systems (SIES), Lisbon, Portugal

Uzam M, Koc I, Gelen G, Aksebzeci B (2009) Asynchronous implementation of discrete event controllers based on safe automation Petri nets. The International Journal of Advanced Manufacturing Technology 41(5-6):595–612

Wegrzyn M, Adamski M, Karatkevich A, Munoz A (2014) FPGA-based embedded logic controllers. In: Proc. 7th Int. Conf. on Human System Interactions (HSI), Costa da Caparica, Portugal