Hindawi Publishing Corporation
Journal of Engineering
Volume 2013, Article ID 970487, 13 pages
http://dx.doi.org/10.1155/2013/970487

Research Article

Controller Synthesis of Time Petri Nets Using Stopwatch

Parisa Heidari and Hanifa Boucheneb

Laboratoire VeriForm, Department of Computer Engineering, Ecole Polytechnique de Montréal, P.O. Box 6079, Station Centre-ville, Montréal, QC, Canada H3C 3A7

Correspondence should be addressed to Parisa Heidari; parisa.heidari@polymtl.ca

Received 30 August 2012; Revised 16 January 2013; Accepted 21 January 2013

Academic Editor: WaiKeung Wong

Scheduling is often a difficult task specially in complex systems. Few tools are targeted at both modeling and scheduling of the systems. In controller synthesis, a scheduler is seen as a controller to manage shared resources and timing requirements of a system. This paper proposes a time Petri net-based approach for controller synthesis and finding a scheduler using stopwatch. The solution suggested here is particularly interesting for preemptive scheduling purposes. This paper deals with time Petri nets with controllable and uncontrollable transitions and assumes that a controllable transition can be suspended and retrieved when necessary. In fact, the paper supposes that every controllable transition can be associated with stopwatch. With this hypothesis, the objective is to model a system by time Petri nets and calculate subintervals where the system violates the given property. Then, the controller associates the corresponding controllable transitions with stopwatch to suspend them in their bad subintervals. The interesting advantage of this solution is that this approach synthesizes an ordinary time Petri net model before adding stopwatch. Therefore, complicated computations and overapproximations required during controller synthesis of time Petri nets associated with stopwatch are avoided.

1. Introduction

A scheduler is a kind of controller that manages shared resources and timing specifications of the system. In the concept of controller synthesis, actions are partitioned into two disjoint sets, *controllable* and *uncontrollable*. Controllable actions are those that can be managed by the controller (forced to happen or prevented from happening). Uncontrollable actions are those that the controller has no control on. As an example, an internal counter in a digital system progresses with each clock cycle until it captures a predefined value and is reset after then. In a multitasking system, a newly arrived task waits for accessing resources. After accessing the required resources, each task spends its execution time and then releases the resources. If the task is periodic, it stays in a passive state before its next arrival. Process arrival depends on its period and its termination depends on the execution time. Then, both process arrival and termination of execution are uncontrollable, whereas access to the shared resources and starting the execution are controllable [1].

In order to manage shared resources among different periodic tasks with different levels of priority, the solution is to suspend a task with the lower priority and let the others with the higher priorities execute and use the resources. Then, the suspended task is retrieved until it finishes its execution. When a task is suspended, the execution time is not progressing. This behavior is modeled by stopwatch.

Stopwatch is an extension of timed models to facilitate modeling of interruption and resumption of a job. Once an interrupt happens, a task is suspended. Later, it is retrieved and continued from where it was interrupted. During the interruption, the clock of interrupted task is stopped while other clocks progress normally. The idea of stopwatch has been discussed and extended to timed automata (TA) as well as time petri nets (TPN) and some types of TA and TPN associated with stopwatch are already introduced [2–5]. In stopwatch, we may have some states whose clocks keep progressing, while in some other states the clock is stopped and keeps the value it had before being interrupted as if the clock has a memory and after being retrieved it continues with its previous value.

In a multitasking real-time system with interruptible tasks and shared resources, a suitable scheduler is necessary to manage the resources and to prevent blocking and deadlock. The scheduler guarantees the well functionality of the system in terms of respecting deadline, priority, and

similar constraints. Each task cannot start execution until the required resources are available. A task releases the occupied resources after finishing its execution.

In multitasking systems, tasks are categorized to periodic, aperiodic, and sporadic [6].

(i) *Periodic Task.* Periodic tasks arrive in regular intervals. Worst-case and best-case execution time is of the specifications defined for periodic tasks. Deadline is the other characteristic defined for these tasks with respect to their worst-case execution time. Deadline in periodic tasks can be either soft or hard. A critical deadline is called hard deadline whereas meeting a soft deadline is not critical.

(ii) *Aperiodic Task.* Tasks with irregular interarrivals are called aperiodic. Aperiodic tasks are usually associated with soft deadlines.

(iii) *Sporadic Task.* Aperiodic tasks with a minimum interarrival are called sporadic. Sporadic tasks are associated with hard deadlines.

A periodic task may have the following three states [7].

(i) *Waiting.* Once a periodic task arrives, it should wait for required resources.

(ii) *Execution.* When all required resources are available, the task starts its execution and spends its execution time in this state.

(iii) *Passive.* After being executed, the task releases the required resources and waits until its next arrival. The time between two consecutive task arrivals is equal with the given period.

In a multitasking system with periodic tasks, the scheduler should manage the shared resources in a way that the predefined period for the tasks is respected. On the other hand, some of the tasks may be associated with a deadline or priority. In that case, the scheduler should suspend the execution of a task with lower priority and let the higher priority tasks to use the resources. Once the tasks with higher priority are executed, the other tasks with lower priorities are retrieved and continue their execution. This behavior is called preemptive scheduling.

In the field of controller synthesis, a scheduler is seen as a controller [7, 8]. The scheduler should manage the shared resources and starts execution of each task in a way that the timing constraints, deadlines, and priorities are respected. Arrival of each task is due to its period and then is uncontrollable. Execution time is also predefined and then is uncontrollable. The only controllable action is the starting of an execution. In preemptive scheduling, a task can be suspended during its execution and leave the resources to some other task with a higher priority.

One solution is to model the system with time Petri nets associated with stopwatch. In this context, each task execution is modeled by a transition equipped with stopwatch and then the corresponding controller is synthesized to manage suspension and resumption of each task such that the corresponding timing constraints are respected. As we will see in this paper, one problem with this solution is that the state space of time Petri nets associated with stopwatch is not exact and requires some overapproximation.

The other problem is that time Petri nets associated with stopwatch does not preserve the boundedness property. Let $\mathcal{N}_1$ be a time Petri net, $\mathcal{N}_2$ be a time Petri net associated with stopwatch, and SCG shows the corresponding state class graph, the following relation holds:

$$\mathcal{N}_1 \text{ is bounded} \Longleftrightarrow \text{SCG of } \mathcal{N}_1 \text{ is finite.} \quad (1)$$

This condition is not always true for a time Petri net associated with stopwatch. In other terms,

$$\mathcal{N}_2 \text{ is bounded} \nRightarrow \text{SCG of } \mathcal{N}_2 \text{ is finite.} \quad (2)$$

Indeed, if the number of reachable markings in a time Petri net associated with stopwatch is finite, the number of state classes is not necessarily finite [3]. These limitations hinder the controller synthesis of time Petri nets with stopwatch.

In this paper, we suggest an alternative solution. We propose to synthesize a time Petri net without stopwatch where the transitions corresponding to the task executions are considered controllable. We extract the controller (i.e., scheduling strategy) and then add the stopwatch to model the controlled system (the system and its scheduler). In fact, we suggest to implement the synthesized controller by means of stopwatch.

In the following, we discuss how the algorithm suggested in [9] for controller synthesis of time Petri nets is used for scheduling purposes to synthesize a controller for interruptible tasks. The algorithm of [9] is permissive and the computed controller is restricting time intervals making the system to satisfy the given properties. In this paper, we suggest to suspend a task during its bad subinterval. This approach is useful for preemptive scheduling purposes where safety properties correspond to meeting deadlines, priorities, preventing deadlock for shared resources, and so forth.

In this paper, we use a synthesized controller (output of the algorithm of [9] in particular), and we show how to control the system using stopwatch. In order to implement the controller by means of stopwatch, we assume that it is possible to associate each controllable transition with a stopwatch so as to suspend or resume it whenever needed.

The rest of this paper is organized as follows. Section 2 is dedicated to time Petri nets, their definition, and semantics. Section 3 presents a literature review on different stopwatch Petri nets. Section 4 has a brief survey on the solution proposed in [9]. Section 5 synthesizes a controller using stopwatch. Section 6 presents an example of a multitasking system. Finally, Section 7 gives the conclusion and future work.

2. Time Petri Nets

A time Petri net (TPN) is a Petri net augmented with time intervals associated with transitions. This paper focuses on the classical semantics, called intermediate semantics in [10], in the context of monoserver and strong semantics [11]. Formally, a TPN is a tuple $(P, T, \text{Pre}, \text{Post}, M_0, \text{Is})$ where

(i) P and T are finite sets of places and transitions such that $(P \cap T = \emptyset)$.

(ii) Pre and Post are the backward and the forward incidence functions (Pre, Post : $P \times T \rightarrow \mathbb{N}$, where $\mathbb{N}$ is the set of nonnegative integers).

(iii) M_0 is the initial marking ($M_0 : P \rightarrow \mathbb{N}$).

(iv) Is is the static interval function (Is : $T \rightarrow \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\})$), $\mathbb{Q}^+$ is the set of nonnegative rational numbers. Is associates with each transition t an interval called the static firing interval of t. Bounds $\downarrow$ Is(t) and $\uparrow$ Is(t) of the interval Is(t) are the minimum and maximum firing delays of t, respectively.

In a controllable time Petri net, transitions are partitioned into controllable and uncontrollable transitions, denoted by T_c and T_u, respectively (with $T_c \cap T_u = \emptyset$ and $T = T_c \cup T_u$). For the sake of simplicity and clarification, in this paper the controllable transitions are depicted as white bars, while the uncontrollable ones as black bars.

A TPN, is called bounded if for every reachable marking M, there is a bound $b \in \mathbb{N}^p$ where $M \leq b$ holds. In this condition, p stands for the number of places in P.

Let M be a marking and t a transition. Transition t is enabled for M if and only if all required tokens for firing t are present in M, that is, $\forall p \in P$, $M(p) \geq \text{Pre}(p,t)$. Firing of t leads to the marking M' defined by: $\forall p \in P$, $M'(p) = M(p) - \text{Pre}(p,t) + \text{Post}(p,t)$. We denote En($M$) the set of transitions enabled for M, that is, $\text{En}(M) = \{t \in T \mid \forall p \in P, \text{Pre}(p,t) \leq M(p)\}$. For $t \in \text{En}(M)$, we denote CF(M,t) the set of transitions enabled in M but in conflict with t, that is, $\text{CF}(M,t) = \{t' \in \text{En}(M) \mid t' = t \vee \exists p \in P, M(p) < \text{Pre}(p,t') + \text{Pre}(p,t)\}$. Let $t \in \text{En}(M)$ and M' the successor marking of M by t; a transition t' is said to be newly enabled in M' if and only if t' is not enabled in the intermediate marking (i.e., $M - \text{Pre}(\cdot,t)$) or $t' = t$. We denote New(M',t) the set of transitions newly enabled M', by firing t from M; that is, $\text{New}(M',t) = \{t' \in \text{En}(M') \mid t = t' \vee \exists p \in P, M'(p) - \text{Post}(p,t) < \text{Pre}(p,t')\}$.

The TPN state is defined as a pair (M, Id), where M is a marking and Id is a firing interval function ($\text{Id} : \text{En}(M) \rightarrow \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\})$). The initial state is (M_0, Id_0) where M_0 is the initial marking and $\text{Id}_0(t) = \text{Is}(t)$, for $t \in \text{En}(M_0)$. Let (M, Id) and (M', Id$'$) be two states of the TPN model, $\theta \in \mathbb{R}^+$ and $t \in T$. The transition relation $\rightarrow$ over states is defined as follows.

(i) $(M, \text{Id}) \xrightarrow{\theta} (M', \text{Id}')$, also denoted by $(M, \text{Id}) + \theta$, if and only if the state (M', Id') is reachable from state (M, Id), after θ time units, that is, $\bigwedge_{t' \in \text{En}(M)} \theta \leq \uparrow \text{Id}(t')$, $M' = M$, and $\forall t'' \in \text{En}(M')$, $\text{Id}'(t'') = [\text{Max}(\downarrow \text{Id}(t'') - \theta, 0), \uparrow \text{Id}(t'') - \theta]$.

(ii) $(M, \text{Id}) \xrightarrow{t} (M', \text{Id}')$ if and only if the state (M', Id') is reachable from the state (M, Id) by immediately firing transition t, that is, $t \in \text{En}(M)$, $\downarrow \text{Id}(t) = 0$, $\forall p \in P$, $M'(p) = M(p) - \text{Pre}(p,t) + \text{Post}(p,t)$, and $\forall t' \in \text{En}(M')$, $\text{Id}'(t') = \text{Is}(t')$, if $t' \in \text{New}(M',t)$, and $\text{Id}'(t') = \text{Id}(t')$, otherwise.

The TPN state space is the structure ($\mathcal{Q}$, $\rightarrow$, q_0), where $q_0 = (M_0, \text{Id}_0)$ is the initial state of the TPN and $\mathcal{Q} = \{q \mid q_0 \xrightarrow{*} q\}$ ($\xrightarrow{*}$ being the reflexive and transitive closure of the relation $\rightarrow$ defined above) is the set of reachable states of the model. A run in the TPN state space ($\mathcal{Q}$, $\rightarrow$, q_0), of a state $q \in \mathcal{Q}$, is a maximal sequence $\rho = q_1 \xrightarrow{\theta_1} q_1 + \theta_1 \xrightarrow{t_1} q_2 \xrightarrow{\theta_2} q_2 + \theta_2 \xrightarrow{t_2} q_3, \ldots$, such that $q_1 = q$. By convention, for any state q_i, the relation $q_i \xrightarrow{0} q_i$ holds. The sequence $\theta_1 t_1 \theta_2 t_2 \ldots$ is called the timed trace of ρ. The sequence $t_1 t_2, \ldots$, is called the firing sequence (untimed trace) of ρ. A marking M is reachable if and only if $\exists q \in \mathcal{Q}$ s.t. its marking is M. Runs (resp., timed/untimed traces) of the TPN are all runs (resp., timed/untimed traces) of the initial state q_0.

To use enumerative analysis techniques with time Petri nets, their generally infinite state spaces are abstracted. Abstraction techniques construct by removing some irrelevant details, a finite contraction of the state space of the model, which preserves properties of interest. For best performances, the contraction should also be the smallest possible and computed with the minimal resources in terms of time and space. The preserved properties are usually verified using standard analysis techniques on the abstractions [12]. Several state space abstraction methods have been proposed in the literature for time Petri nets (the *state class graph* (SCG) [13], the *zone based graph* (ZBG) [14], etc.). They may differ in the characterization of states (interval or clock states), the state agglomeration criteria, the abstract states representation, the preserved properties (markings, linear or branching properties), and their size.

These abstractions are finite for all bounded time Petri nets. However, abstractions based on clocks are less interesting than the interval-based abstractions when only linear properties are of interest. Indeed, abstractions based on clocks do not enjoy naturally the finiteness property for bounded TPN with unbounded intervals as it is the case for abstractions based on intervals. The finiteness is enforced using an approximation operation, which may involve some overhead computation. The algorithm of [9] is based on the state class graph method.

2.1. The State Class Graph Method. In the state class graph method [13], all states reachable by the same firing sequence from the initial state are agglomerated in the same node and considered modulo the relation of equivalence defined by: two sets of states are equivalent if and only if they have the same marking and the same firing domain (the firing domain of a set of states is the union of the firing domains of its states). All equivalent sets are agglomerated in the same node called a *state class* defined as a pair $\alpha = (M, F)$, where M is a marking and F is a formula which characterizes the firing domain of α. For each transition t_i enabled in M, there is a variable $\underline{t}_i$, in F, representing its firing delay. F can be rewritten as a set of atomic constraints of the form $\underline{t}_i - \underline{t}_j \leq c$, $\underline{t}_i \leq c$ or $-\underline{t}_j \leq c$, where t_i, t_j are transitions, $c \in \mathbb{Q} \cup \{\infty\}$, and $\mathbb{Q}$ is the set of rational numbers. (for economy of notation, we use operator $\leq$ even if $c = \infty$):

Each domain is expressed by the canonical form that is usually encoded by a difference bound matrix (DBM) [15]. The canonical form of F is encoded by the DBM D (a square

matrix) of order $|\mathrm{En}(M)| + 1$ defined by: $\forall t_i, t_j \in \mathrm{En}(M) \cup \{t_0\}, d_{ij} = (\leq, \mathrm{Sup}_F(\underline{t}_i - \underline{t}_j))$, where t_0 ($t_0 \notin T$) represents a fictitious transition whose delay is always equal to 0 and $\mathrm{Sup}_F(\underline{t}_i - \underline{t}_j)$ is the largest value of $\underline{t}_i - \underline{t}_j$ in the domain of F. Its computation is based on the shortest path *Floyd-Warshall*'s algorithm and is considered as the most costly operation (cubic in the number of variables in F). The canonical form of a DBM makes some operations over formulas like the test of equivalence easier. Two formulas are equivalent if and only if the canonical forms of their DBMs are identical.

The initial state class is $\alpha_0 = (M_0, F_0)$, where $F_0 = \bigwedge_{t_i \in \mathrm{En}(M_0)} \downarrow \mathrm{Is}(t_i) \leq \underline{t}_i \leq \uparrow \mathrm{Is}(t_i)$. Let $\alpha = (M, F)$ be a state class and t_f a transition and $\mathrm{succ}(\alpha, t_f)$ the set of states defined by: $\mathrm{succ}(\alpha, t_f) = \{q' \in \mathcal{Q} \mid \exists q \in \alpha, \exists \theta \in \mathbb{R}^+ \text{ s.t. } q \xrightarrow{\theta} q + \theta \xrightarrow{t_f} q'\}$. The state class α has a successor by t_f (i.e., $\mathrm{succ}(\alpha, t_f) \neq \emptyset$), if and only if t_f is enabled in M and can be fired before any other enabled transition; that is, the following formula is consistent: $F \wedge (\bigwedge_{t_i \in \mathrm{En}(M)} \underline{t}_f \leq \underline{t}_i)$. A formula F is consistent if and only if there is, at least, one tuple of values that satisfies, at once, all constraints of F. In this case, the firing of t_f leads to the state class $\alpha' = (M', F') = \mathrm{succ}(\alpha, t_f)$ computed as follows [13].

(1) $\forall p \in P, M'(p) = M(p) - \mathrm{Pre}(p, t_f) + \mathrm{Post}(p, t_f)$.

(2) $F' = F \wedge (\bigwedge_{t_i \in \mathrm{En}(M)} \underline{t}_f - \underline{t}_i \leq 0)$.

(3) Replace in F' each $\underline{t}_i \neq \underline{t}_f$, by $(\underline{t}_i + \underline{t}_f)$.

(4) Eliminate by substitution $\underline{t}_f$ and each $\underline{t}_i$ of transition conflicting with t_f in M.

(5) Add constraint $\downarrow \mathrm{Is}(t_n) \leq \underline{t}_n \leq \uparrow \mathrm{Is}(t_n)$, for each transition $t_n \in \mathrm{New}(M', t_f)$.

Formally, the SCG of a TPN model is a structure $(\mathscr{CC}, \rightarrow, \alpha_0)$, where $\alpha_0 = (M_0, F_0)$ is the initial state class, $\forall t_i \in T, \alpha \xrightarrow{t_i} \alpha'$ iff $\alpha' = \mathrm{succ}(\alpha, t_i) \neq \emptyset$ and $\mathscr{CC} = \{\alpha \mid \alpha_0 \xrightarrow{*} \alpha\}$. The SCG is finite for all bounded TPNs and preserves linear properties [16]. Let $\alpha = (M, F)$ be a state class and $\omega \in T^+$ a sequence of transitions firable from α. We denote $\mathrm{succ}(\alpha, \omega)$ the state class reachable from α by firing successively transitions of ω. We define inductively this set as follows: $\mathrm{succ}(\alpha, \omega) = \alpha$, if $\omega = \epsilon$ and $\mathrm{succ}(\alpha, \omega) = \mathrm{succ}(\mathrm{succ}(\alpha, \omega'), t_i)$, if $\omega = \omega' \cdot t_i$.

As an example, Figure 2 shows the state class graph of the TPN presented at Figure 1. Its state classes are reported in Table 1.

Let ω be a sequence of transitions firable from α, the same transition may be newly enabled several times. To distinguish among different enablings of the same transition t_i, we denote t_i^k for $k > 0$ the transition t_i (newly) enabled by the kth transition of the sequence; t_i^0 denotes the transition t_i enabled in M. Let $\omega = t_1^{k_1}, \ldots, t_m^{k_m} \in T^+$ with $m > 0$ be a sequence of transitions firable from α s.t. $\mathrm{succ}(\alpha, \omega) \neq \emptyset$. We define $\mathrm{Fire}(\alpha, \omega)$ the largest subclass α' of α (i.e., $\alpha' \subseteq \alpha$) s.t. ω is firable from all its states, that is, $\mathrm{Fire}(\alpha, \omega) = \{q_1 \in \alpha \mid \exists \theta_1, \ldots, \theta_m, q_1 \xrightarrow{\theta_1} q_1 + \theta_1 \xrightarrow{t_1^{k_1}} q_2, \ldots, q_m + \theta_m \xrightarrow{t_m^{k_m}} q_{m+1}\}$.

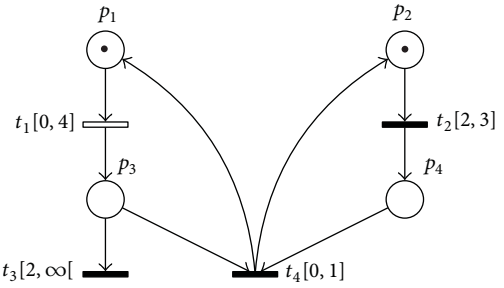

Figure 1: A simple Petri net with $T_c = \{t_1\}$.

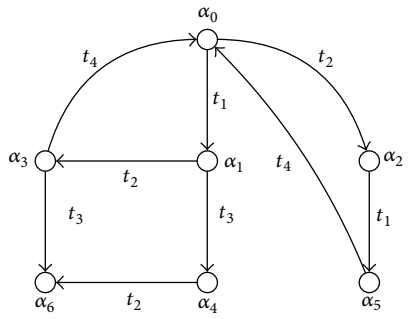

Figure 2: The state graph of the TPN presented at Figure 1.

3. Literature Review

The idea of suspension and resumption of a task in a system is modeled in different ways in timed automata and time Petri nets [2–4, 17, 18]. In [2], the authors have introduced stopwatch automata (SWA) as a subclass of timed linear hybrid automata. In stopwatch automata, an additional binary variable is defined to show the rate of time progression. The clocks may have two velocities (time derivation): zero or one. Zero signifies stopped while one signifies normal progress. If clocks are running, they progress with a global rate, identical to all nonstopped clocks of the model. The authors have shown that stopwatch automata is as expressive as timed languages.

In [3], the authors have introduced inhibitor hyperarcs to interrupt an enabled transition. Once a place p connected to a transition t via an inhibitor arc is marked, the transition t is suspended and stops firing. In other words, once an inhibitor arc is enabled, the corresponding transition is interrupted. They have called these nets IHTPN (time Petri nets with inhibitor hyperarcs). In Figure 3, t_1 is enabled. As soon as P_2 is marked, firing t_1 is suspended. Note that an inhibitor hyperarc is not graphically presented by a classical arrow; instead, it is depicted with an empty circle at the extreme of the edge.

In [17], the authors have discussed an extension of time Petri nets adapted for scheduling. This extension is called scheduling extended time Petri nets (SETPN). SETPN is suitable to model concurrent tasks and shared resources with fixed priority. The behavior of scheduler is implicitly included

Table 1: The state classes of the TPN presented at Figure 2.

$\alpha_0 : p_1 + p_2$	$0 \le \underline{t}_1 \le 4 \wedge 2 \le \underline{t}_2 \le 3$	$\alpha_1 : p_2 + p_3$	$0 \le \underline{t}_2 \le 3 \wedge 2 \le \underline{t}_3$	$\alpha_6 : p_4$
$\alpha_2 : p_1 + p_4$	$0 \le \underline{t}_1 \le 2$	$\alpha_3 : p_3 + p_4$	$0 \le \underline{t}_3 \wedge 0 \le \underline{t}_4 \le 1$	
$\alpha_4 : p_2$	$0 \le \underline{t}_2 \le 1$	$\alpha_5 : p_3 + p_4$	$2 \le \underline{t}_3 < \infty \wedge 0 \le \underline{t}_4 \le 1$	

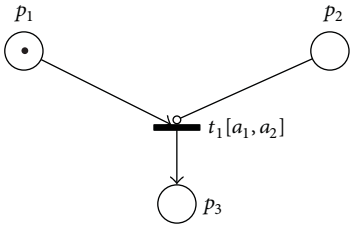

Figure 3: A simple example of time Petri nets with inhibitor hyperarc. t_1 is active if p_2 is not marked, otherwise it is suspended.

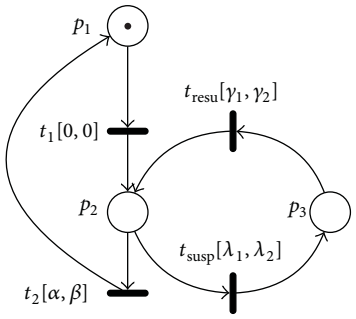

Figure 4: An interruptible task modeled by SWPN, reported from [4].

in the model and the scheduler is not modeled as a separate agent.

In [17], time Petri nets are associated with two more properties γ and ω standing for resource allocation and priority, respectively. As an example, suppose two different concurrent tasks sharing the same resource (γ is identical for both). Although both transitions are enabled, the one with a higher priority is active while the other is suspended. For each marking M, a function Act determines whether M is active or not.

The time Petri nets presented in [17] assign both priority and resources to places. It is particularly suitable for scheduling approaches. State space analysis is done mapping scheduling Petri nets to hybrid automata (state class stopwatch automata). The advantage of this solution is using an automated tool like HYTECH [19] already existing for manipulation of hybird automata.

Another extension of time Petri nets called preemptive time Petri nets is discussed in [18]. Preemptive time Petri nets are suitable for scheduling and preemptive approaches. In Preemptive time Petri nets, resource and priority are characteristics of transitions while in SETPN they were assigned to places. The preemptive time Petri nets as well as scheduling extended time Petri nets are adapted specifically for scheduling purposes with fixed priority rather than general cases of interruption and resumption of some tasks. A more generalized model comes in [4, 5].

Another stopwatch model comes in [4, 5]. Post- and Preinitialized stopwatch Petri nets (SWPN) [4, 5] also model interruption and retrieving of tasks. In such stopwatch Petri nets, transitions are partitioned into two disjoint subclasses $T = T_{\text{int}} \cup T_{\text{no-int}}$ where T_{int} is the set of interruptible transitions and $T_{\text{no-int}}$ is the set of noninterruptible transitions. Stopwatches can suspend enabled interruptible transitions. For each transition t_i, there is a function $v(t_i)$ representing the value of its associated stopwatch. If $t_i \in T_{\text{int}}$, $v(t_i)$ signifies the time elapsed since t_i was first enabled, whereas if $t_i \in T_{\text{no-int}}$, $v(t_i)$ represents the time elapsed since t_i was last enabled. A transition is called firable if it is enabled ($M > \text{Pre}(t_i)$) and $\downarrow \text{Is}(t_i) \le v(t_i) \le \uparrow \text{Is}(t_i)$.

In contrary with IHTPN where a marked place suspended an enabled transition, in this model stopwatch consumes some tokens to disable an enabled transition. It is implemented by means of some extra places/transitions. A stopwatch transition is in conflict with an interruptible transition. Thus, consuming a token can suspend an interruptible enabled transition. With this model, an interrupt is unpredictable and we do not know when it happens. In fact at any time, the interruptible task and the interrupt both have equal chance to happen. The simple Petri nets of Figure 4 reported from [4] are a simple example of SWPN of an interruptible task.

Let t_i be a transition and M a marking, t_i is called:

(i) *enabled*. If $M \ge \text{Pre}(t_i)$ denoted by $t_i \in \text{En}(M)$;

(ii) *firable*. If $t_i \in \text{En}(M) \wedge \downarrow \text{Is}(t_i) \le v(t_i) \le \uparrow \text{Is}(t_i)$, denoted by $t_i \in \text{Firable}(M)$;

(iii) *suspended*. If $\text{Pre}(t_i) > M \wedge v(t_i) > 0$, denoted by $t_i \in \text{susp}(M)$.

The notion $\uparrow \text{enabled}(t_i, M, t_k)$ indicates that t_i is newly enabled by firing of t_k at marking M.

The new concept Preinitialization defined in this model refers to noninterruptible transitions. For every $t_i \in T_{\text{no-int}}$, t_i is firable if $v(t_i)$ has already been initialized when t_i becomes enabled. In other words, $v(t_i) = 0$ when $t_i \in \uparrow \text{enabled}(t_i, M, t_k)$.

The concept, postinitialization is defined for interruptible transitions. When an interruptible transition is fired the associated $v(t_i)$ is initialized ($v(t_i) = 0$). Figures 5 and 6 reported from [4] show the difference between time evolution in an interruptible transition and a noninterruptible transition. In these figures, the notions E, D, and F stand for enabling, disabling, and firing of a transition, respectively.

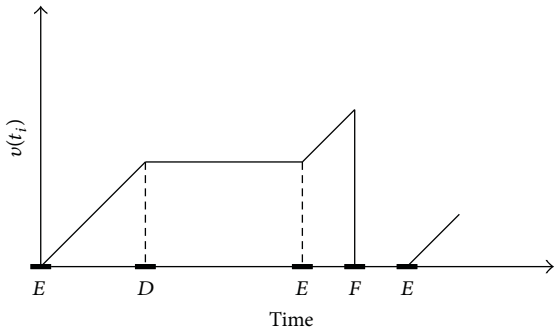

Figure 5: Time elapses, t_i is an interruptible transition.

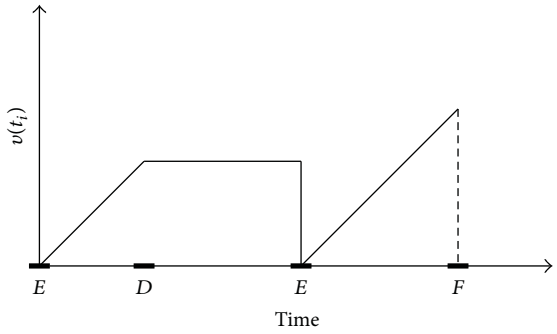

Figure 6: Time elapses, t_i is a noninterruptible transition.

In brief, according to the types of transitions, two types of clock initialization are defined.

(i) *Preinitialization*. Given a transition $t_i \in T_{\text{no-int}}$ where $T_{\text{no-int}}$ is the set of noninterruptible transitions, clock initialization is called preinitialization happening when t_i is recently enabled. As soon as a transition becomes enabled, its associated clock is initialized.

(ii) *Postinitialization*. Given a transition $t_i \in T_{\text{int}}$ where T_{int} is the set of interruptible transitions, clock initialization is called post-initialization happening after firing t_i; it means that the transition initializes the clock after being fired. Post-initialization is dependent on transition firing.

In [4], continuous and discrete transitions are defined as follows.

Let $d \in \mathbb{R}^+$, a continuous transition is denoted by $(M, v) \xrightarrow{d} (M, v')$ iff $\forall t_i \in T$:

(i) $v'(t_i) = v(t_i)$, if the transition t_i is not enabled;

(ii) $v'(t_i) = v(t_i) + d$ if t_i is enabled;

(iii) $M \geq \text{Pre}(t_i) \Rightarrow v' \leq \uparrow \text{Is}(t_i)$.

And a discrete transition is denoted by $(M, v) \xrightarrow{t_i} (M', v')$ iff $\forall t_i \in T$:

(i) $t_i \in \text{Firable}(M)$,

(ii) $M' = M - \text{Pre}(t_i) + \text{Post}(t_i)$,

(iii) $v'(t_i) = 0$ if $t_i \in T_{\text{int}}$ (postinitialization),

(iv) $\forall t_k \in T, v'(t_k) =$

(a) 0 if $t_k \in \uparrow \text{enabled}(t_k, M, t_i)$; (preinitialization)

(b) $v(t_i)$ Otherwise.

In order to perform further timing analysis, the same as scheduling extended time Petri nets of [17], the SWPN is transformed to hybrid automaton. In addition, a forward algorithm is suggested to compute reachable states.

If the number of stopwatches increases, for example if all of the transitions are interruptible, the complexity of calculation will highly increase. Scheduling time Petri nets and preemptive time Petri nets are both subclasses of stopwatch Petri nets. An ordinary TPN is also a SWPN where $T_{\text{int}} = \emptyset$.

After having a survey on different stopwatch Petri nets available in the literature, in the following, we will have a brief review on the algorithm of [9] and then investigate how to integrate stopwatch in the controller synthesis approach of [9].

4. Safety Controller Synthesis Approach Proposed in [9]

Let $\mathcal{N}$ be a TPN with controllable and uncontrollable transitions ($T = T_c \cup T_u$) and a set of markings to be avoided (bad). For a safety property, the approach proposed in [9] (see Algorithms 1 and 2) consists of exploring, on-the-fly and path by path, the state class graph of the TPN $\mathcal{N}$ while collecting the bad sequences and the losing states (sequences or states that lead to bad markings). The list Passed is used to retrieve the set of state classes processed so far, their bad sequences, and their losing subclasses. For a given state class α and a bad sequence ω feasible from α, the function Fire is used to compute forwardly all states of α which lead by a sequence of transitions to an undesirable marking (i.e., a losing subclass of α). The function explore receives parameters α being the class under process, t the transition leading to α, and $\mathscr{C}$ the set of traveled classes in the current path. Succinctly, it recursively computes the bad sequences of α (from bad sequences of its successors) and verifies whether or not α can be controlled so as to avoid its bad sequences. It returns the set of bad sequences if they cannot be avoided from α. Otherwise, it returns an empty set, which means that α can be controlled (i.e., α has no bad sequences or there is at least one controllable transition t_c such that its firing interval in α can be restricted so as to avoid reaching bad states). The restriction of the interval of t_c in α is obtained by subtracting from its interval in α, intervals of t_c in its bad subclasses.

This approach tries to control the system behavior starting from the last to the first state class of bad paths. If it fails to control a state class of a path, so as to avoid all bad state classes, the algorithm tries to control its previous state classes. If it succeeds to control a state class, there is no need to control its predecessors. The aim is to limit as little as possible the behavior of the system (more permissive controller).

In [20], we have proven that this approach gives the maximally permissive controller and if it fails to compute the

Function main (TPN $\mathcal{N}$, Markings bad)
Where $\mathcal{N}$ **is a TPN**
bad **is a set of bad markings**.
Let T_c **be the set of controllable transitions of** $\mathcal{N}$ **and** α_0 **the initial state class of** $\mathcal{N}$.
Passed = $\emptyset$
If (explore $(\alpha_0, \epsilon, \{\alpha_0\}) \neq \emptyset$) **then**
{**Controller does not exist**}
return
end if
for all $((\alpha, \Omega, \text{LI}) \in \text{Passed})$ **do**
$\text{Ctrl}[\alpha] = \bigcup_{(t_c, t_s, \text{BI}) \in \text{LI}} \{(t_c, t_s, \text{INT}(\alpha, \underline{t}_c - \underline{t}_s) - \text{BI})\}$
end for
(*Remarks:)
$\alpha = (M, F)$;
$\text{En}_c(M) = \text{En}(M) \cap T_c$;
$\text{En}_c^0(M) = \text{En}_c(M) \cup \{t_0\}$;
$\text{New}_c(M, t) = \text{New}(M, t) \cap T_c$;
$\text{New}(M_0, \epsilon) = \text{En}(M_0)$;
t_0 is a fictitious transition whose time variable is fixed at 0.
$\text{Dep}(\alpha, t, \text{LI}) \equiv$
$\exists (t_c, t_s, \text{BI}) \in \text{LI}, t_c \notin \text{New}(M, t) \wedge (t_s \notin \text{New}(M, t) \vee$
$\text{INT}(\alpha, \underline{t}_c - \underline{t}_0) \not\subseteq \bigcap_{I \in \text{BI}} (I \oplus \text{INT}(\alpha, \underline{t}_s - \underline{t}_0)))$

Algorithm 1: On-the-fly algorithm for the safety control of TPN-Part I.

Function Traces explore (Class α, Trans t, Classes $\mathscr{C}$)
if $(\exists \Omega, \text{LI}$ **s.t.** $(\alpha, \Omega, \text{LI}) \in$ Passed**) then**
if $(\Omega \neq \emptyset \wedge \text{Dep}(\alpha, t, \text{LI}))$ **then**
return $\{t \cdot \omega \mid \omega \in \Omega\}$
end if
return $\emptyset$
end if
if $(M \in \text{bad})$ **then**
return $\{t\}$
end if
Traces $\Omega = \emptyset$;
for all $t' \in \text{En}(M)$ **s.t** $\text{succ}(\alpha, t') \neq \emptyset \wedge \text{succ}(\alpha, t') \notin \mathscr{C}$
do
$\Omega = \Omega \cup \text{explore}(\text{succ}(\alpha, t'), t', \mathscr{C} \cup \{\text{succ}(\alpha, t')\})$
end for {Ω **contains all bad sequences of** α.}
if $(\Omega = \emptyset)$ **then**
Passed = Passed $\cup \{(\alpha, \emptyset, \emptyset)\}$
return $\emptyset$
end if
$\text{LI} = \{(t_c, t_s, \text{BI}) \mid (t_c, t_s) \in \text{En}_c(M) \times \text{En}_c^0(M) \wedge$
$\text{BI} = \bigcup_{\omega \in \Omega} \text{INT}(\text{Fire}(\alpha, \omega), \underline{t}_c - \underline{t}_s) \subset \text{INT}(\alpha, \underline{t}_c - \underline{t}_s)\}$
Passed = Passed $\cup \{(\alpha, \Omega, \text{LI})\}$
if $(\text{Dep}(\alpha, t, \text{LI}))$ **then**
return $\{t \cdot \omega \mid \omega \in \Omega\}$
end if
return $\emptyset$

Algorithm 2: On-the-fly algorithm for the safety control of TPN-Part II.

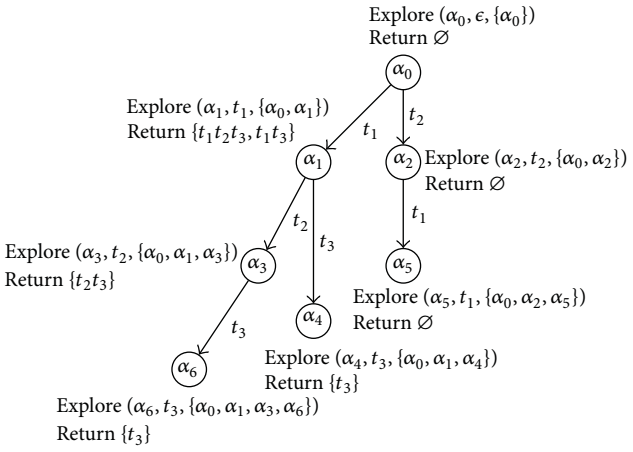

Figure 7: Applying Algorithms 1 and 2 on the TPN at Figure 1 for AG not $p_1 + p_3 = 0$.

controller then the controller does not exist. We have also investigated the hardness complexity of the approach step by step.

(i) The complexity of computing BI of a state class (in worst case) is $O(K \times L \times {n_t}^2)$ where K is the number of paths starting from the state class and leading to bad markings, L is the maximal length of such paths, and n_t is the number of transitions in the model.

(ii) The complexity for computing Dep relation used in Algorithms 1 and 2 is $O(|\mathrm{LI}| \times m \times |\mathrm{BI}|)$, where m is the number of places in the model (complexity of testing $\mathrm{New}(M, t)$).

In [14], the authors have discussed the completeness complexity for bounded TPNs. They have proven that model checking of TPN-TCTL formula on a bounded TPN is PSPACE-complete.

Let us explain this approach, by means of an example. Consider the TPN shown at Figure 1, its state class graph presented at Figure 2 and its state classes reported in Table 1. Suppose that only t_1 is controllable (i.e., $T_c = \{t_1\}$) and we want to avoid reaching markings where places p_1 and p_3 are both not marked (i.e., state classes α_4 and α_6) by choosing appropriately the firing intervals of t_1. For this example, we have $T_c = \{t_1\}$, bad $= \{p_2, p_4\}$, Passed $= \emptyset$, and $\alpha_0 = (p_1 + p_2, 0 \le \underline{t}_1 \le 4 \wedge 2 \le \underline{t}_2 \le 3)$.

The process starts by calling explore$(\alpha_0, \epsilon, \{\alpha_0\})$ (see Figure 7). Since α_0 is not in Passed and its marking is not forbidden, explore is successively called for the successors of α_0: explore$(\alpha_1, t_1, \{\alpha_0, \alpha_1\})$ and explore$(\alpha_2, t_2, \{\alpha_0, \alpha_2\})$. In explore of α_1, function explore is successively called for α_3 and α_4. In explore of α_3, function explore is called for the successor α_6 of α_3 by t_3: explore$(\alpha_6, t_3, \{\alpha_0, \alpha_1, \alpha_3, \alpha_6\}$. For the successor of α_3 by t_4 (i.e., α_0), there is no need to call explore as it belongs to the current path. Since α_6 has a forbidden marking, explore of α_6 returns to explore of α_3 with $\{t_3\}$, which, in turn, adds $(\alpha_3, \{t_2t_3\}, \emptyset)$ to Passed and returns to explore of α_1 with $\{t_2t_3\}$.

In explore of α_1, function explore is called for α_4 (explore$(\alpha_4, t_3, \{\alpha_0, \alpha_1, \alpha_4\})$). This call returns, to explore of α_1, with $\{t_3\}$, since α_4 has a forbidden marking. In explore of α_1, the tuple $(\alpha_1, \{t_2t_3, t_3\}, \emptyset)$ is added to Passed and $\{t_1t_2t_3, t_1t_3\}$ is returned to explore of α_0. Then, explore of α_0 calls explore$(\alpha_2, t_2, \{\alpha_0, \alpha_2\})$, which in turn calls explore$(\alpha_5, t_1, \{\alpha_0, \alpha_2, \alpha_5\})$. Since α_5 has only one successor (α_0) and this successor belongs to the current path, the call of explore for α_5 adds $(\alpha_5, \emptyset, \emptyset)$ to Passed and returns to explore of α_2 with $\emptyset$, which, in turn, returns to explore of α_0.

After exploring both successors of α_0, in explore of α_0, we get in $\Omega = \{t_1t_2t_3, t_1t_3\}$ the set of bad paths of α_0. As the state class α_0 has a controllable transition t_1, its bad subclasses are computed: $\mathrm{Fire}(\alpha_0, t_1t_2t_3) = \{(p_1 + p_2, 0 \le \underline{t}_1 \le 2 \wedge 2 \le \underline{t}_2 \le 3 \wedge 1 \le \underline{t}_2 - \underline{t}_1 \le 3)$ and $\mathrm{Fire}(\alpha_0, t_1t_3) = (p_1 + p_2, 0 \le \underline{t}_1 \le 1 \wedge 2 \le \underline{t}_2 \le 3 \wedge 2 \le \underline{t}_2 - \underline{t}_1 \le 3)\}$. The firing interval of t_1 in α_0 ($[0, 4]$) is not covered by the union of intervals of t_1 in bad subclasses of α_0 ($[0, 2] \cup [0, 1] \neq [0, 4]$). Then, $(\alpha_0, \{t_1t_2t_3, t_1t_3\}, \{(t_1, t_0, \{[0, 2]\})\})$ is added to Passed. As t_1 is newly enabled, the empty set is returned to the function main, which concludes that a controller exists. According to the list Passed, α_0 needs to be controlled ($\mathrm{Ctrl}[\alpha_0] = \{(t_1, t_0, \{[0, 4] - [0, 2]\})\}$). For all others, there is nothing to do.

5. Controller Synthesis and Stopwatch

In this section, we consider time Petri nets with controllable/uncontrollable transitions and show how such a system is controlled by associating some of the controllable transitions with stopwatch. First, we apply the on-the-fly algorithm of [9] and calculate the appropriate controller. Then, instead of restricting time intervals of corresponding controllable transitions, we associate them with stopwatch. In fact, we suspend some transition during its bad subinterval to control a system so as to satisfy a given property.

We consider the inhibitor hyperarcs of [3]. The controller shall suspend a controllable transition in its bad subinterval and retrieve it otherwise. Let us remember the example of Figure 1 with the state class graph of Figure 2 and Table 1. We have seen that in order to prevent the system entering the forbidden state classes α_4 and α_6, the controller should prevent t_1 from firing before $]2, 4]$. Thus, an inhibitor hyperarc is added to the transition t_1. This inhibitor hyperarc connects t_1 to a place called p_{susp}. At the beginning, this place is marked and then t_1 is suspended. At $[2, 2]$ the token of p_{susp} is consumed and the corresponding hyperarc becomes disabled. The controllable transition t_1 is now firable again. The token is returned to place p_{susp} after t_1 is fired. Thus, the controller suspends t_1 during its bad subinterval and resumes it after then. At the beginning, all clocks are initialized to zero. Then, time elapses but in $[0, 2]$, this transition is suspended and hence its clock does not elapse anymore. Later at time $[2, 2]$, this transition is retrieved and becomes active. We do not need to delay it anymore, that is why the lower bound of the interval associated with t_1 is modified to 0. Now the clock starts elapsing. Within 2 time units t_1 should be fired (i.e., when the actual time of the general clock of the system reaches to 4). Then, the interval associated with

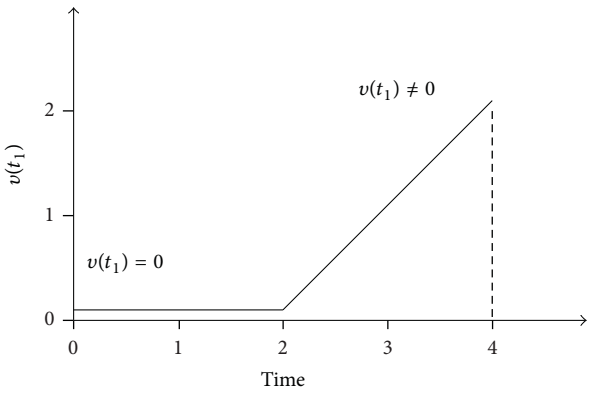

Figure 8: Clock evaluation of t_1 in the controlled TPN of Figure 9.

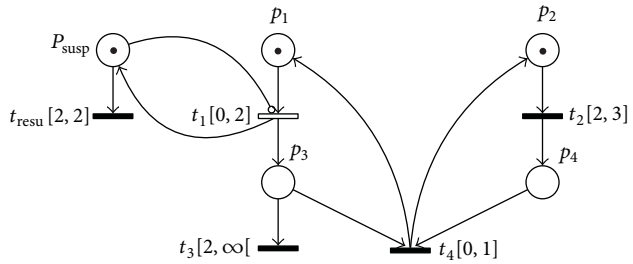

Figure 9: Time Petri net of Figure 1, controlled by inhibitor hyperarcs ($T_c = \{t_1\}$).

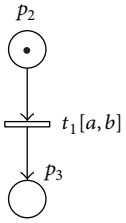

Figure 10: A simple time Petri net ($T_c = \{t_1\}$).

t_1 in a controlled TPN associated with stopwatch is $[0, 2]$. Figure 8 represents the clock evaluation of the transition t_1. The controlled Petri nets of this example are presented in Figure 9.

In summary, our goal is to synthesize the state class graph of the system through Algorithms 1 and 2 (see [9]) and calculate bad subintervals of the corresponding controllable transitions. Then, suspend the appropriate controllable transitions during their bad subintervals. Note that in [9], we were restricting and limiting time intervals while in this approach the idea is to delay and suspend them. In this research, we suppose every controllable transition can be associated with a stopwatch if needed. As explained earlier in Section 1, in a multitasking system with periodic tasks only execution of a task is controllable. Then, those transitions modeling the task executions are considered controllable and consequently could be suspended and retrieved.

Above, we have shown how to control a system when the bad subinterval is at the beginning of the firing interval. What if the acceptable subinterval is at the beginning and bad subintervals are after? Consider the example of Figure 10. A controllable transition t_1 is associated with firing interval $[a, b]$. Suppose that, based on Algorithms 1 and 2, the subinterval $[a, \alpha_1]$ is acceptable while $]\alpha_1, b]$ is a bad subinterval, where $a \leq \alpha_1 < b$. The controlled time Petri nets are presented in Figure 11. Figure 12 represents the clock evaluation of t_1 and shows how the new interval associated with t_1 is calculated. In $[0, a[$, p_{susp} is marked and t_1 is neither active nor enabled. Meanwhile, at the time a, t_{r1} becomes enabled and is fired consuming the token in p_{susp}. Thus, right at a, the transition t_1 becomes active and its associated clock starts elapsing. Note that associated stopwatch delays t_1 for a time units and we do not want to delay it anymore; then, the lower bound of the new associated interval in controlled TPN is 0. After α_1 time units, t_{r2} is fired, and p_{susp} becomes marked again. Consequently, t_1 is suspended.

Consider the same example of Figure 10 and this time suppose that Algorithms 1 and 2 give the following output.

$[a, \alpha_1[$ is acceptable, $[\alpha_1, \alpha_2[$ is a bad subinterval, $[\alpha_2, b]$ is acceptable where the condition $a < \alpha_1 < \alpha_2 \leq b$ holds. With the same idea, the controlled Petri nets are presented in Figure 13 and the clock evaluation of t_1 is shown in Figure 14.

5.1. Why Inhibitor Hyperarcs? We have shown, how to achieve a controlled model from an uncontrolled time Petri nets by adding inhibitor hyperarcs. One question is why among different types of stopwatch inhibitor hyperarcs are chosen? Is it possible to use another stopwatch model? The answer is yes; but, inhibitor hyperarcs provide more flexibility and less complication. Let us see if this idea is feasible using other types of Petri nets with stopwatch.

We consider the post- and preinitialized Petri nets proposed in [4, 5] for stopwatch and try to control a model where a controllable transition associated with time interval $[a, b]$ has a bad subinterval $[a, \alpha]$. Based on our hypothesis, the controller should suspend the controllable transition at $[a, a]$ and resume it at $[\alpha, \alpha]$. The characteristic of post and preinitialized Petri nets is that they consume the same token of the original model and in addition, the exact time when the model becomes suspended is unknown as stopwatch transition and original transition have the same chance for firing. At the first glance, the idea is feasible by adding one place and two transitions. However, there are some issues. We explain the problem through an example.

We consider the same example of Figure 1. The suggested controlled model using post and pre-initialized Petri nets comes in Figure 15. The controllable transition t_1 is associated with stopwatch. The stopwatch interrupts the task at $[0, 0]$ and resume it at $[2, 2]$. The problem is at $[0, 0]$, both transitions t_1 and t_{sus} are enabled with equal chance of firing whereas, in order to have a controlled model, t_{sus} should fire at $[0, 0]$ before t_1. Thus, the controller fails unless if we modify the interval associated to t_1 to $]0, 2]$ which is not of interest in this approach.

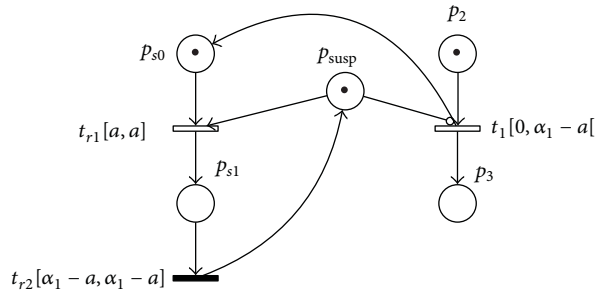

FIGURE 11: The controlled TPN of Figure 10 using inhibitor hyperarcs ($T_c = \{t_1\}$). Forbidden interval is $]\alpha_1, b]$ where $a \leq \alpha_1 < b$.

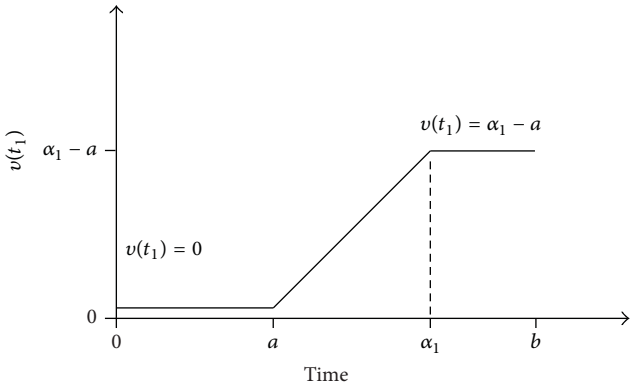

FIGURE 12: Clock evaluation of t_1 in the controlled TPN of Figure 10. Forbidden interval is $]\alpha_1, b]$ where $a \leq \alpha_1 < b$.

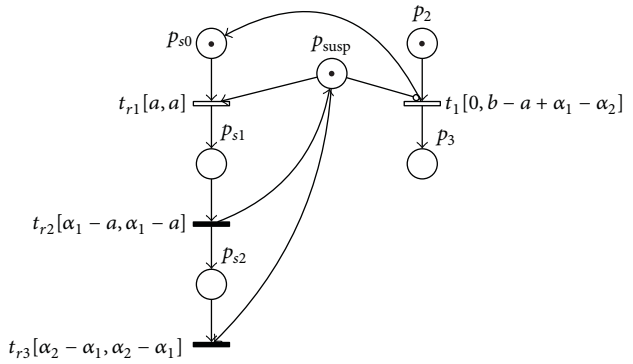

FIGURE 13: A simple time Petri net controlled by inhibitor hyperarcs ($T_c = \{t_1\}$); the bad subinterval is $[\alpha_1, \alpha_2[$ where $a < \alpha_1 < \alpha_2 \leq b$.

The other alternative is presented at Figure 16. An auxiliary place p_s solves the problem. Hence, using the stopwatch Petri nets suggested in [4, 5], it is not easy to give a general solution to control a model considering the output of Algorithms 1 and 2. In addition, the resulting controlled nets are more complicated. For example, in case the controllable transition is associated with the interval $[a, b]$ where its bad subinterval is $[\alpha, b]$ and $a \leq \alpha$ then, the controlled model becomes complicated.

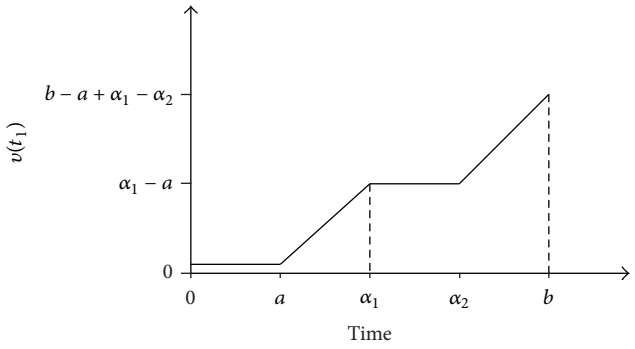

FIGURE 14: Clock evaluation of t_1 in a controlled TPN of Figure 10. The forbidden interval is $]\alpha_1, \alpha_2]$ where $a < \alpha_1 < \alpha_2 \leq b$.

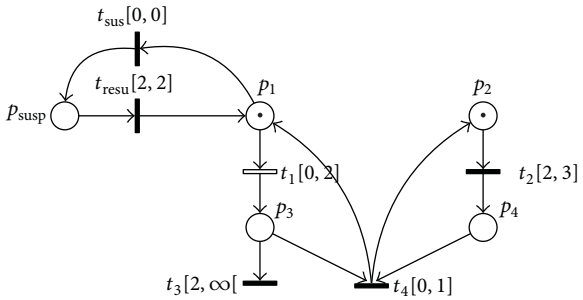

FIGURE 15: Controlling the example of Figure 1 using stopwatch of [4]. The controller fails.

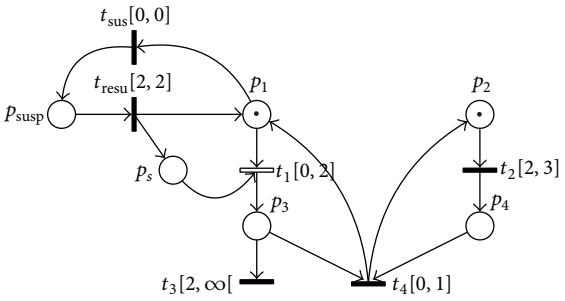

FIGURE 16: Controlled model of Figure 1 using stopwatch of [4].

6. Illustrative Example

Suppose the multitasking system depicted in Figure 17 with the following specifications: four tasks T_1, T_2, T_3, and T_4 are being executed. Two tasks T_3 and T_4 are uncontrollable. The task T_1 is periodic with a period of $[6, 6]$ and then, its start execution is controllable. Suppose that starting execution of T_2 is also controllable. T_4 is depended to T_2 and T_3 meaning that it can be executed only when T_2 and T_3 are executed. Two tasks T_1 and T_2 share a common resource. On the other hand, T_1 enters through a single capacity buffer. It means that if T_1 is not executed by the end of its period, the system will block. In Figure 7, tasks are modeled by tokens and the execution of each task T_i is shown by the transition t_i. If T_1 is not executed by the end of its period the system enters the state Block, and, t_5 is the transition to be avoided. Delay is a place modeling the

Table 2: State classes of the TPN presented at Figure 17.

α_0 : Delay + P_1 + Resource	$2 \le t_2 \le 3,\ 5 \le t_3 \le 6,\ 2 \le t_1 \le 2,\ 6 \le t_5 \le 6$
α_1 : Delay + P_1 + Wait	$3 \le t_3 \le 4,\ 4 \le t_5 \le 4$
α_2 : Delay + P_1 + Resource	$2 \le t_2 \le 3,\ 3 \le t_3 \le 4,\ 2 \le t_1 \le 2,\ 6 \le t_5 \le 6$
α_3 : Delay + P_3 + Wait	$0 \le t_4 \le 1,\ 0 \le t_5 \le 1$
α_4 : P_1 + Wait + Block	$0 \le t_3 \le 0$
α_5 : Delay + P_1 + Wait	$1 \le t_3 \le 2,\ 4 \le t_5 \le 4$
α_6 : Delay + P_1 + Resource	$2 \le t_2 \le 3,\ 1 \le t_3 \le 2,\ 2 \le t_1 \le 2,\ 6 \le t_5 \le 6$
α_7 : Delay + Resource + P_1	$2 \le t_2 \le 3,\ 5 \le t_3 \le 6,\ 2 \le t_1 \le 2,\ 0 \le t_5 \le 1$
α_8 : P_3 + Wait + Block	$0 \le t_4 \le 1$
α_9 : Delay + P_3 + Wait	$0 \le t_4 \le 1,\ 2 \le t_5 \le 3$
α_{10} : Delay + P_1 + Wait	$0 \le t_3 \le 0,\ 4 \le t_5 \le 4$
α_{11} : Delay + Resource + P_3	$0 \le t_2 \le 2,\ 0 \le t_1 \le 1,\ 4 \le t_5 \le 5,$ $0 \le t_2 - t_1 \le 1,\ -4 \le t_2 - t_5 \le -3,\ -4 \le t_1 - t_5 \le -4$
α_{12} : Delay + Resource + P_1	$2 \le t_2 \le 3,\ 0 \le t_3 \le 0,\ 2 \le t_1 \le 2,\ 6 \le t_5 \le 6$
α_{13} : Resource + P_1 + Block	$1 \le t_2 \le 3,\ 4 \le t_3 \le 6,\ -4 \le t_2 - t_3 \le -2$
α_{14} : Resource + P_1 + Block	$2 \le t_2 \le 3,\ 5 \le t_3 \le 6$
α_{15} : Delay + Resource + P_1	$2 \le t_2 \le 3,\ 5 \le t_3 \le 6,\ 2 \le t_1 \le 2,\ 1 \le t_5 \le 3$
α_{16} : Delay + P_3 + Wait	$0 \le t_4 \le 1,\ 4 \le t_5 \le 4$
α_{17} : Delay + Resource + P_3	$2 \le t_2 \le 3,\ 2 \le t_1 \le 2,\ 6 \le t_5 \le 6$
α_{18} : P_1 + Wait + Block	$2 \le t_3 \le 4$
α_{19} : Delay + P_1 + Wait	$3 \le t_3 \le 4,\ 0 \le t_5 \le 1$
α_{20} : Resource + P_1 + Block	$0 \le t_2 \le 2,\ 3 \le t_3 \le 5,\ -4 \le t_2 - t_3 \le -2$
α_{21} : Delay + Resource + P_1	$2 \le t_2 \le 3,\ 5 \le t_3 \le 6, 2 \le t_1 \le 2,\ 3 \le t_5 \le 4$
α_{22} : Delay + P_1 + Wait	$3 \le t_3 \le 4,\ 1 \le t_5 \le 2$
α_{23} : P_1 + Wait + Block	$1 \le t_3 \le 3$

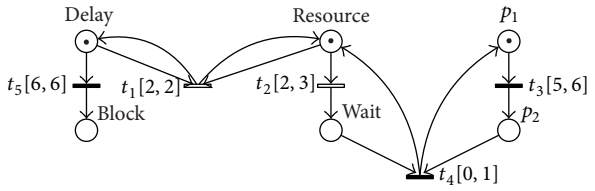

Figure 17: A periodic system with $T_c = \{t_1, t_2\}$.

behavior of the buffer considering the period of the task T_1. And finally, T_1 executes within $[2, 2]$ time units, T_2 in $[2, 3]$, T_3 within $[5, 6]$ and T_4 in $[0, 1]$.

The state class graph of the model is given in Figure 18 and Table 2 shows the state class informations. Based on the state class graph of the system, the state classes α_4, α_8, α_{13}, α_{14}, α_{18}, α_{20}, and α_{23} are forbidden. Let us briefly trace the algorithm on the state class graph. First, we follow the algorithm on the left branch where the state classes α_4, α_8, and α_{13} are located. The algorithm starts from α_0 and is executed recursively to α_1, then α_3, α_7 and finally reaches to α_{13} with a forbidden marking. Then, it comes back to α_7 with $\{t_5\}$ and to α_3 with $\{t_4t_5\}$. The other successor available from α_3 is α_8 which is also forbidden. Till now there is no enabled controllable transition to avoid these classes and the algorithm continues returning back to α_1 with $\{t_3t_4t_5, t_3t_5\}$. The other successor available from α_1 is α_4 which is also forbidden. Thus, it returns back to α_0 with $\{t_2t_3t_4t_5, t_2t_3t_5, t_2t_5\}$. The other successor available from α_0 is α_2 which is safe and the algorithm continues to α_5, α_9, α_{15}, α_{19}, and finally α_{18} which is a forbidden state. Then, it goes back to α_{19} with $\{t_5\}$ and consequently to α_{15} with $\{t_2t_5\}$. The other successors available from α_{15} are the forbidden state α_{20} and α_2 which is already under processing. Note that two enabled controllable transitions are available at α_{15}, but the controller cannot act at this level because t_5 may fire before t_1 or t_2 and lead to a forbidden state. Then, the algorithm continues and returns back to α_9 with $\{t_4t_2t_5, t_2t_5\}$ where no controllable transition is available. Consequently, the algorithm continues and returns back to α_5 and finally reaches to α_2 where newly enabled controllable transitions are available. Till now the path to be avoided at α_2 includes $\{t_2t_3t_4t_2t_5, t_2t_3t_4t_5\}$.

The procedure is similar for the other path available from α_2 including α_6 and its successors. The forbidden state reachable through this path is α_{23} and in order to avoid this forbidden state, the controller can act at α_{21} with two enabled controllable transitions. So, nothing is returned to α_2 from this path. Having a closer look to the state class graph and the output of the algorithm, we conclude that at each state where t_1 and t_2 are newly enabled and the controller should act (i.e., α_0, α_2, and α_{21}), the controller should force t_1 to fire before t_2 (i.e., $t_2 - t_1 > 0$). In other words, t_2 should not be active at $[2, 2]$. It is sufficient to add an inhibitor hyperarc to deactivate t_2 at $[2, 2]$. The controller cannot act at the

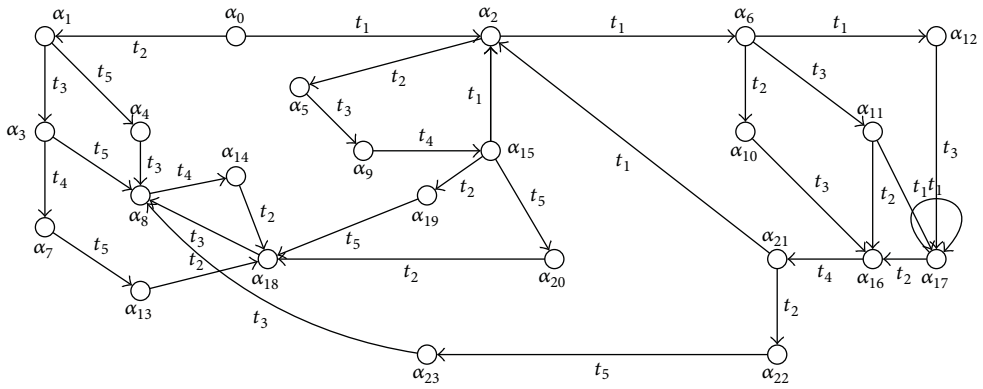

Figure 18: The state class graph of the TPN presented at Figure 17.

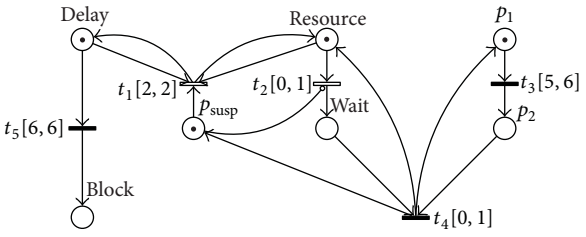

Figure 19: Controlled model of Figure 17 using inhibitor hyperarcs.

state class α_{15} as discussed earlier and has nothing to do at the state class α_6 (the permissive controller will act later at the state class α_{21}). Note that a state class like α_{14} which is reachable through a forbidden state (α_8) is not processed by the algorithm as it is supposed to be avoided in the controlled system. The controlled model using inhibitor hyperarcs is given at Figure 19.

Our model is now safe. Yet some challenges exist. Look at the state class graph of the model presented at Figure 18. In some paths one task is executed frequently while others are still waiting. For example, see the paths leading to α_{17}. Although the system is not blocked, some of the tasks cannot be executed. A good scheduler had better to perform different tasks alternatively. We may want to consider alternation, particular sequences, add a deadline for each task or other possible policies. And finally, we may ask if it is possible to restrict the execution time of a task even if it is in its safe subinterval? Then, while synthesizing a scheduler, we can add more constraints and scheduling policies beyond "safe states." Further researches are required to answer these questions.

7. Conclusion

In this paper, we have extended the approach discussed in [9] to time Petri nets with stopwatch and have suggested to associate the controllable transitions with stopwatch in order to prevent their bad subintervals. We have proposed that in case a controller (like a scheduler) cannot restrict time intervals associated with controllable transitions, the controllable transitions can be suspended in their bad subintervals. Synthesizing a time Petri net associated with stopwatch is complicated and needs some overapproximations. Besides, time Petri nets associated with stopwatch do not preserve the boundedness property. In this paper, we have suggested a more effective alternative. We synthesize a time Petri net without stopwatch and then, equip the controllable transitions with stopwatch where necessary. With this assumption, every controllable transition can be suspended in its bad subinterval and resumed otherwise. Amongst different types of stopwatch, our solution is based on inhibitor hyperarcs.

The approach suggested in this paper is particularly useful at design level for preemptive scheduling purposes, managing shared resources and critical sections. This is a good starting point for further researches on scheduling and to calculate automatically a scheduler in a multitasking system. It is interesting to see how the approach can be applied in a more general case with a variable number of tasks and dependencies. Consider different scheduling policies are challenging and are worth to be considered in further studies.

References

[1] P. Heidari, *A forward on-the-fly approach for safety and reachability controller synthesis of timed systems [Ph.D. thesis]*, École Polytechnique de Montréal, 2012.

[2] F. Cassez and K. Larsen, "The impressive power of stopwatches automata," in *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR '00)*, pp. 138–152, Springer, 2000.

[3] O. H. Roux and D. Lime, "Time petri nets with inhibitor hyperarcs. formal semantics and state space computation," in *Proceedings of the International Conference on Application and Theory of Petri Nets (ICATPN '04)*, pp. 371–390, 2004.

[4] A. Allahham and H. Alla, "Post and pre-initialized stopwatch Petri nets: formal semantics and state space computation," *Nonlinear Analysis*, vol. 2, no. 4, pp. 1175–1186, 2008.

[5] A. Allahham and H. Alla, "Réseaux de Petri à chronomètres post et pré-initialisés," in *Proceedings of the 6 ème Colloque Francophone sur la Modélisation des Systémes Réactifs (MSR '07)*, pp. 263–280, Lyon, France, 2007.

[6] D. Isovic and G. Fohler, "Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints," in

Proceedings of the 21st IEEE Real-Time Systems Symposium, pp. 207–216, 2000.

[7] K. Altisen, G. Gossler, and J. Sifakis, "A methodology for the construction of scheduled systems," in *Proceedings of the Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT '00)*, pp. 106–120, 2000.

[8] K. Altisen, G. Gossler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine, "A framework for scheduler synthesis," in *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99)*, pp. 154–163, December 1999.

[9] P. Heidari and H. Boucheneb, "Efficient method for checking the existence of a safety/ reachability controller for time petri nets," in *Proceedings of the 10th International Conference on Application of Concurrency to System Design (ACSD '10)*, pp. 201–210, June 2010.

[10] O.-H. Roux, D. Lime, S. Haddad, F. Cassez, and B. Brard, "Comparison of different semantics for time Petri nets," in *Proceedings of the 3rd Automated Technology for Verification and Analysis*, vol. 3707 of *Lecture Notes in Computer Science*, pp. 293–307, 2005.

[11] M. Boyer and F. Vernadat, "Language and bisimulation relations between subclasses of timed petri nets with strong timing semantic," Tech. Rep., LAAS, 2000.

[12] W. Penczek and A. Polrola, "Specification and model checking of temporal properties in time Petri nets and timed automata," in *Proceedings of the 25th International Conference on Application and Theory of Petri Nets*, vol. 3099 of *Lecture Notes in Computer Science*, pp. 37–76, 2004.

[13] B. Berthomieu and M. Diaz, "Modeling and verification of time dependent systems using time Petri nets," *IEEE Transactions on Software Engineering*, vol. 17, no. 3, pp. 259–273, 1991.

[14] H. Boucheneb, G. Gardey, and O. H. Roux, "TCTL model checking of time petri nets," *Journal of Logic and Computation*, vol. 19, no. 6, pp. 1509–1540, 2009.

[15] J. Bengtsson, *Clocks, DBMs and states in timed systems [Ph.D. thesis]*, Uppsala Universitet, Sweden, 2002.

[16] B. Berthomieu and F. Vernadat, "State class constructions for branching analysis of time Petri nets," in *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '03)*, pp. 442–457, 2003.

[17] D. Lime and O. Roux, "A translation based method for the timed analysis of scheduling extended time Petri nets," in *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS '04)*, pp. 187–196, December 2004.

[18] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario, "Timed state space analysis of real-time preemptive systems," *IEEE Transactions on Software Engineering*, vol. 30, no. 2, pp. 97–111, 2004.

[19] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "HYTECH: a model checker for hybrid systems," in *Proceedings of the 9th International Conference on Computer-Aided Verification (CAV '97)*, vol. 1254 of *Lecture Notes in Computer Science*, pp. 460–463, Springer, Berlin, Germany, 1997.

[20] P. Heidari and H. Boucheneb, "Maximally permissive controller synthesis for time petri nets," *International Journal of Control*, vol. 86, no. 3, pp. 493–511, 2013.