mathematics

MDPI

Article

Formal Modeling and Verification of Embedded Real-Time Systems: An Approach and Practical Tool Based on Constraint Time Petri Nets

Libero Nigro [1,*] and Franco Cicirelli [2]

[1] Engineering Department of Informatics Modelling Electronics and Systems Science, University of Calabria, 87036 Rende, Italy
[2] CNR—National Research Council of Italy, Institute for High Performance Computing and Networking (ICAR), 87036 Rende, Italy; f.cicirelli@icar.cnr.it
* Correspondence: libero.nigro@unical.it

Abstract: Modeling and verification of the correct behavior of embedded real-time systems with strict timing constraints is a well-known and important problem. Failing to fulfill a deadline in system operation can have severe consequences in the practical case. This paper proposes an approach to formal modeling and schedulability analysis. A novel extension of Petri Nets named Constraint Time Petri Nets (C-TPN) is developed, which enables the modeling of a collection of interdependent real-time tasks whose execution is constrained by the use of priority and shared resources like processors and memory data. A C-TPN model is reduced to a network of Timed Automata in the context of the popular Uppaal toolbox. Both functional and, most importantly, temporal properties can be assessed by exhaustive model checking and/or statistical model checking based on simulations. This paper first describes and motivates the proposed C-TPN modeling language and its formal semantics. Then, a Uppaal translation is shown. Finally, three models of embedded real-time systems are considered, and their properties are thoroughly verified.

Keywords: embedded real-time systems; timing constraints; schedulability analysis; formal modeling; high-level time Petri nets; timed automata; Uppaal; model checking; simulation

MSC: 68M20; 68Q10; 68Q60; 68Q85; 68U01

check for updates

Citation: Nigro, L.; Cicirelli, F. Formal Modeling and Verification of Embedded Real-Time Systems: An Approach and Practical Tool Based on Constraint Time Petri Nets. *Mathematics* **2024**, *12*, 812. https://doi.org/10.3390/math12060812

Academic Editor: Hsien-Chung Wu

Received: 17 February 2024
Revised: 6 March 2024
Accepted: 8 March 2024
Published: 10 March 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In the last years, the diffusion of embedded real-time systems [1] in such application domains as automobiles, communication networks, Internet-of-Thing devices [2] and so forth, has placed in great evidence the importance of assuring the logical and temporal correctness (fulfillment of deadlines or end-to-end delays) of these systems. Failing to guarantee that some tasks meet their strict timing constraints can have severe consequences in practical cases. Nowadays, there is a widespread consensus on the use of formal tools for modeling and property checking of real-time systems. A fundamental aspect concerns the modeling of *task preemption*, that is, the capability to interrupt a task execution, store its control status and replace the task with a different one whose execution is resumed from the point it was last left off or from its beginning. Task preemption is required to enforce an execution environment where, from time to time, only the most important next action, that is, the one having the highest priority, is permitted to progress. These issues are practically the responsibility of a real-time operating system (RTOS) [3], which can operate according to fixed, pre-runtime established priorities or with dynamically defined priorities, e.g., the Earliest Deadline First algorithm [3–5]. From the viewpoint of formal modeling, task preemption can be achieved by using *stopwatches* [6], that is, clocks whose advancement can temporarily be stopped and subsequently resumed. Stopwatches have been experimented

with both in extended Time Petri Nets (TPN)-based [7] tools, e.g., SETPN (Scheduling Extended TPN) [8], IHTPN (Inhibitor Hyperarcs TPN) [9], ROMEO [10], TINA [11,12], ORIS [13,14], ITPN (Inhibitor arc TPN) [15], and in Timed Automata (TA)-based [16] tools like Uppaal [17–21]. However, the use of stopwatches complicates the generation of the state execution graph and the analysis of a system model, which can become undecidable. A lot of work has been successfully carried out with the use of normal, not stoppable clocks in time-dependent models. In these cases, in fact, it is possible to build the state graph, e.g., of a TA model [17], by compact data structures representing state classes. A state class is a node composed of a data part that holds the valuation of data variables in a given execution state of the model, and a time part, which is a set of clock inequalities conjunctions (*zone*) that characterize, in a compact way, all the possible occurrence times for entering the particular execution state. It has been shown that, in many cases, although the use of a dense time model and thus of infinite occurrence times of state transitions, the state graph is finite and contains convex zones which enable, by exhaustive *model checking* [17], a rigorous assessment of both functional (e.g., reachability) and temporal (e.g., deadline fulfillment) properties, with the mediation of a proper temporal logic language for expressing the property queries to navigate the state graph. The fundamental consequence of using stopwatches in a model is that time zones are no longer convex and over-approximation techniques [6] can be introduced to permit property exploration. A non-convex zone is turned convex by expanding it, thus including time states which logically are not part of the original system model behavior. Despite the use of over-approximation, the modeler is normally capable of extracting important information, e.g., ensuring that safety properties hold, or predicting the Worst Case Response Time (WCRT) of real-time tasks, by model checking activities. In addition, the use of Statistical Model Checking [18,22], which does not build the state graph and instead uses simulations to infer properties, can significantly contribute to the analysis of a real-time system.

The main contribution of this paper concerns the proposal of a formal modeling and verification method for embedded real-time systems with timing constraints, which is based on a purposely developed extension of Time Petri Nets (TPN) [7], named Constraint Time Petri Nets (C-TPN), which is structurally reduced to the Timed Automata of the Uppaal toolbox [17,18]. The approach specifically addresses task preemptions and schedulability analysis of a modeled system, similar to existing tools based on TPN extended with stopwatches and preemption [8–15]. Differences introduced by the C-TPN/Uppaal tool from other known TPN-based tools can be characterized by first summarizing the main features of existing tools as follows:

- TPN-based tools normally depend on a special case implementation of the computational state graph based on the over-approximation, which in some cases can furnish more strict predictions of the WCRT of tasks.
- They enable the schedulability analysis of a real-time system according to fixed priorities only. In addition, static priorities of tasks can often be specified by using inhibitor arcs [9,15], which cause visual cluttering in modeling complex systems.
- Some tools [8,13,14] permit a source model to be split among multiple processors [23] and the internal control flow of tasks to be specified by detailing the mutex objects protecting shared data or the exchange of messages between tasks.

Distinguishing features introduced by the proposed C-TPN/Uppaal tool are highlighted below:

- The tool depends on the state graph built by Uppaal on a reduced TA model of a source C-TPN net. A reduced model, with over-approximation, can be thoroughly analyzed by exhaustive model checking and the overall schedulability rigorously assessed along with the WCRT of tasks. In addition, the Statistical Model Checker of Uppaal can be exploited for the graphical monitoring of particular executions of a real-time task set, for example, to observe the preemptions/suspensions and the subsequent resumes of specific task actions/transitions. All of this can help the modeler's reasoning and expectations about the behavior of a real-time model.

- Both statically and dynamically defined task priorities are supported. This way, more general schedulability schemes can be studied, including the EDF scheduling policy [3–5], which has been demonstrated to be capable of better exploitation of the computing resources and can be a choice of the RTOS managing a final model implementation.
- Similar to TPN-based tools in [8,13,14], both the internal control and data aspects of tasks can be specified, e.g., the mutex objects used to protect shared data or the messages exchanged between tasks. In addition, the partitioning of tasks among multiple processors is supported.

The power and flexibility of a C-TPN model derive from the introduction of high-level *constraint functions* attached to transitions. Constraint functions capture and hide model-wide conditions (e.g., scheduling issues like priorities), which can require the stopwatch associated with a transition to be temporarily stopped or its advancement resumed. As a side benefit, the use of constraint functions improves the readability of models and permits inhibitor arcs to be employed in the usual way for modeling purposes.

This paper is structured as follows. Section 2 describes the basic concepts, syntax and formal operational semantics of the C-TPN modeling language. Section 3 details a reduction of C-TPN models onto Uppaal Timed Automata. Section 4 shows the resultant approach in action through three task modeling examples, with increasing scheduling difficulties, and their in-depth schedulability verification. Finally, Section 5 presents conclusions together with an indication of ongoing and future work.

2. Constraint Time Petri Nets

This paper is focused on the use of Petri Nets [24] for modeling embedded real-time systems. A Petri Net (PN) model (see Figure 4) is made up of two kinds of subjects, places (represented by circles) and transitions (denoted by black bars), interconnected by weighted directed arcs. Input arcs link places to transitions. Output arcs connect transitions to places. Although the interpretation of a PN is ultimately a matter of the modeler's intents, places are often used as containers of resources (messages, memory data, processors and so forth). Transitions, instead, denote events or actions that may occur in the system model. Tokens (black balls or non-negative integers reported inside a place) are generic resources assigned to places. Their number changes during the evolution of the model. The number of tokens in a place denotes its local status and it is said to be its marking. The distribution of tokens, at a given moment, among the places of a model, represents the global status of the Petri Net; it is said the marking of the model that is often indicated by M. In the classic Petri Net formalism, transitions are viewed as untimed events. They can instantaneously and atomically occur (firing process) by consuming tokens from the input places (preset of the transition) and generating tokens in the output places (said postset of the transition). However, for a transition to possibly fire, it must satisfy the precondition of being enabled. A transition is said to be *enabled* if enough tokens are available in each place of the preset, as demanded by the weights of the input arcs.

The power and flexibility of Petri Nets are mainly due to their ability to represent concurrent/distributed systems with non-determinism (among multiple simultaneously enabled transitions, one is randomly selected for firing), conflicts (two or more transitions that share some input places in their presets can fire in such a way that one excludes, momentarily, the other to consuming shared scarce tokens), mutual exclusion (which is a consequence of a conflict), message passing features and so forth.

In the context of timed systems, Petri Nets are augmented by timing attributes (e.g., Time Petri Nets [7]), often in the form of a time interval $[a, b]$, $0 \leq a \leq b \leq \infty$, with a, b in Q+, associated with each transition [7]. The time quantities a and b are relative to the last instant that the transition became enabled. In this way, a firing duration is introduced for each transition, which at least evaluates to a and, at the maximum, is prolonged to b time units. Time progression is controlled by a local clock x[t] associated with the transition t, which is reset at the enabling instant of t and goes on to measure the time elapsed. Normally,

all the active clocks in a TPN model increase at the same rate. In the case that the clock x[t] can be temporarily stopped and subsequently resumed, as required for modeling task preemption in a real-time system, it becomes a stopwatch.

Specific technical issues of the proposed Constraint Time Petri Nets (C-TPN) formal modeling language are clarified in the following sections.

2.1. Syntax

The C-TPN modeling language extends the basic Merlin & Farber Time Petri Nets (TPN) formalism [7], which has been proven to be effective for the specification of real-time systems with timing constraints. The extension specifically addresses the modeling of embedded real-time systems in the presence of stopwatches and task preemption [8–15], and in particular of scheduling constraints (static or dynamic priorities associated with transitions), multiple processors, shared data and control aspects among tasks [8,13,14]. The support of dynamic priorities, though, is a distinguished feature of C-TPN.

In the following, N denotes the set of natural numbers $\{0, 1, 2, \ldots\}$. A C-TPN net model [24] is a tuple $\left\{P, T, F, W, M^0, EFT, LFT, C\right\}$ where

- P is a non-empty set of places, $P = \left\{p_1, p_2, \ldots, p_{|P|}\right\}$;
- T is a non-empty set of transitions, $T = \left\{t_1, t_2, \ldots, t_{|T|}\right\}$, $P \cap T = \emptyset$;
- F is the flow relation: $F \subseteq (PxT) U (TxP)$, that is, the set of possible directed arcs connecting input places to transitions and transitions to output places;
- W is the weighting function of arcs: $W : F \to N$;
- M^0 is the initial marking of the model: M^0: $P \to N$;
- EFT and LFT functions specify, respectively, the Earliest Firing Time and the Latest Firing Time statically associated with each transition [7]: $EFT : T \to N$, LFT: $T \to NU\{\infty\}$, $0 \leq EFT(t) \leq LFT(t) \leq \infty$;
- $C : T \to$ bool is the constraint function of a transition, false by default when not specified. A true value of $C(t)$ mirrors the fact that at a given moment, the transition t is constrained by the model state, and its clock must be stopped. Clock advancement is resumed as soon as $C(t)$ evaluates to false. The definition of $C(t)$ is model-dependent and purposely can express scheduling concerns like a static or dynamic priority assigned to the transition, along with the processor, on top of which the transition is assigned for execution (see later in this paper).

It is worth noting that the time intervals of transitions are assumed to be dense and of real values, although the bounds are supposed to be expressed by natural numbers only. The syntax of C-TPN makes it possible to also exploit in a model input inhibitor arcs (ending with a small circle as shown in Figure 12), which have a 0 weight. In a C-TPN model, an inhibitor arc contributes to transition enabling in the usual way of a negative test: checking that an input place has no token. For simplicity, normal arcs with default unitary weight will be drawn without an explicit weight indication. The preset (input places) and postset (output places) of a transition $t \in T$ will be denoted, respectively, as ${}^\bullet t$ and $t^\bullet$.

2.2. Formal Operational Semantics

The formal semantics of a C-TPN model can be provided by a Labelled Timed Transition System (LTTS): $\{S, s0, \to\}$ where S is a set of states, $s0$ is the initial state and $\to$ is a state transition relation [25]. States of a C-TPN model mainly relate to its marking and the states of the transitions and their stopwatches. The following are some basic helper functions. The first one is a boolean function with no effect on the state of the net, the others are void functions governing the firing process of a transition.

$$\text{enabled}(t, M): \ \forall p \in {}^\bullet t: \ (M(p) \geq W(p,t) \wedge W(p,t) > 0) \vee (M(p) = 0 \wedge W(p,t) = 0)$$

$$\text{withdraw}(t, M, M') : \forall p \in {}^\bullet t: \ M'(p) = M(p) - W(p,t)$$

$$\text{deposit}(t, M', M''): \ \forall p \in t^{\bullet}: \ M''(p) = M'(p) + W(t, p)$$

Although transition firing is an atomic process, the above functions help clarify that the firing process correctly consists of two sub-steps: the withdrawal and the deposit steps. These distinct sub-steps are necessary because a place can be used both as an input and as an output place, and a transition can lose its enabling, due to a conflict, immediately after the withdraw sub-step, and then re-gain the enabling following the deposit sub-step, but in these cases, its clock must be correctly reset after the withdraw. Other fundamental Boolean functions follow:

$$\text{activated}(t, M) : \text{enabled}(t, M) \text{ and not constrained}(t, M)$$

$$\text{suspended}(t, M) : \text{enabled}(t, M) \text{ and constrained}(t, M)$$

The constrained(t, M) function evaluates the constraint function $C(t)$ of the transition t in the current marking M. The only difference between the activated and suspended state of an enabled transition is the management of the underlying clock, which in the first case can progress, whereas in the second case, it is temporarily blocked. For a disabled transition, the underlying clock plays no role, but it could also be reset and frozen to 0 for convenience.

Each possible execution state *s* has the following component sets:

$eT(M) = \{t \in T : \text{enabled}(t, M)\}$ is the set of enabled transitions in marking M;
$aT(M) = \{t \in T : \text{activated}(t, M)\}$ is the set of active transitions in marking M;
$sT(M) = \{t \in T : \text{suspended}(t, M)\}$ is the set of suspended transitions in marking M;
$fT(M) = \{t \in aT(M) : EFT(t) \leq x(t) \leq LFT(t)\}$ is the set of fireable transitions in marking M.

Among the fireable set $fT(M)$, one transition can be chosen non-deterministically for completing its firing. In addition, the *single-server* semantics is assumed. In other words, each transition consumes its enabling one at a time, sequentially. It should be noted that a fireable transition can actually complete its firing at any time instant of its interval $[EFT(t), LFT(t)]$ and then also at, but never beyond, its last $LFT(t)$, provided the transition remains continually enabled. Of course, a transition t can lose its enabling for a conflict, or it can become suspended at any time from the enabling instant until its last firing time: $x[t] \leq LFT(t)$.

The initial state $s0$ directly mirrors the consequences of establishing the initial marking M^0 in the C-TPN model. Moreover, in particular, the initial value of $eT\left(M^0\right)$, $aT\left(M^0\right)$, $sT\left(M^0\right)$ and $fT\left(M^0\right)$ are computed. From then on, two possible transitions $\rightarrow$ can be executed, thus moving the LTTS from one state to the next one. The first transition is the *delay transition* $\xrightarrow{d}$, which advances the system time, and then the clocks of active transitions in $aT(M)$ of a permitted amount of d time units. Executing this transition, the marking M of the model remains unchanged. The second transition $\xrightarrow{t}$ is the instantaneous *action transition* that completes the firing of a C-TPN transition t, and stores/updates its effects on the state component sets to reflect the marking changes in the net model. A definition of the two transitions follows.

2.3. Delay Transition $\xrightarrow{d}$

A first change in the system state can occur by the advancement of time only, without modifying the net marking M. If Δ is the minimal value between the minimal time duration required by the clock x(t) of any active transition t for reaching its $LFT(t)$ (that is: $\min_{t \in fT(M)}\{LFT(t) - x(t)\}$), and the minimal time duration that is needed by the clock of an active but not yet fireable transition t to reach its earliest time $EFT(t)$ (that is: $\min_{t \in aT(M) \setminus fT(M)}\{EFT(t) - x(t)\}$), then a delay transition can occur whose effects cause any active clock to be advanced of an amount d. This amount d is chosen non-deterministically

between 0 and Δ. More formally (the formal notation specifies the transition, its effects and its precondition as: $\frac{transition}{effects}$ *precondition*):

Let Δ be the $\min\Big\{\min_{t\in fT(M)}\{LFT(t)-x(t)\}, \min_{t\in aT(M)\setminus fT(M)}\{EFT(t)-x(t)\}\Big\}$ we have that

$$\frac{M \xrightarrow{d} M}{\forall t \in aT(M) : x(t) \leftarrow x(t)+d} \; \Delta > 0 \;\&\&\; 0 < d \leq \Delta$$

It is worthy of note that a delayed transition cannot occur if $\Delta = 0$. This situation happens, e.g., when a fireable C-TPN transition t arrives at its last time: $x[t] = LFT(t)$. Pseudo-advancements of 0 time units (even an infinite number of times) simply are not allowed (i.e., Zeno cycles are excluded [26]). In these cases, provided the C-TPN transition t continues to be enabled, it *must* complete its firing with a $\xrightarrow{t}$ action transition, thus ensuring the *strong-firing semantic model* of Merlin & Farber Time Petri Nets [7].

2.4. Action Transition $\xrightarrow{t}$

Provided one or more fireable transitions exist in the current system state, one of them is chosen non-deterministically and its firing process is completed by executing the two atomic sub-steps of withdrawal and deposit of the involved tokens. In formal terms:

$$\frac{M \xrightarrow{t} M''}{\begin{array}{c}\text{withdraw}(t, M, M') : \;\; \text{update state component sets accordingly and} \\ \text{reset clock of any newly enabled transition in } M'' \\ \text{deposit}(t, M', M'') : \;\; \text{update state component sets accordingly and} \\ \text{reset clock of any newly enabled transition in } M''\end{array}} \exists t \in fT(M)$$

A key point of an action transition $\xrightarrow{t}$ is the fact that its effects modify the current marking from M to M'' by applying the withdrawal and the deposit functions separately. As a consequence, the $eT(.)$, $aT(.)$, $sT(.)$ and $fT(.)$ sets are changed accordingly.

A transition $t_{p,}$ which was enabled in M and rests enabled in the intermediate marking M' and in the final reached marking M'', is said to be *persistent* to the firing of t. A transition t_{ne}, which was disabled in M and which gains its enabling in M' or M'', or which was enabled in M, loses its enabling in M' but reacquires the enabling in M'', is said to be *newly enabled*. For any newly enabled transition t_{ne}, its clock is reset: $x[t_{ne}] = 0$. The clocks of persisted transitions, instead, remain unchanged.

As a final remark, it should be noted that there are situations when the two kinds of transitions $\xrightarrow{d}$ and $\xrightarrow{t}$ are both eligible to occur. In these cases, one of the two is supposed to be chosen non-deterministically and accomplished. All of this corresponds to the fact that an active and fireable transition can complete its firing either at the current time or at a subsequent time instant belonging to its time interval.

3. Reducing C-TPN onto Uppaal

A C-TPN model can directly be mapped onto the Uppaal toolbox [17,18,27]. Uppaal, in fact, exposes powerful modeling abstractions that can be exploited to reproduce the operational semantics of an extended timed Petri Net formalism [20,28].

Uppaal is based on Timed Automata (TA) [16] augmented with high-level data variables, data structures (structs and arrays) and C-like functions, which greatly improve the modeling capabilities. A basic model consists of a network of concurrent TA (instantiable template processes), interacting with each other by unicast channels with rendezvous synchronization: the sender (!) or the receiver (?) who first arrives at the appointment waits for the arrival of the partner. Following the synchronization, both processes resume their concurrent execution. Broadcast channels are also supported, which allow one sender to synchronize with 0, 1 or multiple receivers. Broadcast synchronization is always asynchronous for the sender. The sender never waits for receivers to be ready to interact. Both

kinds of channels do not carry data. The transport of parameters, if there are any, can be realized through the mediation of some global data variables. Each process is modeled as an automaton with *locations* and *edges*. In a normal location (see the N location in Figure 2), the process can remain an arbitrary amount of time. An *invariant* (logical condition) can be attached to a normal location (see location A in Figure 2) to force it to be exited, after a finite time, before the invariant (based on time and/or data value constraints) is up to be falsified. Committed locations (see, e.g., the W and D locations in Figure 2) have a higher priority with respect to the non-committed ones and must immediately be exited without any time passage. An edge can be annotated by a *command* that is composed of three optional attributes: a conditional *guard* (true by default), a *channel synchronization* and an *update* (list of variable assignments or clock resets). A command constitutes the unit atomic action of the concurrent language of Uppaal. The automaton evolves by executing one command at a time, which also includes the case of two (or more) joint commands, in distinct processes, linked by a channel synchronization. The time passage rests on the use of *clock* variables. A clock measures the time elapsed from its last reset. All the clocks grow at the same pace. As an extension, U-MC allows clocks in a model to be temporarily stopped and subsequently resumed [6] (stopwatches).

From a basic TA model, the state graph (with convex zones) is first built upon which a TCTL (Timed Computational Tree logic) [17] formula/query can be verified by the non-deterministic model checker (verifyta), here denoted as U-MC, which analyzes all the possible execution paths (command inter-leavings) and, if required, can generate a counter example (diagnostic trace) to help the modeler detect, through an event trace, the reasons why a query was not satisfied.

The use of U-MC is limited in practice by the state explosions problem, which arises when handling complex models that can imply an enormous or infinite state graph.

In recent years, the Uppaal toolbox was enriched by a Statistical Model Checker (U-SMC) [18,22], too, which avoids the construction of the state graph and checks properties by simulations. A simple example is represented by Monte Carlo simulations used to estimate the occurrence probability of an event by counting the number of times the event occurs in a certain number of runs (the number of runs can be inferred by U-SMC). U-SMC enables the use of double quantities and restricts synchronizations to be based on broadcast channels only. Property queries can be expressed using the MITL (Metric Interval Temporal Logic) [18] language. By using U-SMC, TA are interpreted as stochastic processes. A further extension of U-SMC enables TA to model also hybrid processes, where the rate of clocks can be varied and continuous variables [29] can be specified by first-order differential equations, used as invariants of locations.

The use of stopwatches implies that U-MC generates a state graph whose zones depend on the over-approximation [6]. As a consequence, a zone can be enlarged (see dashed lines in Figure 1 taken from [30]) for it to become convex:



Figure 1. Example of zone convex hull for over-approximation [30].

Simply stated, the use of the over-approximation makes a satisfied existential query, which is one that checks if some state exists where a given property holds, to be inconclusive. In fact, the property could be satisfied in some of the extra states added by the convex hull. On the other hand, queries of the type "is it always true that", which assesses safety property holds in all the model states, if verified, necessarily hold also in the proper states of the model [6].

3.1. Data Structures

A C-TPN model is reduced onto Uppaal by first preparing the following global data structures:

- The B(ackward TxPRE $\rightarrow$ Info matrix, used for checking transition enabling and for assisting the withdrawal sub-step during transition firing;
- The F(orward TxPOST $\rightarrow$ Info matrix, which supports the deposit sub-step of transition firing;
- The I(nterval Tx2 $\rightarrow$ int matrix, which holds the EFT/LFT of transitions, and the int const INF (e.g., -1) is used to specify ∞ as LFT;
- The M(arking P $\rightarrow$ N vector, which holds the net marking.

P and T denote, respectively, the number of places and the number of transitions used in a C-TPN model. A fundamental sub-range type (typedef) tid is introduced

$$\text{typedef int } [0, \text{T} - 1] \text{ tid}$$

whose values correspond one to one to the transition names. PRE and POST denote, respectively, the maximum degree of a transition preset and postset in a model. Info is a *struct* that holds a place id and its corresponding weight. B, F and I are purposely implemented as constant data structures so as for them to not take part in the state graph nodes. The marking vector M, of course, changes its content during model evolution. Its declaration

$$\text{int}[0, \text{b}]\ \text{M}[\text{P}]$$

is directly related to the boundedness degree b of the C-TPN model.

Another fundamental global array is x[T], which associates a distinct clock to each separate transition:

$$\text{clock x}[\text{T}]$$

3.2. Transition Automata

The basic Transition template process (automaton) is shown in Figure 2. The automaton has one single parameter (const tid t), which provides the unique id (t) to each transition instance.

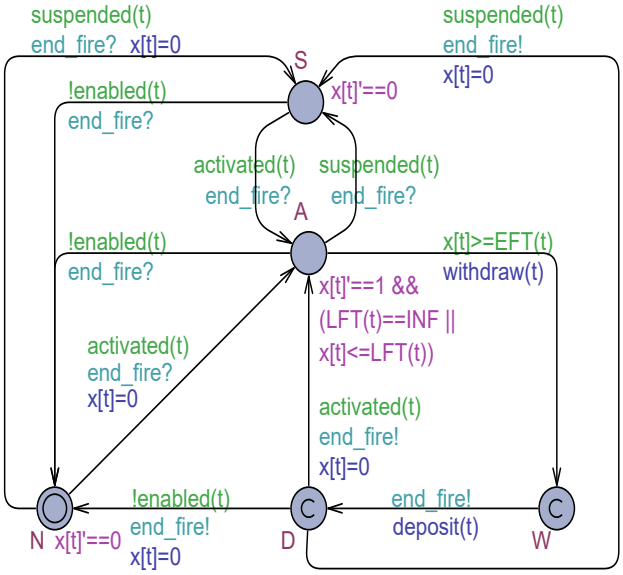


Figure 2. The *Transition* automaton, having a single parameter: const tid t.

What contributes to the compactness of Figure 2 is the use of helper functions, which directly correspond to the C-TPN fundamental functions discussed in Section 2.2. Here, for

simplicity, each function implicitly refers to the global marking vector M of the model. The meaning of the locations in Figure 2 is as follows:

- N, transition t is not enabled;
- A, transition t is active, that is, it is enabled and unconstrained;
- S, transition t is suspended, that is, it is enabled but constrained;
- W, transition t has started its firing by just executing the withdrawal sub-step;
- D, transition t is terminating its firing by executing its deposit sub-step.

As one can see in Figure 2, in the A location, the clock x[t] of this transition is allowed to grow by specifying, in the invariant of the location, that its first derivative is 1: x[t]′ == 1. Similarly, in the S location, the clock x[t] is stopped by putting its first derivative to 0: x[t]′ == 0. The full invariant of the A location states that the transition can remain there provided the clock x[t] is within the interval [EFT(t),LFT(t)]. In the case that the upper bound is infinite, the transition can stay in A for an arbitrary amount of time. The EFT(t) and LFT(t) functions simply return the interval bounds of t stored in the matrix I. The blocking of x[t] in N is a matter of simplicity, to help the visualization of results during the use of U-SMC (see later in this paper).

The firing process of a transition t is ensured using the global shared broadcast channel end_fire. In particular, the transition that is completing its firing sends in sequence two synchronizations over the end_fire channel. The first one is following the withdrawal sub-step and entering the W location. The second one is when exiting the D location, that is, after the deposit sub-step was executed. All the remaining transition instances in the model can hear each end_fire signal and adjust their state accordingly. For instance, a transition in N that finds itself (newly) enabled can reach either A or S depending on its constraint function. In both of the two cases, its clock is reset as required. But a transition in A or S can come back to N as soon as it becomes disabled, or it can move from A to S, or from S to A if the transition, respectively, is suspended or activated by its constraints. Finally, the transition that is completing its firing can move from D to A or S, or from D to N if (according to single-server semantics) it is still enabled after the current firing, or if it finds itself disabled.

A critical point in Figure 2 is the enforcement of the atomicity to the transition firing. Toward this, the two locations W and D were realized as committed locations, which are exited without time passage and certainly *before* any other transition can start its own firing process. For instance, Uppaal guarantees that even a transition t′ in A whose clock arrived at its last time LFT(t′) and, therefore, its invariant is up to be falsified would the time advance even of an ε quantity, and has less urgency than the exiting from a committed location, which does not increase the model time.

It should be noted that any transition can receive (?) an end_fire synchronization, provided it can execute a command with a satisfied guard. Would the guard be false, the transition is not a receiver of the end_fire signal.

3.3. Model Initialization

The simple Starter automaton shown in Figure 3 is responsible for the initialization of the C-TPN model reduced in Uppaal. M0() is a function that establishes the initial marking of the model. After that, an end_fire signal is broadcasted to permit transitions that are enabled in the initial marking to reach their A or S location. Only one end_fire synchronization is sufficient to bootstrap the model. Finally, Starter moves to a normal location and no longer interacts with the rest of the model.

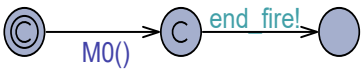


Figure 3. The *Starter* automaton for model initialization.

3.4. System Declaration

Each Uppaal model must specify, in the system declaration section, how many instances of the processes/automata have to be created and assign a name to each instance. A C-TPN model can easily be created by the following statement:

system Transition, Starter;

The above declaration will automatically create T instances of the Transition automaton and one instance of the Starter automaton. Due to the use of the typedef tid, each instance of Transition is uniquely identified as Transition(id), where id is a transition identifier. As a good practice, each C-TPN reduced model can anticipate in the global declarations, the constant ids (high-level names) for the places (from 0 to $P-1$) and for the transitions (from 0 to $T-1$). This simple provision will facilitate the expression and interpretation of the property queries issued by the modeler. As a final remark, it is important to note that a C-TPN reduced model can be checked, without changes, by either the U-MC and/or the U-SMC. What can imply U-SMC to be used exclusively is when double quantities (e.g., clocks values) are employed/checked in the constraint functions (see later in this paper).

4. Modeling and Verification Examples

Three real-time task models, with increasing use of modeling features and increasing scheduling difficulties, will be presented and thoroughly studied to check the effectiveness of the reduction of C-TPN onto Uppaal. As observed, e.g., in [15], due to the over-approximation, the timing results generated by different tools cannot be expected to coincide with one another. The experimental work was carried out by using the latest version Uppaal 5.0.0, 64 bit, running on a Win11 Pro desktop platform, Dell XPS 8940, Intel i7-10700 (8 physical cores), CPU@2.90 GHz, 32GB RAM.

4.1. First Example

The three-task model shown in Figure 4 was derived from [15] where it was expressed using the Inhibitor Time Petri Nets (ITPN) formalism, here rewritten according to the more readable C-TPN language. Transitions t1, t2 and t3 are *activation transitions*, often regulated by periodic behavior. Such transitions with an empty preset are always enabled. Each time an activation transition fires, a new instance of the corresponding task is generated (released). Tokens in the places p1, p2 and p3 denote the pending task instances to be completed. Generally speaking, each such place should hold one token only (except for some limit cases, which will be discussed), which should be consumed before the task activation transition injects a new task instance. For simplicity, the deadline of each task coincides with the task period. Therefore, to be schedulable, a task instance should never be consumed (processed) beyond its deadline. Transitions t4, t5 and t6 model the *task bodies* and their *computation times*. Despite its simplicity, the task set of Figure 4 cannot be analyzed according to the classic real-time scheduling theory [3–5]. This is because the t2 activation transition and all the task body transitions have non-deterministic dense temporal intervals. In the worst situation, all three tasks are supposed to be ready to execute at time 0 (no phasing). All of this is mirrored by the initial marking of the model in Figure 4 where one token is pre-assigned to p1, p2 and p3. The three-task model executes on a single *cpu* (default hypothesis).

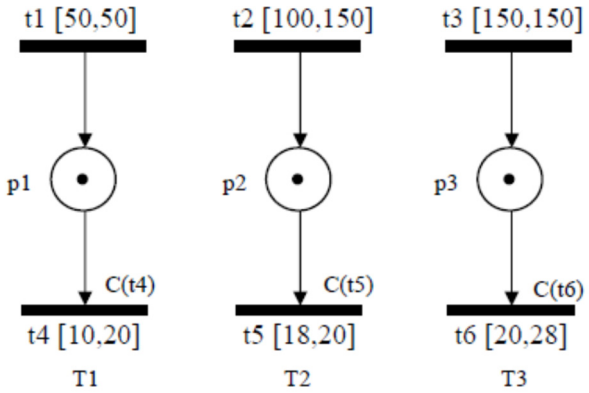


Figure 4. A three-task set C-TPN model.

4.1.1. Schedulability Analysis under Fixed Priorities (FP)

Each task is statically assigned a priority inversely proportional to the task period, as required by the Rate Monotonic method [3]. Therefore: $\pi(T1) > \pi(T2) > \pi(T3)$. At any instant, the active task transition can execute which has the greatest priority. All of this is clarified by the constraint functions shown in Table 1.

Table 1. The constraint functions of t4, t5, t6 of the model in Figure 4 under FP scheduling.

Transition	Constraint Function C(t)
t4	false
t5	M[p1]>0
t6	M[p1]+M[p2]>0

Obviously, activation transitions are never constrained. Transition t4 is unconstrained because, when active, it has the highest priority. On the other hand, transition t5 is constrained (and then suspended) as long as t4 is active (one token is available in p1). Similarly, transition t6 is constrained as long as t4 and/or t5 are active (one token is present in p1 and/or p2). If a constraint is not satisfied, the constraint evaluates to false and the transition is active. It is worth recalling that the constraint functions in Table 1, are checked each time a transition is enabled and the model marking changes, which would imply the transition is activated or suspended. For simplicity, all the transition constraints are collected in the Uppaal model, in a global function named constrained(t).

The first concern about schedulability is assessing that the model is bounded. In particular, the schedulability is guaranteed provided the model is 1-bounded, which means that, at each time, at most one instance exists of every task that is processed before the deadline. The declaration:

int[0,1] M[P]

expresses the model is 1-bounded. Would the model checker try to assign a new token to any place p already containing a token, an out-of-range assignment is generated in M[p], which witnesses the model is not 1-bounded. The first TCTL query [17] launched on the three-task model was

A[] !deadlock

which checks if it is always true (invariantly) that in any state of the state graph, there is no deadlock. This query also helps to assess whether the model is bounded. It was found that p3 (the lower priority task) is bounded to 2, whereas the other places are 1-bounded. Changing the declaration to int[0,2] M[P], the following queries were found satisfied:

A[] M[p1]<=1

A[] M[p2]<=1

A[] M[p3]<=2

and the previous query A[] !deadlock is also satisfied. Therefore, tasks T1 and T2 never generate a new instance before the previous one is completed within the deadline. The schedulability of task T3 instead deserves to be better investigated. The fact that M[p2] can have 2 tokens does not necessarily imply that T3 is not schedulable. What can happen is that a previous instance of the task completes its execution exactly *at* the deadline and that the new one will be processed in the newly started period. It should be observed that when t3 fires, it finds itself again enabled and its clock x[t3] is immediately reset (see transitions from D to A or S in Figure 2). The *leads-to* (-->) operator [17] was used in the following query (satisfied):

M[p3]==2 --> M[p3]==1 && x[t3]==0

to check that starting from a state where M[p3]==2, it always (inevitably) follows that M[p3] goes down to 1 and x[t3] is 0. In other terms, it is ensured that the pending instance of T3 is completed exactly after 150 time units (i.e., the deadline or period of T3) starting from its instantiation, without violating the deadline, which would otherwise happen in the case M[p3] went from 2 to 1 by consuming time units of the new period. The same leads-to query was used to check functional correctness, that is, model liveness:

M[p1]>0 --> Transition(t4).W

and, similarly, for t5 and t6 starting, respectively, from M[p2]>0 and M[p3]>0. These queries were satisfied, and guarantee, for example, that each time a token is in p1, transition t4 eventually fires. The Worst Case Response Time (WCRT) and the Best Case Response Time (BCRT) of each task can be determined by finding the worst/best time that elapses before a task body transition fires. The elapsed times can be detected by checking the clock values of the activation transitions. For the T1 tasks, the following queries were used:

A[] Transition(t4).W imply x[t1]<=20

A[] Transition(t4).W imply x[t1]>=10

The two queries check if it is always true that when t4 fires (i.e., it is in the W location of the automaton of Figure 2), the task clock x[t1] is lesser than or equal to 20 and greater than or equal to 10. Since task T1 has the highest priority, it is expected that the response times of the task coincide with the bounds of the t4 time interval. Both queries are satisfied. It should be noted that the first query, which establishes the WCRT, does not necessarily hold with bound 19, and the second query, which provides the BCRT, is not necessarily true with bound 11. The BCRT/WCRT could also be detected by using the suprema (sup) and infima (inf) queries of the TCTL:

sup{ Transition(t4).W } : x[t1]

inf{ Transition(t4).W } : x[t1]

which ask to detect, respectively, the maximum/minimum value of x[t1] in the system state where Transition(t4) fires. Table 2 collects the BCRT/WCRT for the three tasks.

Table 2. The response times of the model in Figure 4 under FP.

Task	BCRT	WCRT
T1	10	20
T2	18	60
T3	30	150

The BCRT of the task T3 was determined through the query:

A[] Transition(t6).W && M[p3]==0 imply x[t3]>=30

This is because when M[p3]=2, this happens at the end of the period and x[t3] has been just reset. As a consequence, the lowest value of x[t3] when t6 fires would be (incorrectly) 0. The following query could be used as well:

inf{ Transition(t6).W && M[p3]==0 } : x[t3]

All the above verification work was carried out by using U-MC and the exhaustive model checker. The scheduling can also be investigated graphically by showing some particular task executions that U-SMC can generate. The simulated query with a time limit set to (about) the hyper-period (the hyper-period is the least common multiple of the periods of the three tasks.) of the various tasks is suited for the purpose of

simulate[<=300] {x[t4],Transition(t4).S,Transition(t4).A}

The simulated query accumulates information, in the chosen time limit, about the value of x[t4] and the status of the transition t4. At the query termination, right clicking on the query line allows U-SMC to visualize the registered data as shown in Figure 5. It is confirmed that task T1, having the highest priority, does not experiment with any suspension or preemption. To avoid showing the values of the clock x[t4] when the transition t4 is disabled, the clock was purposely blocked in the N location (see Figure 2).

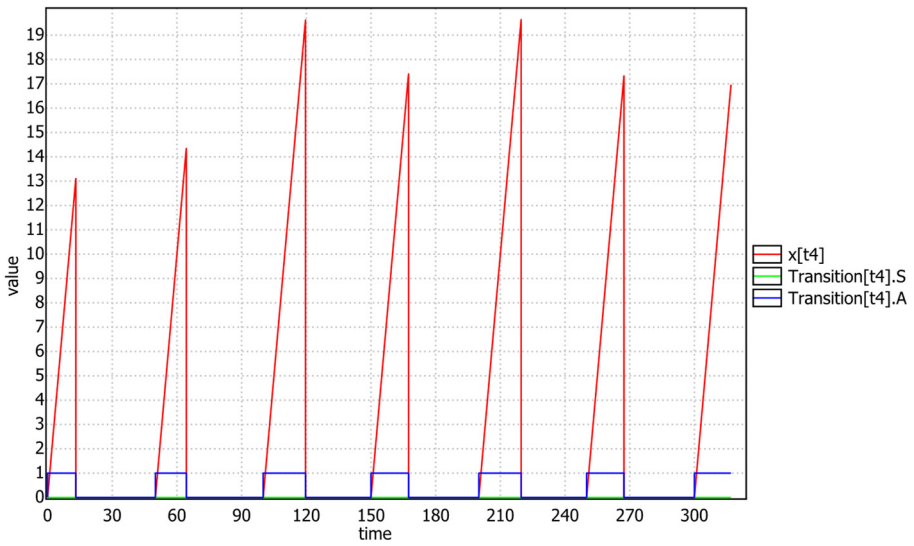


Figure 5. Highest-priority task T1 behavior observed by monitoring the t4 status and its clock.

Figure 6 reports the captured snapshot of t6 executions:

simulate[<=300] {x[t6],Transition(t6).S,Transition(t6).A}

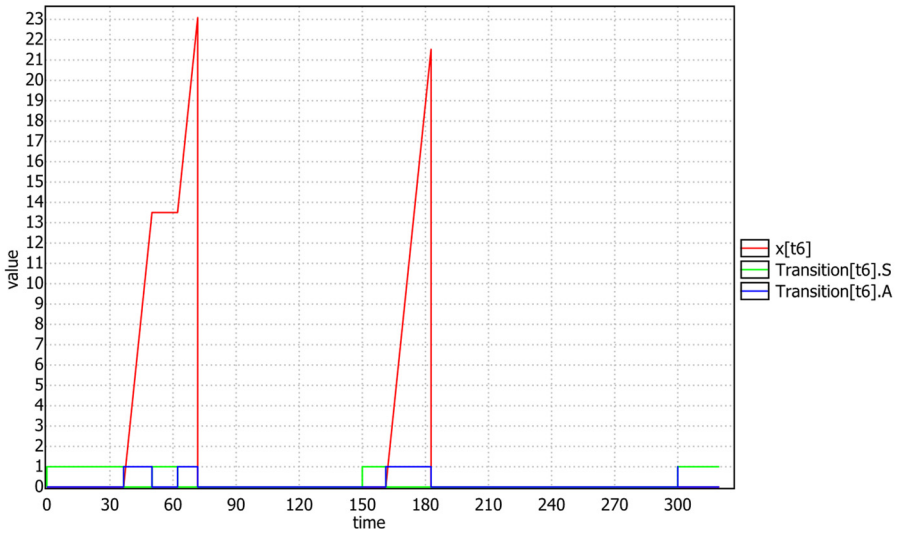


Figure 6. A snapshot of the lowest-priority T3 task t6 transition with suspension and preemption.

From Figure 6, one can see that the transition t6 is suspended following its enabling and that even when active it is suspended because of preemption. A similar behavior was observed for the task T2 and the transition t5.

In light of the experimental work under fixed priorities, it emerged that some timing results in [15] seem stricter than those achieved with Uppaal. However, the proposed tool was in any case capable of correctly predicting the schedulability. In addition, valuable information about task executions can be generated by U-SMC. Further flexibility of the C-TPN/Uppaal tool is demonstrated in the next section. As a final remark, it is worthy of note that the developed reduction in Uppaal allows the modeler to check the properties of a C-TPN source model by using exactly the C-TPN vocabulary: place and transition names, marking of specific places and so forth.

4.1.2. Schedulability Analysis under Dynamic Priorities

The task set of Figure 4 was also studied by using the Earliest Deadline First (EDF) scheduling strategy on a single *cpu*. It is known [3] that EDF is capable of better exploiting computational resources, but it is more challenging to analyze in a real-time tasking model. EDF was chosen here to demonstrate the flexibility and modeling power of the C-TPN/Uppaal proposed tool. Under EDF, at any time, the active task with a minimal distance from its deadline is given the highest priority and occupies the *cpu*. The same C-TPN reduced model into Uppaal was exploited, but the constraint functions of t4, t5 and t6 were changed according to the EDF policy, as shown in Table 3. A task transition is constrained (its clock stopped) provided a different task exists that is active and has a closer deadline. Otherwise, the task is unconstrained and can continue its execution.

Table 3. The constraint functions of t4, t5, t6 of the model in Figure 4 under EDF scheduling.

Transition	Constraint Function C(t)
t4	M[p2]>0 && 100-x[t3]<50-x[t1] || M[p3]>0 && 150-x[t5]<50-x[t1]
t5	M[p1]>0 && 50-x[t1]<100-x[t3] || M[p3]>0 && 150-x[t5]<100-x[t3]
t6	M[p1]>0 && 50-x[t1]<150-x[t5] || M[p2]>0 && 100-x[t3]<150-x[t5]

Since the use of floating-point operations in the constraint functions, U-SMC has to be used to estimate the model properties. Under EDF, the model was found to be 1-bounded. By using temporarily the declaration of int[0,2] for M[P], the following MITL query was issued:

Pr[<=300](<>M[p1]>1 || M[p2]>1 || M[p3]>1)

The query asks U-SMC to estimate the probability of the event that "*a model state exists in which p1 or p2 or p3 contain more than 1 token*". U-SMC, after 29 runs/simulations, where each run lasts 300 time units, proposes a confidence interval of [0,0.0981446] with a confidence degree of 95%, which indicates the event is practically impossible. On the other hand, by restricting each element of M[P] to an int[0,1], all the subsequent verification work never raised an assignment violation to M[.]. The WCRT of the task T1 was checked by the following query:

Pr[<=300] (<>Transition(t4).W && x[t1]>39)

which requires quantifying the probability of the event that "*a model state exists where the t4 transition fires and the task clock x[t1] is found to be greater than 39*". Such an event, after 54 runs, has an almost 0 probability. However, the same query with the condition x[t1]>38 proposes a non-zero probability. In a similar way, the BCRT of T1 was assessed with the query:

Pr[<=300] (<>Transition(t4).W && x[t1]<10)

that suggested the event was almost impossible. However, the condition x[t1]<11 makes the query to propose as a result a confidence interval of [0.356958,0.456821] with a confidence degree of 95%. Table 4 collects the BCRT/WCRT response times estimated for the three tasks under EDF.

Table 4. Response times of the model in Figure 4 under EDF.

Task	BCRT	WCRT
T1	10	39
T2	18	40
T3	34	83

Figures 7 and 8 show a snapshot of the T1 and T3 executions under EDF. It emerges that now also T1 can be preempted and suspended, although with a minor degree wrt T3.

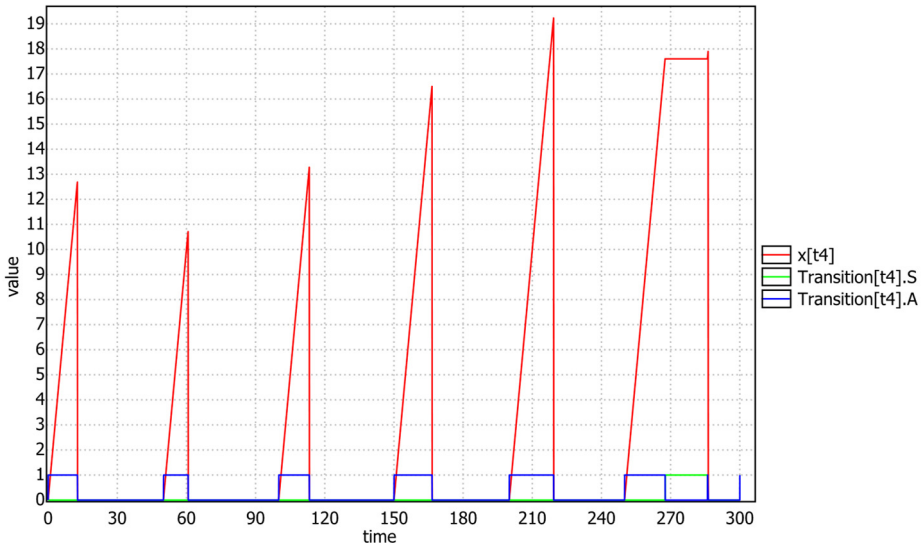


Figure 7. Snapshot of T1 executions of the t4 transition of the model in Figure 4 under EDF.

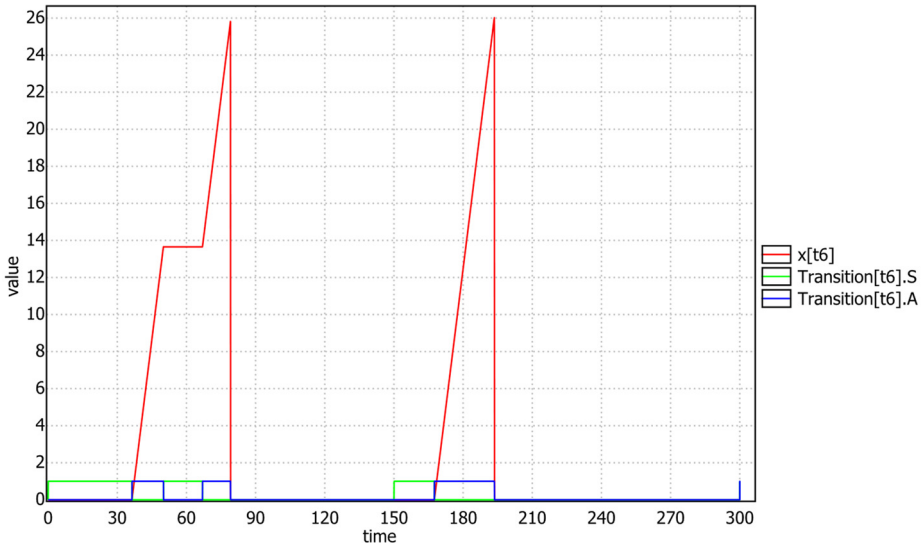


Figure 8. Snapshot of T3 executions of the t6 transition of Figure 4 under EDF.

4.2. Second Example

The task model shown in Figure 9 was considered in [15] to compare the size of the generated state graph of an ITPN model with over-approximation, and the time needed for the generation, with similar preemptive TPN tools. However, the schedulability and the task response times were not predicted. This model, which becomes rather cluttered when using inhibitor arcs for expressing the task priorities, was thoroughly investigated using the reduction of C-TPN onto Uppaal proposed in this paper. First, it was preliminarily verified that the task model is not schedulable on a single *cpu*, neither under FP nor EDF. For example, under FP, with $\pi(T1) > \pi(T2) > \pi(T3) > \pi(T4) > \pi(T5)$, even with the hypothesis of int[0,20] M[P], the model checker signals, after 389 sec, an out-of-range assignment to M[p4], to witness the boundedness degree is greater than 20. We renounced to find the exact boundedness degree of the net model. Then, the task set was split into two *cpu*s with T1 and T2 assigned to *cpu*1 and the remaining tasks to *cpu*2. Under FP scheduling, the constraint functions in Table 5 were adopted.

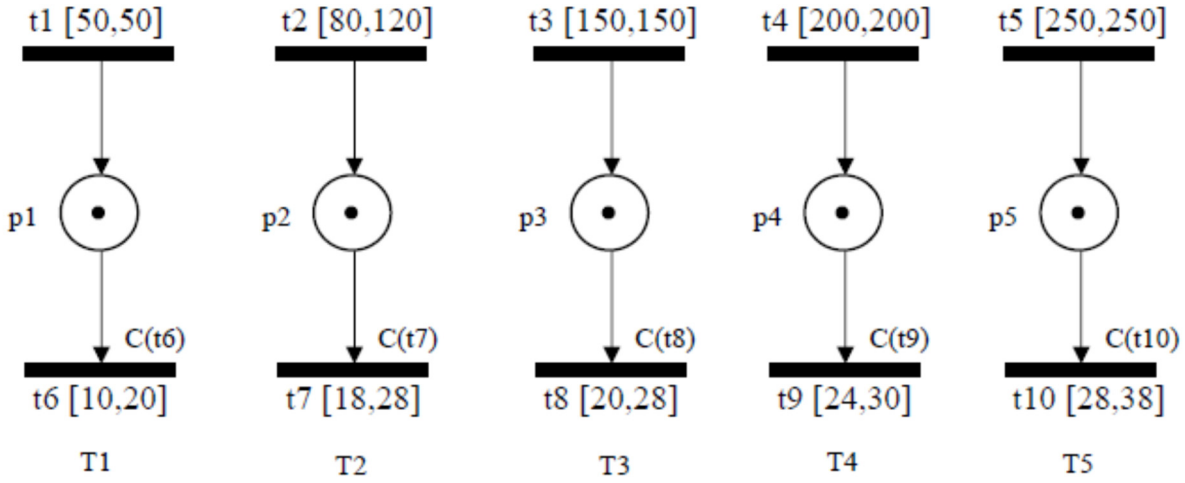


Figure 9. A C-TPN model with 5 tasks.

Table 5. The constraint functions for the task set of Figure 9 under FP and two *cpu*s.

Transition	Constraint Function C(t)
t6 (cpu1)	false
t7 (cpu1)	M[p1]>0
t8 (cpu2)	false
t9 (cpu2)	M[p3]>0
t10 (cpu2)	M[p3]+M[p4]>0

With two *cpu*s and FP, the model is now 1-bounded. The BCRT and WCRT of the various tasks were found to be as shown in Table 6.

Table 6. Response times of the task set of Figure 9 under FP.

Task	BCRT	WCRT
T1 (cpu1)	10	20
T2 (cpu1)	18	76
T3 (cpu2)	20	28
T4 (cpu2)	24	58
T5 (cpu2)	28	96

A snapshot of observed executions of task T2 and, in particular, of the "task body" t7 transition, is shown in Figure 10. It was generated by the query:

simulate[<=500]{x[t7],Transition(t7).A,Transition(t7).S}

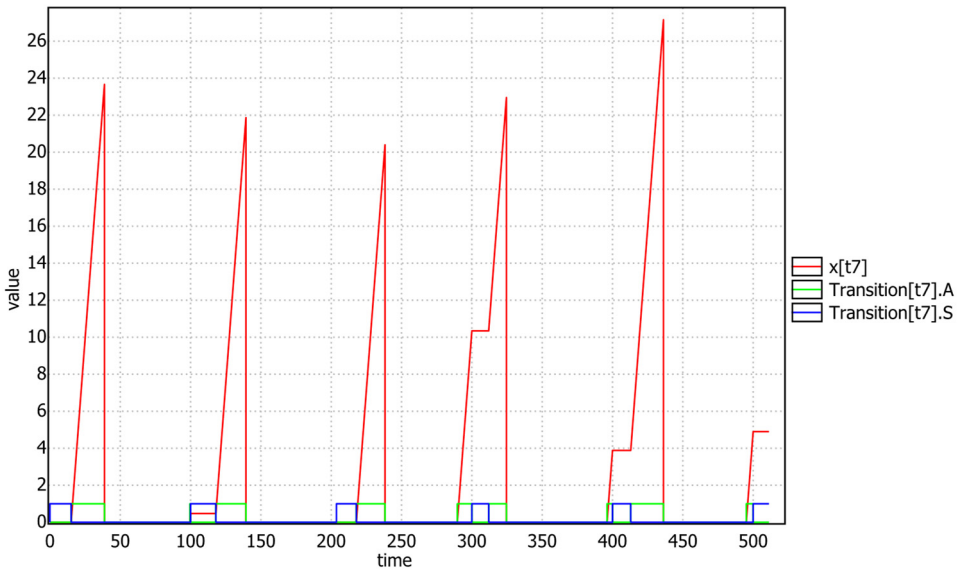


Figure 10. Snapshot of executions of transition t7 of task T2 under FP (model in Figure 9).

Also, under EDF and two *cpu*s, the task set of Figure 9 was found to be 1-bounded. The newly adopted constraint functions are indicated in Table 7. The estimated response times are collected in Table 8. Figure 11 depicts a snapshot of t7 execution under EDF.

Table 7. The constraint functions for the task set of Figure 12 under EDF and two *cpu*s.

Transition	Constraint Function C(t)
t6 (cpu1)	M[p2]>0 && 80-x[t2]<50-x[t1]
t7 (cpu1)	M[p1]>0 && 50-x[t1]<80-x[t2]
t8 (cpu2)	M[p4]>0 && 200-x[t4]<150-x[t3] || M[p5]>0 && 250-x[t5]<150-x[t3]
t9 (cpu2)	M[p3]>0 && 150-x[t3]<200-x[t4] || M[p5]>0 && 250-x[t5]<200-x[t4]
t10 (cpu2)	M[p3]>0 && 150-x[t3]<250-x[t5] || M[p4]>0 && 200-x[t4]<250-x[t5]

Table 8. Response times of the task set of Figure 12 under EDF and two *cpu*s.

Task	BCRT	WCRT
T1 (cpu1)	10	47
T2 (cpu1)	18	47
T3 (cpu2)	20	28
T4 (cpu2)	24	57
T5 (cpu2)	28	90

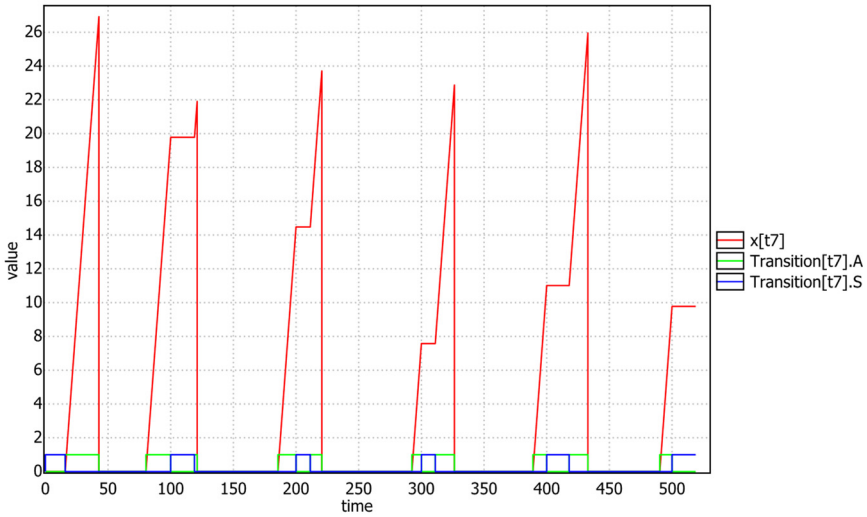


Figure 11. Snapshot of executions of transition t7 of task T2 under EDF (model in Figure 9).

4.3. Third Example

The C-TPN model of Figure 12 was taken from [13], where it was specified according to the Preemptive Time Petri Nets (pTPN) formalism implemented in the Oris tool [14]. The same model was also analyzed in [17], where pTPNs were informally mapped onto U-MC through a special set of library automata. The task set represents a realistic embedded system with 4 tasks partitioned among two *cpu*s, with $\pi(\text{T1}) > \pi(\text{T2}) > \pi(\text{T3})$ and $\pi(\text{T1}) = \pi(\text{T4})$ under FP. T1, T3 and T4 are periodic. T2 is a sporadic process with 100 being the minimal inter-distance between two consecutive instances (releases) of the task. It is worth noting that the model in Figure 12 is hard to reproduce in a readable way by using inhibitor arcs and the ITPN language described in [15].

The T1 task body consists of t2, t3, t4, t5, t6 and t7 timed actions. Task T2 generates, with its t9 transition, a message toward T1. Without such a message, T1 could execute the t3, t4 and t5 (see the inhibitor arc from p6 to t3) actions. With a message pending in p6, the T1 task instance will execute a different control path modeled by transitions t6, t7, t4 and t5. The model tasks' behavior is controlled by two mutex objects: p8 and p13. Executing t5 or t13 requires first acquiring the mutex p8. Similarly for the actions t7 and t18, which

need the mutex p13. Under priority-based scheduling, the task set assumes that mutex objects are realized according to a priority ceiling protocol [3]. Therefore, the lower priority T3 task, after having gained the p8 mutex, immediately raises its priority to that of the highest priority task, which can require the mutex, in this example, T1. T3 reacquires its low priority at the exit from its critical section. No change of priority is required between T1 and T4, which have the same highest priority and run on distinct *cpu*s, after acquiring the p13 mutex.

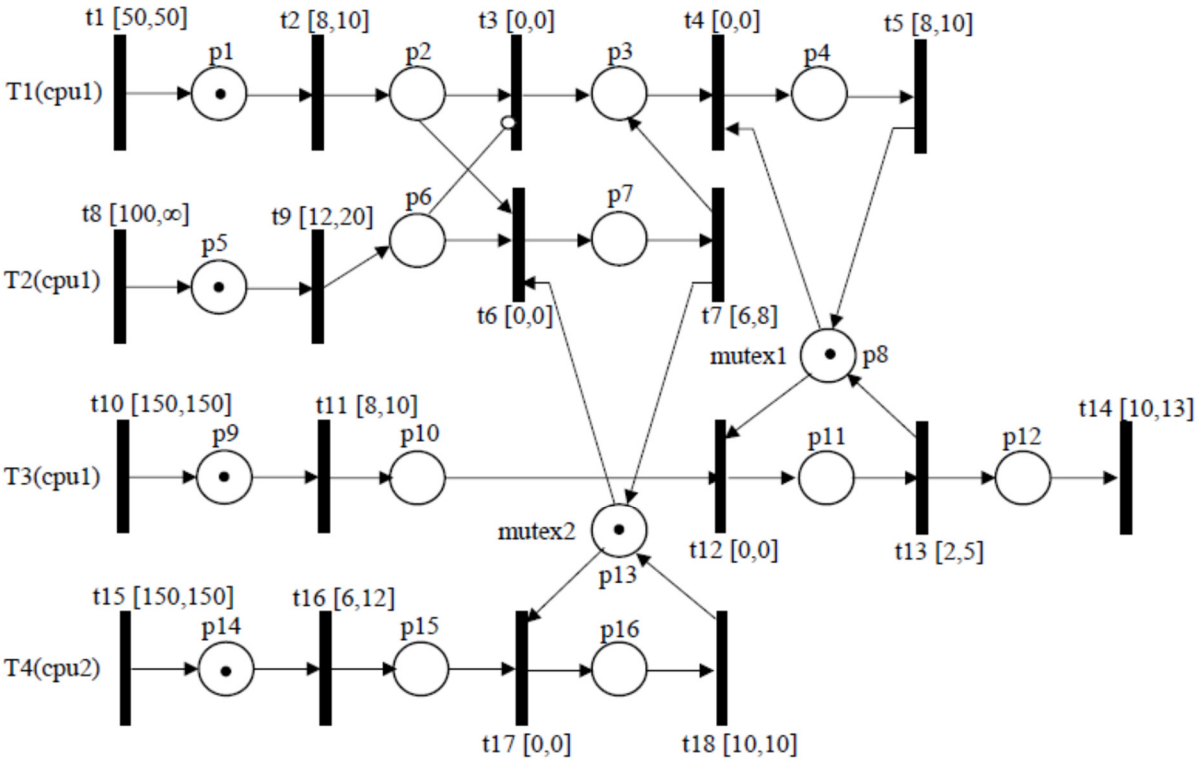


Figure 12. A task model with two *cpu*s, two mutexes and message passing.

The task model of Figure 12 will be analyzed under FP and EDF. For simplicity, the constraint functions are not shown in the basic model of Figure 12. Details about constrained transitions will be given in the next sub-sections devoted separately to FP and EDF scheduling analysis. As a preliminary step, though, the sporadic task T2 was turned into a periodic one. The worst case occurs when using the so-called "non-equivalent periodic process", where the upper bound of t8 is put equal to 100, a provision which, obviously, complicates the attainment of the schedulability. Alternatively, more than one equivalent periodic process can be considered, using the conditions, e.g., discussed in [5] on page 100. In the following, the time interval of t8 is preliminary adjusted to [100,100].

4.3.1. Schedulability Analysis under Fixed Priorities

The C-TPN model of Figure 12 is adapted for fixed priorities (FP) schedulability analysis using the constraint functions reported in Table 9. It is worth noting that transitions of task T4 are always unconstrained (default) because the task runs on the separate *cpu*2. Transitions of T1 are constrained only in the case (priority ceiling) in which the T3 task is within its critical section (there is one token in place p11). Transition t9 of task T2, which has intermediate priority between T1 and T3, is constrained either when task T1 is active or when T3 is within its critical section (p11). Finally, transitions of task T3, out of its critical section, are constrained when either the task T1 or the task T2 is active. Transition t5 is always unconstrained. While it is running, it holds the p8 mutex and neither T2 nor T3 can suspend it. For similar reasons, transition t13 of task T3 is always unconstrained.

Table 9. The constraint functions for the task set of Figure 12 under FP and two *cpu*s.

Transition	Constraint Function C(t)
t2, t3, t6, t7 (cpu1)	M[p11]>0
t9 (cpu1)	M[p1]+M[p4]+M[p7]+M[p11]>0
t11, t12, t14 (cpu1)	M[p1]+M[p4]+M[p5]+M[p7]>0
t16, t17, t18 (cpu2)	false

The model is 1-bounded. The query that asks for the absence of deadlocks is satisfied and terminates in 0.015 sec. The BCRT and the WCRT of the four tasks are reported in Table 10. Transition t5 completes the execution of an instance of T1, and, similarly, t9 of T2, t14 of T3 and t18 of T4.

Table 10. Response times of the task set of Figure 12 under FP.

Task	BCRT	WCRT
T1	16	43
T2	28	40
T3	42	128
T4	16	28

As it emerges from the response times in Table 10, the task set of Figure 12 is schedulable under FP even when the T2 sporadic task is replaced by its non-equivalent periodic version. The temporal results comply with similar results achieved in [19]. Unfortunately, in [13], the complete schedulability analysis with the response times of the chosen task set was not reported.

Figure 13 confirms, after a simulated query with a time limit of 600, that the transition t13 of task T3, once the mutex p8 was acquired, always runs to completion without suspensions, because it (due to the emulated priority ceiling protocol) holds the highest priority of T1. Instead (see Figure 14), after exiting the critical section (i.e., t13 fired), task T3 re-gains its low priority and the t14 transition can be affected by preemption and suspension.

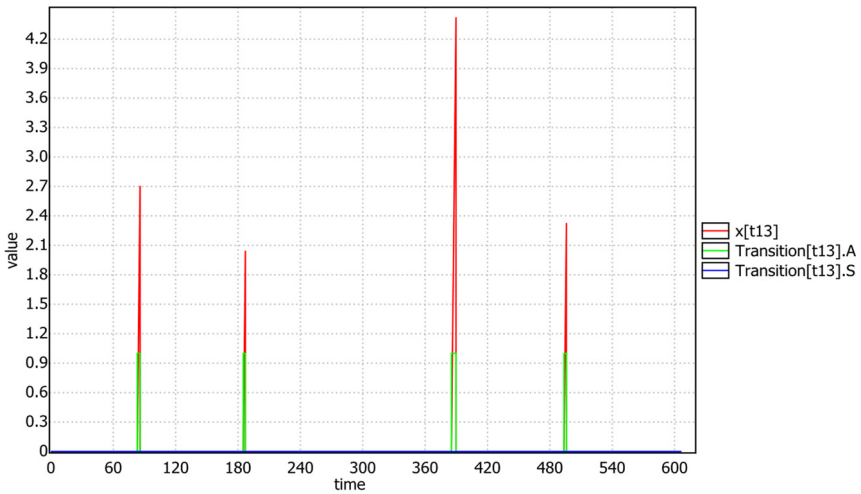


Figure 13. Snapshot of t13 executions (model in Figure 12) under FP.

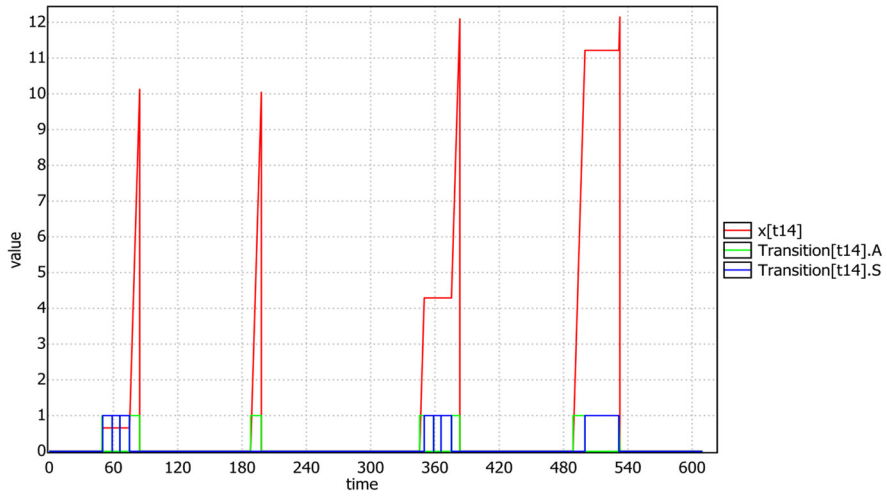


Figure 14. Snapshot of t14 executions (model in Figure 12) under FP.

Figure 15 confirms that by not using the ceiling protocol on the mutex p8, the transition t13 (critical section of task T3) can possibly be preempted by transitions of T1 and T2. The snapshot of Figure 15 was observed after some simple modifications to the constraint functions of Table 9.

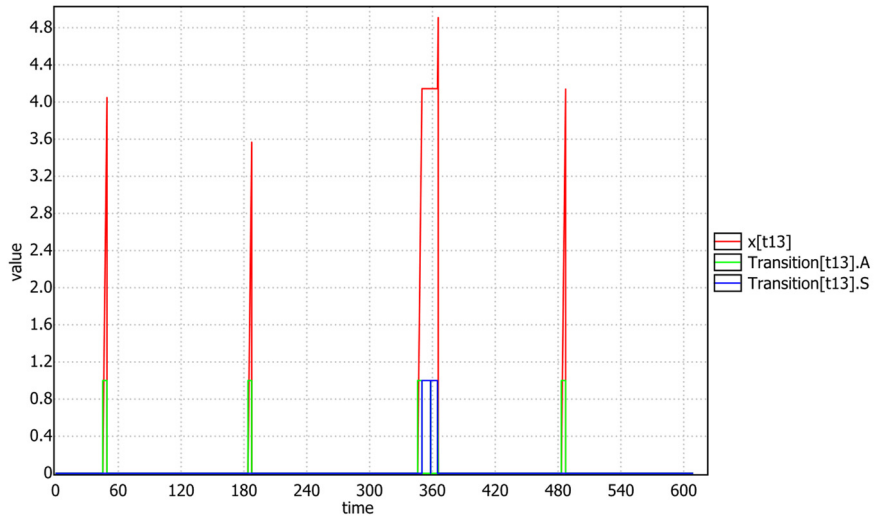


Figure 15. Snapshot of t14 executions (model in Figure 12) under FP but without using the ceiling protocol onto p8.

4.3.2. Schedulability Analysis under Dynamic Priorities

The C-TPN model of the task set in Figure 12 was adapted to work with EDF and with one cpu only, by introducing the constraint functions shown in Table 11.

The following preliminary queries were all satisfied with a proposed (practically) 0 occurrence probability. As a consequence, only one token would exist, which moves from one place to another in any task body at any time. In addition, the model is 1-bounded. The declaration int [0,1] M[P] was never violated during the schedulability work.

Pr[<=300](<>M[p1]+M[p2]+M[p3]+M[p4]+M[p7]>1)
Pr[<=300](<>M[p5]>1)
Pr[<=300](<>M[p9]+M[p10]+M[p11]+M[p12]>1)
Pr[<=300](<>M[p14]+M[p15]+M[p16]>1)

Table 11. The constraint functions for the task set of Figure 12 under EDF and one *cpu*.

Transition	Constraint Function C(t)
t2. . .t7 (T1)	M[p5]>0 && 100-x[t8]<50-x[t1] || M[p9]+M[p11]+M[p12]>0 && 150-x[t10]<50-x[t1] || M[p9]+M[p11]+M[p12]>0 && 150-x[t10]<50-x[t1]
t9 (T2)	M[p1]+M[p4]+M[p7]>0 && 50-x[t1]<100-x[t8] || M[p9]+M[p11]+M[p12]>0 && 150-x[t10]<100-x[t8] || M[p14]+M[p16]>0 && 150-x[t15]<100-x[t8]
t11. . .t14 (T3)	M[p1]+M[p4]+M[p7]>0 && 50-x[t1]<150-x[t10] || M[p5]>0 && 100-x[t8]<150-x[t10] || M[p14]+M[p16]>0 && 150-x[t15]<150-x[t10]
t16. . .t18 (T4)	M[p1]+M[p4]+M[p7]>0 && 50-x[t1]<150-x[t15] || M[p5]>0 && 100-x[t8]<150-x[t15] || M[p9]+M[p11]+M[p12]>0 && 150-x[t10]<150-x[t15]

The estimated BCRT/WCRT of the task set under EDF and one single *cpu* are reported in Table 12.

Table 12. Estimated response times of the task set of Figure 12 under EDF and one *cpu*.

Task	BCRT	WCRT
T1	16	34
T2	26	39
T3	45	90
T4	40	69

It should be noted that, under EDF, even a critical section could be interrupted if provisions like the priority ceiling protocol are not adopted, as it was completed, for simplicity, in the task set adjusted for EDF. However, if the execution of t12 occurs when it has a minimal distance to the T2 deadline, with a high probability of the execution of t13, which has a small duration, can also be completed without preemption. The situation is different for t14, which has a longer duration. All of this was confirmed by running the same simulated queries behind Figures 13 and 14. The observed behavior for the transition t13 and t14 under EDF is shown, respectively, in Figures 16 and 17.

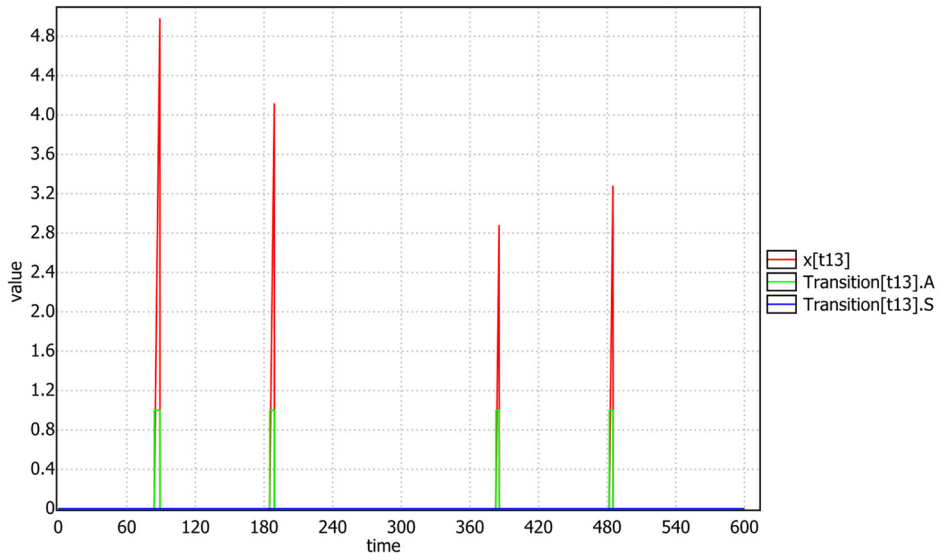


Figure 16. Snapshot of t13 executions (model in Figure 12) under EDF.

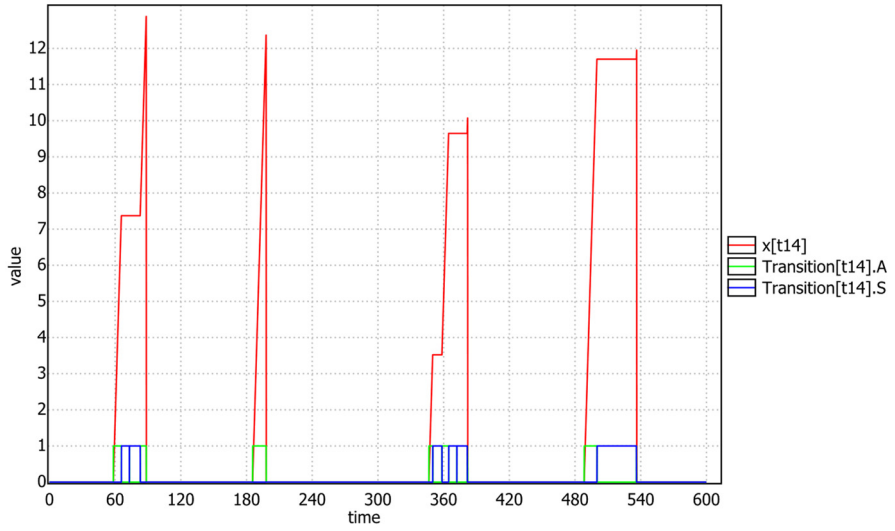


Figure 17. Snapshot of t14 executions (model in Figure 12) under EDF.

5. Conclusions

The development of embedded real-time systems with strict timing constraints is a well-known challenging problem because it requires ensuring that both functional and, most importantly, temporal requirements are ultimately met.

This paper proposes Constraint Time Petri Nets (C-TPN), a novel high-level extension of Time Petri Nets [7] with stopwatches [8–15] for formal modeling of a real-time system under the general hypothesis of the component tasks, which can be (a) periodic or sporadic, (b) can have their control and data aspects detailed, (c) can have non-deterministic execution times, (d) can use mutex objects to protect shared data variables, (e) can exchange data messages and (f) can be allocated among different *cpu*s. In addition, the behavior of a real-time system can be regulated by static or dynamic defined priorities. Some supported features are not available in known tools based on preemptive Time Petri Nets.

As a fundamental contribution, this paper proposes a reduction of a C-TPN model onto the Timed Automata (with hybrid and stochastic extensions) of the popular Uppaal toolbox [17,18], guided by formal operational semantics of a source model and the exploitation of the over-approximation in the generation of the underlying state graph. The reduction proves very effective because it allows for the assessment of the functional and temporal properties of a C-TPN model by directly using the exhaustive model checker [17] and/or the statistical model checker [18,22].

The prosecution of the research aims to address the following points. First, to experiment with further scheduling strategies, e.g., based on deadline monotonic or least slack time [3], and to integrate in a C-TPN model different kinds of servers [3] to specifically handle aperiodic tasks. Second, to extend the C-TPN modeling to cyber–physical systems [31] by explicitly specifying the communication network and the continuous aspects of environmental data variables. Third, to establish a development software methodology, e.g., in Java, based on C-TPN and the concepts of *model continuity* [31], that is, the possibility of transitioning, with minimal distortions, a source model onto a final custom implementation architecture [32]. Fourth, due to the uncertainty that unavoidably accompanies the correct prediction of the execution time of task actions (e.g., [5]), an important goal of the final real-time software is to possibly admit state degradation modes and abort decisions [33] with recovery means, so as to minimize the risks corresponding to a deadline miss. Fifth, to port C-TPN on top of the Theater actor system [34] for the schedulability analysis of large models of embedded systems and, e.g., for studying workflow scheduling schemes in cloud computing [35], by parallel simulations [28] on a multi-core machine.

Author Contributions: Both authors contributed to the conceptualization, the verification work and the writing of the paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: All the developed models and data can be asked to the authors.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lee, E.A.; Seshia, S.A. Introduction to Embedded Systems—A Cyber-Physical Systems Approach. 2011. Available online: https://ptolemy.berkeley.edu/books/leeseshia/ (accessed on 16 February 2024).
2. Yousif, A.; Bashir, M.B.; Ali, A. An Evolutionary Algorithm for Task Clustering and Scheduling in IoT Edge Computing. *Mathematics* **2024**, *12*, 281. [CrossRef]
3. Buttazzo, G. *Hard Real-Time Computing Systems*, 3rd ed.; Springer: New York, NY, USA, 2011.
4. Sha, L.; Abdelzaher, T.; Årzén, K.-E.; Cervin, A.; Baker, T.; Burns, A.; Buttazzo, G.; Caccamo, M.; Lehoczky, J.; Mok, A.K. Real time scheduling theory: A historical perspective. *Real-Time Syst.* **2004**, *28*, 101–155. [CrossRef]
5. Shaw, A.C. *Real-Time Systems and Software*; John Wiley and Sons, Inc.: Hoboken, NJ, USA, 2001.
6. Cassez, F.; Larsen, K.G. The impressive power of stopwatches. In Proceedings of the CONCUR 2000, State College, PA, USA, 22–25 August 2000; LNCS 1877. Springer: Berlin/Heidelberg, Germany, 2000; pp. 138–152.
7. Merlin, P.; Farber, D.J. Recoverability of communication protocols. *IEEE Trans. Commun.* **1976**, *24*, 1036–1043. [CrossRef]
8. Lime, D.; Roux, O.H. Expressiveness and analysis of scheduling extended time Petri nets. In Proceedings of the IFAC 2003, Singapore, 11–14 November 2003; Volume 36, pp. 189–197.
9. Roux, O.H.; Lime, D. Time Petri nets with inhibitor hyperarcs formal semantics and state space computation. In Proceedings of the ICATPN 2004, Bologna, Italy, 21–25 June 2004; Springer: Berlin/Heidelberg, Germany, 2004; pp. 371–390.
10. Lime, D.; Roux, O.H.; Seidner, C.; Traonouez, L.M. Romeo: A parametric model-checker for Petri nets with stopwatches. In Proceedings of the TACAS 2009, York, UK, 22–29 March 2009; LNCS. Springer: Berlin/Heidelberg, Germany, 2009; Volume 5505, pp. 54–57.
11. Berthomieu, B.; Lime, D.; Roux, O.; Vernadat, F. Reachability problems and abstract state spaces for time Petri nets with stopwatches. *Discret. Event Dyn. Syst.* **2007**, *17*, 133–158. [CrossRef]
12. TINA On-Line 2024. Available online: https://projects.laas.fr/tina/home.php (accessed on 16 February 2024).
13. Bucci, G.; Fedeli, A.; Sassoli, L.; Vicario, E. Timed state space analysis of real time preemptive systems. *IEEE Trans. Softw. Eng.* **2004**, *30*, 97–111. [CrossRef]
14. Bucci, G.; Carnevali, L.; Ridi, L.; Vicario, E. Oris: A tool for modeling, verification and evaluation of real-time systems. *Int. J. Softw. Tools Technol. Transf.* **2010**, *12*, 391–403. [CrossRef]
15. Abdelli, A. Time distance-based computation of the DBM over-approximation of preemptive real-time systems. *J. Log. Algebr. Methods Program.* **2024**, *136*, 100927. [CrossRef]
16. Alur, R.; Dill, D. A theory of timed automata. *Theor. Comput. Sci.* **1994**, *126*, 183–235. [CrossRef]
17. Behrmann, G.; David, A.; Larsen, K.G. A tutorial on UPPAAL. In *Formal Methods for the Design of Real-Time Systems*; Bernardo, M., Corradini, F., Eds.; LNCS 3185; Springer: Berlin/Heidelberg, Germany, 2004; pp. 200–236.
18. David, A.; Larsen, K.G.; Legay, A.; Mikucionis, M.; Poulsen, D.B. Uppaal SMC tutorial. *Int. J. Softw. Tools Technol. Transf.* **2015**, *17*, 397–415. [CrossRef]
19. Cicirelli, F.; Furfaro, A.; Nigro, L.; Pupo, F. Development of a schedulability analysis framework based on pTPN and UPPAAL with stopwatches. In Proceedings of the IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications, Dublin, Ireland, 25–27 October 2012; pp. 57–64.
20. Cicirelli, F.; Furfaro, A.; Nigro, L. Model checking time-dependent system specifications using Time Stream Petri Nets and Uppaal. *Appl. Math. Comput.* **2012**, *218*, 8160–8186. [CrossRef]
21. Pimkote, A.; Vatanawood, W. Simulation of preemptive scheduling of the independent tasks using timed automata. In Proceedings of the 10th International Conference on Software and Computer Applications, Kuala Lumpur, Malaysia, 23–26 February 2021; pp. 7–13.
22. Agha, G.; Palmskog, K. A Survey of Statistical Model Checking. *ACM Trans. Model. Comput. Simul.* **2018**, *28*, 1–39. [CrossRef]
23. Baek, H.; Lee, J. Contention-Free Scheduling for Single Preemption Multiprocessor Platforms. *Mathematics* **2023**, *11*, 3547. [CrossRef]
24. Murata, T. Petri Nets: Properties, analysis and applications. *Proc. IEEE* **1989**, *77*, 541–580. [CrossRef]
25. Plotkin, G. *A Structural Approach to Operational Semantics*; Tech. Report DAIMI FN-19; Computer Science Department, Aarhus University: Aarhus, Denmark, 1981.
26. Bowman, H.; Gomez, R.; Su, L. A tool for the syntactic detection of zeno-timelocks in Timed Automata. *Electron. Notes Theor. Comput. Sci.* **2005**, *139*, 25–47. [CrossRef]
27. Uppaal On-Line. 2024. Available online: http://www.uppaal.org (accessed on 16 February 2024).

28. Cicirelli, F.; Nigro, L. Analyzing stochastic reward nets by model checking and parallel simulation. *Simul. Model. Pract. Theory* **2022**, *116*, 102467. [CrossRef]
29. Cicirelli, F.; Nigro, L. Admission control in Home Energy Management Systems using Theatre with hybrid actors. *Modelling* **2021**, *2*, 288–307. [CrossRef]
30. Uppaal Tutorial. 1999. Available online: https://www.cis.upenn.edu/~lee/09cis480/lec-part-3-uppaal-inside.pdf (accessed on 16 February 2024).
31. Cicirelli, F.; Nigro, L.; Sciammarella, P.F. Model continuity in Cyber-Physical Systems: A control centered methodology based on Agents. *Simul. Model. Pract. Theory* **2018**, *83*, 93–107. [CrossRef]
32. Zagan, I.; Gaitan, V.G. Designing a custom cpu architecture based on hardware RTOS and dynamic preemptive scheduler. *Mathematics* **2022**, *10*, 2637. [CrossRef]
33. Qiu, Q.; Maillart, L.M.; Prokopyev, O.A.; Cui, L. Optimal condition-based mission abort decisions. *IEEE Trans. Reliab.* **2022**, *72*, 408–425. [CrossRef]
34. Nigro, L. Parallel Theatre: An actor framework in Java for high performance computing. *Simul. Model. Pract. Theory* **2021**, *106*, 102189. [CrossRef]
35. Khaleel, M.I.; Safran, M.; Alfarhood, S.; Zhu, M. Workflow Scheduling Scheme for Optimized Reliability and End-to-End Delay Control in Cloud Computing Using AI-Based Modeling. *Mathematics* **2023**, *11*, 4334. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.