

A new extension of activity networks for modeling and verification of timed systems

Hassan MOTALLEBI,¹ Mohammad Abdollahi AZGOMI,^{1,*} Mohammad Saber MIRZAEI,¹
Ali MOVAGHAR²

¹School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran

²Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Received: 07.12.2011

• Accepted: 14.06.2012

• Published Online: 02.10.2013

• Printed: 28.10.2013

Abstract: Stochastic activity networks (SANs) are a well-known petri net-based formalism used for the performance and dependability modeling of a wide range of systems. On the other hand, the growing complexity of timed systems makes it imperative to apply formal analysis techniques in the early stages of the system's development. Finding a suitable framework for the modeling, evaluation, and verification of these systems is still a great challenge. In this paper, we introduce a new formalism named timed activity networks (TANs), which are based on the activity networks that are the nondeterministic settings of the SANs. The advantages of TANs are 2-fold: 1) allowing the construction of more compact petri net-based models of timed systems and 2) allowing the assignment of time intervals to timed activities where the activity completion rates are marking-dependent, which makes TANs a better model for timed systems. Thanks to the presence of the input/output gates, TANs are capable of describing a situation whose specification using petri net-based formalisms was not practical, due to the naivety of the enabling and firing rules in these models. In addition, a great benefit of TANs is the similarity of their primitives and notations to ordinary SANs, which allows us to easily obtaining these 2 models from each other. Accordingly, SANs can be used for performance, and dependability modeling and evaluation, while TANs can be used for the model checking of timed systems. In this paper, we present the definitions, semantics, and model checking techniques of the proposed formalism. In order to model check TAN models, a transformation procedure is given for translating TAN models into the equivalent linear hybrid automaton, which can then be used with the existing techniques and tools.

Key words: Timed systems, modeling and verification, stochastic activity networks, timed activity networks, polyhedral computations

1. Introduction

Nowadays, real-time systems are widely used in a wide range of safety-critical applications. Therefore, formal guarantees of their behavioral correctness are crucial, to ensure that all of their requirements are satisfied. The behavioral correctness of real-time systems relies not only on the logical correctness of the operations, but also on the time at which the operations are performed. In hard real-time systems, the completion of an operation after its deadline is considered either a useless or even a catastrophic event. To gain confidence in the behavioral correctness of real-time systems, formal specification methods are used for expressing these systems.

In order for specifications and verification of timed systems, several time-dependent formalisms have been proposed, such as the timed extensions of the well-known untimed models like automata and petri nets [1–4].

*Correspondence: azgomi@iust.ac.ir

The theory of these formalisms creates the foundations of many studies in verification, control, and scheduling.

Timed automata (TA) may be the most successful model of real-time systems. Today, TA serves as a widely used formalism for modeling time systems with a rich theoretical background and fully developed verification tools like UPPAAL [5], HYTECH [6], CMC [7], and KRONOS [8]. However, despite the powerful modeling power of TA and its strong theoretical background, there are some subtle lacks that TA suffers from, which will be enumerated as follows. First, because of their low-level expressiveness, the size of the models for many practical systems might be unnecessarily huge [9]. Second, due to the explicit use of the invariants and constraints on the clocks, during the design of a model, inadvertent deadlocks may be injected into the model. These conditions may appear while the clocks cannot progress and no activity completion is possible [9].

With the introduction of hybrid automata [10], multirate real-valued variables can be modeled easily, but there are still some modeling situations where we need to use other formalisms for the sake of ease. For example, consider a situation where the 2 components of a system are modeled by 2 parallel hybrid automata and the state transition of one component can change the other one's evolution rate. In circumstances like this, extra elements like observers are needed to achieve this modeling goal, which will enlarge the model size. Fortunately, in many situations, petri net-based models can help if using the automata is not convenient.

In the context of real-time systems, time petri nets (TPNs), introduced by Merlin [11,12], are an extension of petri nets, which provide a framework for specifying the behavior of timed systems. Bounded TPNs form a subclass of TA, and so all of the bounded TPNs can be translated into TA [13]. Furthermore, in [13], the authors have claimed and showed that TPNs are a proper subset of TA in the sense that there exist TA A_ϕ , in which no bisimilar (even weakly) TPN exists for it. In addition, unbounded TPNs (where no limit for the number of tokens is forced) are not suitable for automatic verification, as they are too expressive. Hence, most of the verification techniques, as in this paper, are limited to the bounded TPNs.

Many stochastic extensions of petri nets were developed for performance, dependability modeling, and evaluation [1,14–16]. The behavior of these models may be demonstrated by discrete state Markov processes [17] or other stochastic processes. These powerful models have urged researchers toward further study and development of more stochastic models. One of these extensions is stochastic activity networks (SANs) [1], which have been supported by several powerful modeling tools (such as UltraSAN [18] and Möbius [19]). Different settings and extensions of SANs [1,14,21–24] have been widely used for the modeling and analysis of the functional and nonfunctional aspects of systems.

SANs have several variations and extensions:

1. **The original definitions of SANs** [20,21,24]: These models have been introduced in 1984, and are a stochastic extension of petri nets. This definition was used in the UltraSAN and Möbius modeling tools.
2. **New definition of SANs:** These models have been introduced in 2001 to alleviate some challenges of the original SANs and are based on a unified view of the system in 3 settings: nondeterministic, probabilistic, and stochastic [1]. The nondeterministic setting of SANs is called activity networks. This definition has been used in the modeling tools like SharifSAN [25], SANBuilder [14], and the Partial Differential Equation Toolbox [26].
3. **High-level extensions for SANs:** SANs have 3 high-level extensions [22]. The first extension is called hierarchical SANs (HSANs). HSANs define well-defined and encapsulated submodels, which do not limit a modeler to some predefined structures of the hierarchy. The second extension is called colored SANs

(CSANs). The main goal of CSANs is to enhance SANs with facilities for data manipulation and generate more compact models. The third extension, which is called object SANs (OSANs), has integrated object-orientation with SAN models.

The high-level expressiveness of SANs and their extensions, and their modeling power, motivated us to propose a new extension for the activity networks for the verification of timed systems. Here, in this paper, we introduce a new timed formalism, based on a new setting of activity networks. This new formalism, which is called timed activity networks (TANs), has the benefit of the similarity of its basis with SAN models, which easily allows obtaining or converting the existing SAN models into TAN models, which are appropriate for timing analysis.

Many automaton-based and petri net-based formalisms have been proposed so far for timed and hybrid systems, most of which help just in specific usages [27–30]. However, there are some difficulties in modeling timed systems with automaton-based models and petri net-based formalisms, which motivated us to propose this new activity network-based formalism. Some of these difficulties are as follows: automaton-based models like hybrid automata are not a well-adapted formalism for the behavioral description of timed systems. By increasing the complexity of the systems, the dimensions of the models increase dramatically. Meanwhile, although many timed systems can be described as a network of automata, in a wide family of systems, different system components have more sophisticated types of communications, which cannot be specified by a network of automata and one cannot exhaustively encode all of the possible behaviors of these systems as different locations of a single hybrid automaton.

The task progress speed in petri nets is usually fixed to one. However, the task progress speed in many timed systems is restricted by the types of the required resources, the amounts of the available resources, and the number of simultaneous executing tasks using those resources. The following are some examples of such situations. In stream processing systems, where data streams are processed by the processing elements, the progress speed may be restricted by the amount of available storage. In scheduling systems, tasks share a single processor using a round-robin scheduling policy and the task progress speed is determined by the number of executing tasks. In communication systems, where data packets are uploaded through a channel with a fixed capacity, the time needed for uploading a packet depends on the number of packets being uploaded simultaneously and the upload capacity of the channel.

In order to specify these situations, the task progress speed should be state dependent. However, petri net-based models do not allow for marking dependent process rates. Although the approach in [27] was capable of modeling the special case of a round-robin scheduling policy, a more general model is needed to deal with the above-mentioned situations. On the other hand, the enabling and firing rules in petri net-based models are very naive. However, in practical applications, we need more sophisticated enabling and firing rules to specify that in each system's configuration, which activities are in progress and how the progress speed of the ongoing activities is determined according to the system's configuration.

Despite the previous stochastic settings, which are used for the performance and dependability modeling of systems, TAN models are based on a nonstochastic setting. TANs are appropriate for the modeling and analysis of timed systems. For the performance and dependability evaluation based on the stochastic settings (i.e. SANs), the underlying stochastic processes are computed. However, for the model checking of the TAN models, its symbolic state space is computed using polyhedral computations.

Using the expressive power of activity networks, which is mainly achieved by the input/output gates and the related input/output functions, we have attempted to propose a new formalism for the compact description

of timed systems. In addition, we will propose our approach towards the state space generation and verification of systems based on TAN formalism.

The rest of this paper is organized as follows. Section 2 gives the informal and formal definitions of the TAN, alongside its graphical representation. The exact usage of the TAN will further be described by an example in this section. In Section 3, the semantics and behavior of the TAN are presented. In Section 4, we describe how the state space of a TAN model can be computed as state class graphs. In Section 5, we focus on the verification approaches, and especially translating TAN models into the state class hybrid automata. Next, we prove that this generated automaton is bisimilar to the original TAN. In Section 6, some related works and a comparison are given. Finally, some concluding remarks are given in Section 7.

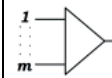
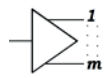
2. The proposed formalism

In this section, we introduce TANs as a new formalism for compact modeling and analysis of timed systems. To this end, we have applied some changes and refinements to activity networks to meet this goal. Before presenting a formal definition, we begin with some informal descriptions based on the definitions of the SANs that appeared in [1].

2.1. Informal presentation and graphical notations

In this section, we give the informal presentation and graphical notations of the TANs. TANs have a close relation to time petri nets and activity networks; thus it will help if the reader is already familiar with these formalisms. We try to follow the notations and graphical representations of activity networks in [1], as shown in Table 1.

Table 1. Graphical symbols of the TAN primitives.

Instantaneous activity	Timed activity	Place	Input gate	Output gate
	▭	○		

TANs have the following primitives:

Activities. Like SANs, there are 2 types of activities in the TAN models: instantaneous and timed. We represent the instantaneous activities by bars, while the timed activities are shown by rectangles. Instantaneous activities model the actions that complete in a negligible time span, which can be considered zero, while the timed activities model the actions that must be done within a rigid and predefined time interval. This is the opposite of the timed activities of ordinary activity networks that assume nondeterministic completion times. A static interval is assigned to each timed activity, representing the minimum and maximum time needed for the completion of the activity, and a *PROC* function represents the processing rate of the activity as a function of the system's marking. The static interval and *PROC* function of the timed activities can be given either near each activity or separately in a tabular format.

Places. Places in TANs have the same definition as in petri nets and are shown as circles.

Gates. The other important primitives, which have remained unchanged from the original definitions of activity networks, are gates. Using gates, more sophisticated rules can be modeled on the enabling conditions and impact of the activity completions on the markings. There are 2 types of gates: input and output.

- **Input gates.** Input gates help the modeler to assign more sophisticated enabling conditions and completion rules to the activities. An input gate has a finite set of inputs and one output. Each input of an input gate is connected to a place, while its only output is connected to an activity. Each input gate is associated with an n -ary enabling predicate, which specifies the set of markings for which the gate is enabled, and an n -ary input function, which specifies the impact of the input gate on its input places upon completion of the relevant activity. The input function is defined for all of the markings for which the enabling predicate is true.
- **Output gates.** Output gates, on the other hand, determine the impact of the activities' completion on their output places. An output gate has one input and a finite set of outputs. Each output of an output gate is connected to a place, while its only input is connected to an activity. To each output gate is associated with an n -ary output function, which specifies the impact of the output gate on its output places by the completion of the relevant activity.

2.2. Formal definitions

Now, we formally define the TANs:

Definition 1 (TAN) A timed activity network (TAN) is a 11-tuple $(P, IA, TA, IG, OG, IR, OR, D, \Pi, PROC, \mu_0)$, where

- $P = \{p_1, \dots, p_n\}$ is a finite set of places,
- IA is a finite set of instantaneous activities,
- TA is a finite set of timed activities. Moreover, $A = (TA \cup IA)$ is the set of activities.
- IG is a finite set of input gates. Each $g \in IG$ is associated with the following information:
 - m_g , the number of inputs (with $m_g \leq n$),
 - a function $f_g : N^n \rightarrow N^n$, called the input function of g that determines the impact of g on its input places by the completion of the relevant activity, and
 - a predicate $p_g : N^n \rightarrow \{true, false\}$, called the enabling predicate of g , which determines that if g is enabled in a marking or not.
- OG is a finite set of output gates. Each $g \in OG$ is associated with the following information:
 - m_g , the number of inputs (with $m_g \leq n$),
 - a function $f_g : N^n \rightarrow N^n$, called the output function of g . This function determines the impact of g on its output places after completion of the relevant activity.
- $IR \subseteq P \times \{1, \dots, |P|\} \times IG \times A$ is the input relation. IR satisfies the following conditions:
 - For any $(p, i, g, a) \in IR$, we have $i \leq m_g$,
 - For any $g \in IG$ and $i \in N, i \leq m_g$, there exist $a \in A$ and $p \in P$, such that $(p, i, g, a) \in IR$,

- For any $(p, i, g_1, a), (p, j, g_2, a) \in IR, i = j$ and $g_1 = g_2$.
- $OR \subseteq A \times OG \times 1, \dots, |P| \times P$ is the output relation. OR satisfies the following conditions:
 - For any $(a, g, i, p) \in OR$ we have $i \leq m_g$,
 - For any $g \in OG$ and $i \in N, i \leq m_g$, there exist $a \in A$ and $p \in P$, such that $(a, g, i, p) \in OR$,
 - For any $(a, g_1, i, p), (a, g_2, j, p) \in OR, i = j$ and $g_1 = g_2$.
- D is the tuple (D_{min}, D_{max}) representing the static intervals of timed activities, where $D_{min}:TA \rightarrow Q_{\geq 0}$ and $D_{max}:TA \rightarrow (Q_{\geq 0} \cup \infty)$ are functions giving, respectively, the minimum and maximum time needed for the completion of an activity. We have $\forall a \in A. D_{min}(a) \leq D_{max}(a)$.
- $\Pi : TA \times N^n \rightarrow \{true, false\}$ is the reactivation predicate. This predicate means that by entering a specific marking, the related timed activity should be reset like it has just become enabled.
- $PROC : TA \times N^n \rightarrow Q_{\geq 0}$ is the process rate function, where n is defined as before. The process rate function determines the processing rate of an enabled timed activity in the current marking.
- μ_0 is the initial marking of the network.

In a graphical representation, $(p, j, g, a) \in IR$ means that place p , is linked to the j th input of an input gate g whose output is connected to activity a . p is said to be an input place of a and g is referred to as an input gate of a . Similarly, $(a, g, j, p) \in OR$ means that activity a is linked to the input of an output gate g whose j th output is connected to the place p . g is said to be an output gate of a and p is referred to as an output place of a .

A marking of a TAN is a function $\mu \in N^n$, where $n = |P|$. For each place $p \in P$, $\mu(p)$ is the number of tokens in the place p . An activity is enabled in a marking μ if in that marking all of the enabling predicates of its input gates are true. The set of enabled activities in the marking μ is denoted by $enabled(\mu)$.

Definition 2 (Enabled activities) An activity $a \in (TA \cup IA)$ is enabled in a marking μ , if for each input gate g of a , we have $p_g(\mu) = true$. Hence:

$$a \in enabled(\mu) \Leftrightarrow \forall (p, i, g, a) \in IR, p_g(\mu) = true.$$

Upon the completion of an activity a in a marking μ , first, the impact of the input gates of a is applied and a transient marking $\mu' = pre(\mu, a)$ is obtained, then the impact of the output gates of a is applied on μ' and the result marking $post(\mu', a)$ will be obtained. These 2 functions are defined as follows:

Definition 3 (pre(μ, a)) For each activity a , the impact of the input gates g_1, \dots, g_k of a on the marking μ , is defined as follows:

$$pre(\mu, a) = f_{g_1}(f_{g_2}(\dots f_{g_k}(\mu) \dots)),$$

where f_{g_i} is the function of the input gate g_i of a .

Definition 4 ($\mathbf{post}(\mu, \mathbf{a})$) For each activity a , the impact of the output gates g_1, \dots, g_k of a on the marking μ , is defined as follows:

$$post(\mu, a) = f_{g_1}(f_{g_2}(\dots f_{g_k}(\mu)\dots)),$$

where f_{g_i} is the function of the output gate g_i of a .

The conditions imposed on the input and output relations (IR and OR in Definition 1) imply that the input (output) gates g_1, \dots, g_k of an activity a affect different places. Therefore, the ordering of gates in Definitions 3 and 4 do not matter, i.e. different orderings produce the same pre ($post$) functions.

The activity's completion is an atomic process, so it cannot be broken. Therefore, when an activity is chosen to complete, both or none of the pre and $post$ functions will perform. Therefore, the completion of a in a marking μ will result in $post(pre(\mu, a), a)$, which in the rest of this paper is shortly denoted as $next(\mu, a)$.

Aside from the instantaneous activities of TANs, between any 2 activity completions, the timed activities progress based on the $PROC$ function. According to the $PROC$ functions, the vector \vec{PROC} can be defined, which shows the progress rate of all of the timed activities. This definition is used for finding the state space of the TAN in the rest of this paper.

Definition 5 (*Process rate vector*) The process rate vector is denoted by:

$$\vec{PROC}(\mu) = (PROC(a_1, \mu), \dots, PROC(a_{|TA|}, \mu)),$$

where $PROC(a, \mu)$ is the process rate of the activity a at μ . We have $PROC(a, \mu) = 0$, if the timed activity a is not enabled in μ .

Based on the activities' enabling rules, more than one activity can be enabled in a marking. A marking is called unstable if it enables at least one instantaneous activity. Since instantaneous activities have priority over timed activities, in an unstable marking, no timed activity can complete. Another notable fact about the completion rules of the activities is that when more than one timed activity is enabled, among simultaneously enabled activities, in a nondeterministic manner, one will complete first, based on its static interval.

2.3. Example

As a simple example of TAN models, consider the model of Figure 1 whose static intervals, $PROC$ functions, and gate table are, respectively, shown in Tables 2 and 3. However, the primitives and behavior of TANs will be explained further in the following illustrative example.

Table 2. Static intervals and $PROC$ functions of the model in Figure 1.

Activity	Static interval	$PROC$ function
a_1	$[0, 3]$	1
a_3	$[1, 2]$	1
a_4	$[4, 5]$	1

Table 3. The gate table of the model in Figure 1.

Gate	Enabling predicate	Function
g_1	$\mu(x_1) \geq 2 \wedge \mu(x_2) \geq 1$	$\mu(x_1) = \mu(x_1) - 2; \quad \mu(x_2) = \mu(x_2) - 1;$

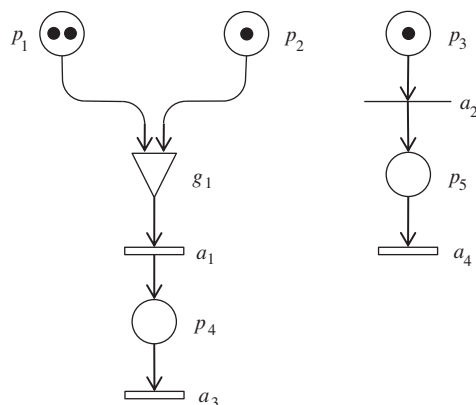


Figure 1. Graphical representation of an example TAN model.

Example 1 (File uploading system) This example describes a file uploading system. In this system, n parallel client machines with video capturing capabilities store the captured video packets on their local buffer and then upload the packets to a centralized server. The server is a real-time packet receiving and storing server with a constant download capacity ω that concurrently receives multiple data packets from different clients. Each client alternates between 2 modes. A client c_i spends $[\alpha_i^p, \beta_i^p]$ time units in the passive mode and then switches to the active mode. When a client is active, it generates packets of data (if its local memory of size φ_i (the place $P_{buf, i}$) is not full).

The generation of each packet takes $[\alpha_i^c, \beta_i^c]$ time units to complete. After $[\alpha_i^a, \beta_i^a]$ time units, c_i switches back to the passive mode. The data packets generated by the client c_i are first stored in its local memory (the place $P_{buf, i}$) and are then are uploaded to the server. The uploading time for each packet varies based on ω and the number of clients that are uploading simultaneously. At the server side, the stored packets are downloaded, compressed, and archived. If only one packet is being compressed, the compression of each packet takes $[\alpha^{comp}, \beta^{comp}]$ time units.

If the aggregate number of compressed and uncompressed packets exceeds the size of the server's local memory, φ_{server} , the uploading process in all of the clients will cease. The compressed packets are stored on the server's disk each packet taking $[\alpha^{bckup}, \beta^{bckup}]$ time units to be written.

The TAN model for the above file uploading system is shown in Figure 2. In this model, the clients, 1 to n , are separated by dashed boxes. For the i th client, the places $P_{p, i}$ and $P_{a, i}$ represent the passive and active modes, respectively, while $T_{p, i}$, $T_{a, i}$ and the related functions, shown in Table 4, regulate the duration of these 2 modes. The timed activity $T_{buf, i}$ models the packet generating and the place $P_{buf, i}$ models the number of captured packets in the local memory of the i th client, which are buffered for uploading. The number of uploading clients at each moment is kept by the place $P_{counter}$. Once a packet is produced in c_i , the number of tokens in this place (denoted by $\mu(P_{counter})$) is increased by one if there is no token in the place $P_{buf, i}$ (which means it is the first time that the i th client has generated a packet). Moreover, when the last packet in the buffer $P_{buf, i}$ is uploaded, $\mu(P_{counter})$ is decreased by one.

These rules are established using the functions of the output gate $OG_{buf, i}$ and the input gate $IG_{upl, i}$, which are shown in Table 5. On the other hand, at the server side, the place P_{uncomp} models the recently downloaded packets. These packets are analyzed and compressed, which takes $[\alpha^{comp}, \beta^{comp}]$ time units for each packet. The packet compression is modeled by T_{comp} . The place P_{comp} represents the compressed packets.

The predicate function of the input gate IG_{full} checks if the number of packets in the server's buffer exceeds the threshold φ_{server} , and, if so, the instantaneous activity I_{full} is activated and instantaneously a token is put in the place P_{full} . When the place P_{full} contains a token (which means the server's buffer is full), the enabling predicates of all of the gates $IG_{upl,i}$ are false and the clients cannot upload their packets. The gate IG_{backup} and the activity T_{backup} model the backup function of the server. Based on the function of IG_{backup} , if the server's buffer was already full, after completion of T_{backup} , P_{full} is reset, which means that the server's buffer is not full anymore and the uploading of packets can continue.

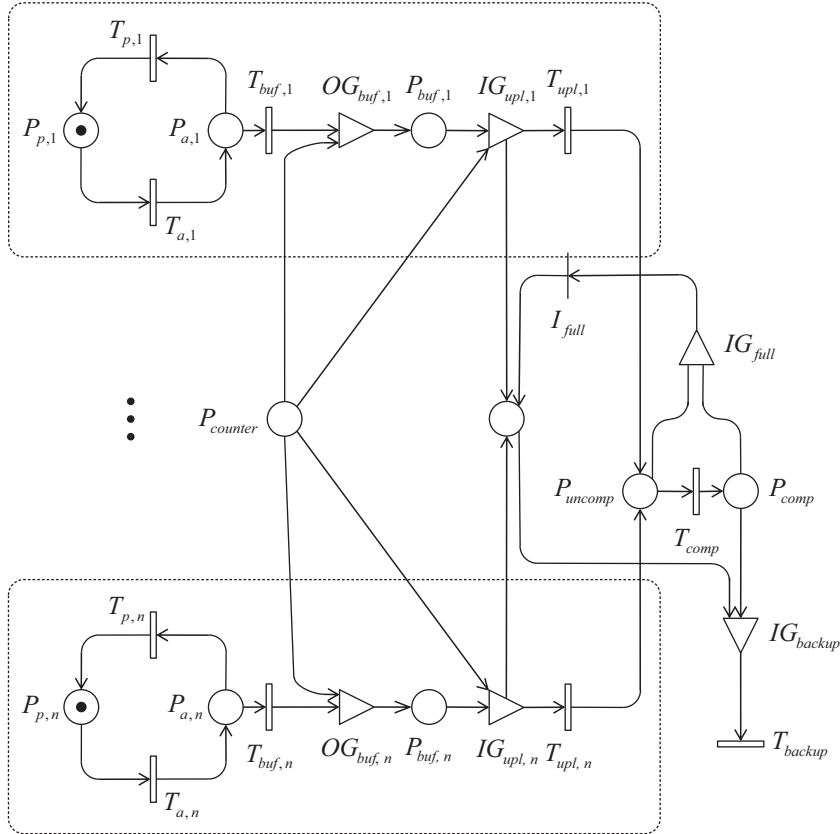


Figure 2. The TAN model of the real-time file uploading system.

Table 4. Static intervals and PROC functions of the timed activities of the model in Figure 2.

Timed activity	Static interval	PROC function
$T_{a,i}$	$[\alpha_i^a, \beta_i^a]$	1
$T_{p,i}$	$[\alpha_i^p, \beta_i^p]$	1
$T_{crt,i}$	$[\alpha_i^{crt}, \beta_i^{crt}]$	1
T_{upl}	$[\alpha_i^{upl}, \beta_i^{upl}]$	$\frac{\omega}{\mu(P_{counter})}$
T_{comp}	$[\alpha^{comp}, \beta^{comp}]$	1
T_{bckup}	$[\alpha^{bckup}, \beta^{bckup}]$	1

As shown in the TAN model of Figure 2, the input gates connect the places to the activities, while the output gates connect activities to the places. In addition to the explicit gates, some standard gates exist that are depicted by a single arrow from the sources to the destinations. For example, in this model, $P_{a,1}$ is

connected to $T_{p,1}$ using a simple arrow, which is a short depiction for an input gate with the enabling predicate $P_g=true$ iff $\mu(P_{a,1}) > 0$. The function of this standard gate decreases $\mu(P_{a,1})$ by one. As an example of the standard output gates, consider the simple arrow connecting the activity $T_{a,1}$ to the place $P_{a,1}$ with the default function, which increases $\mu(P_{a,1})$ by one.

Table 5. Gate table of the model in Figure 2.

Gate	Enabling predicate	Function
$OG_{buf,i}$		$if(\mu(P_{buf,i}) == 0) \mu(P_{counter}) ++, \mu(P_{buf,i}) ++;$
$IG_{buf,i}$	$\mu(P_{a,i}) > 0 \wedge \mu(P_{buf,i}) < \varphi_i$	
$IG_{upl,i}$	$\mu(P_{full}) = 0 \wedge \mu(P_{buf,i}) > 0$	$\mu(P_{buf,i}) --, if(\mu(P_{buf,i}) == 0) \mu(P_{counter}) --;$
IG_{full}	$\mu(P_{uncomp}) + \mu(P_{comp}) \geq \varphi_{server}$	$\mu(P_{uncomp}) := \mu(P_{uncomp}), \mu(P_{comp}) := \mu(P_{comp});$
IG_{backup}	$\mu(P_{comp}) \geq 0$	$\mu(P_{comp}) --, \mu(P_{full}) := 0;$

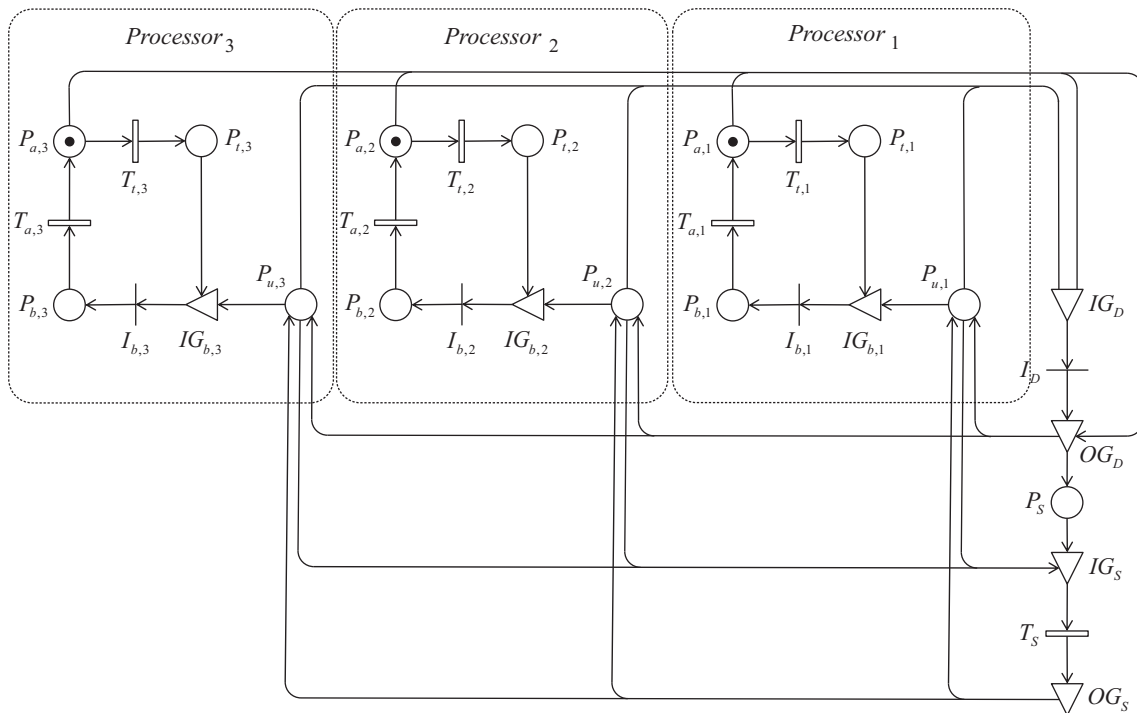


Figure 3. The TAN model of the task processing system.

The file uploading system is an example of a system whose specification using the TPN formalism is not practical. Since during the completion of an activity the process rate of that activity may change, the TPN formalism is not suitable for describing such systems. On the other hand, the specification of this system using the automaton-based models, like hybrid or TA, requires the encoding of a large number of states. However, as shown in Figure 2, using the TAN model, the file uploading system can be expressed in a natural way. In our next example, we discuss a system that can be practically described either as a TPN or a TAN model.

Example 2 (Task processing system) As our next example, consider an N -processor system whose processors alternates between 2 modes: busy and available. Each processor p_i spends $[\alpha_b, \beta_b]$ time units in the busy mode and $[\alpha_a, \beta_a]$ time units in the available mode. We aim to use the free processing cycles of the processors

to accomplish a sequence of tasks. Each time that a task is accomplished and some processors are still in the available mode, the next task is taken from the task sequence, divided among the available processors, and its execution is started. However, if during the execution of a task some other processors become available, those processors cannot be used in the execution of the current tasks. When the execution of a task is finished, each executing processor may either switch back to busy mode (if their available duration is over) or start the next task. Assume that all of the N processors are identical and when k processors participate in the execution of a task, the execution of that task takes $[\alpha/k, \beta/k]$ time units.

The TAN model shown in Figure 3 describes the task processing system for $N = 3$. The static intervals, PROC functions, and gate table of this model are, respectively, shown in Tables 6 and 7. Using the TAN model, the system can be expressed in a natural way. As shown in Figure 3, in the TAN model, the processors are separated by dashed boxes. For the i th processor, p_i , the places $P_{a,i}$ and $P_{b,i}$ represent the available and busy modes, respectively. The transition from the busy mode to the available mode is modeled by the timed activity $T_{a,i}$, while the timed activity $T_{t,i}$ keeps track of the available duration. The marking of the place $P_{a,i}$ determines whether p_i is participating in executing the current tasks or not. When the available duration of p_i is over, 2 cases are possible: if p_i is not participating in executing the current tasks, the instantaneous activity $I_{b,i}$ is completed and p_i is in the busy mode; otherwise, the enabling predicate of the input gate $IG_{b,i}$ remains false until the execution of the current task is finished and p_i is free, then $I_{b,i}$ is completed and p_i transitions to the busy mode.

Table 6. Static intervals and PROC functions of the timed activities of the model in Figure 3.

Timed activity	Static interval	PROC function
$T_{a,i}$	$[\alpha_b, \beta_b]$	1
$T_{t,i}$	$[\alpha_a, \beta_a]$	1
T_S	$[\alpha, \beta]$	$\mu(P_{u,1}) + \mu(P_{u,2}) + \mu(P_{u,3})$

Table 7. Gate table of the model in Figure 3.

Gate	Enabling predicate	Function
$IG_{b,i}$	$\mu(P_{t,i}) = 1 \wedge \mu(P_{u,i}) = 0$	$\mu(P_{t,i}) := 0;$
IG_D	$\left(\bigvee_{i=1}^3 \mu(P_{a,i}) = 1\right) \wedge \left(\bigwedge_{i=1}^3 \mu(P_{u,i}) = 0\right)$	
OG_D		for $i := 1$ to 3 do $\mu(P_{u_{a,i}}) := \mu(P_{a,i});$
IG_S	$\mu(P_S) = 1$	
OG_S		for $i := 1$ to 3 do $\mu(P_{u,i}) := 0;$

Due to the enabling predicate of IG_D , when one or more processors are available, a new task is divided among the available processors and the marking of the places $P_{u,1}$, $P_{u,2}$, and $P_{u,3}$ is set accordingly. As shown in Table 6, the process rate of the timed activity depends on the number of available processors. As the execution of the current task is completed (activity T_S), the marking of the places $P_{u,1}$, $P_{u,2}$, and $P_{u,3}$ is set to zero.

As shown in Figure 4, the task processing system can also be specified as a TPN model. In this model, the marking of the place B determines if a task is currently being processed or not. The instantaneous transitions, I_{001} through I_{111} in this model, represent the 7 possible cases for the 3 processors, with at least 1 available processor. For the number of available processors, 3 cases are possible. In case i processors ($1 \leq i \leq 3$) are available, the place N_i is marked, and the timed transition T_i is enabled with execution time $[\alpha/i, \beta/i]$.

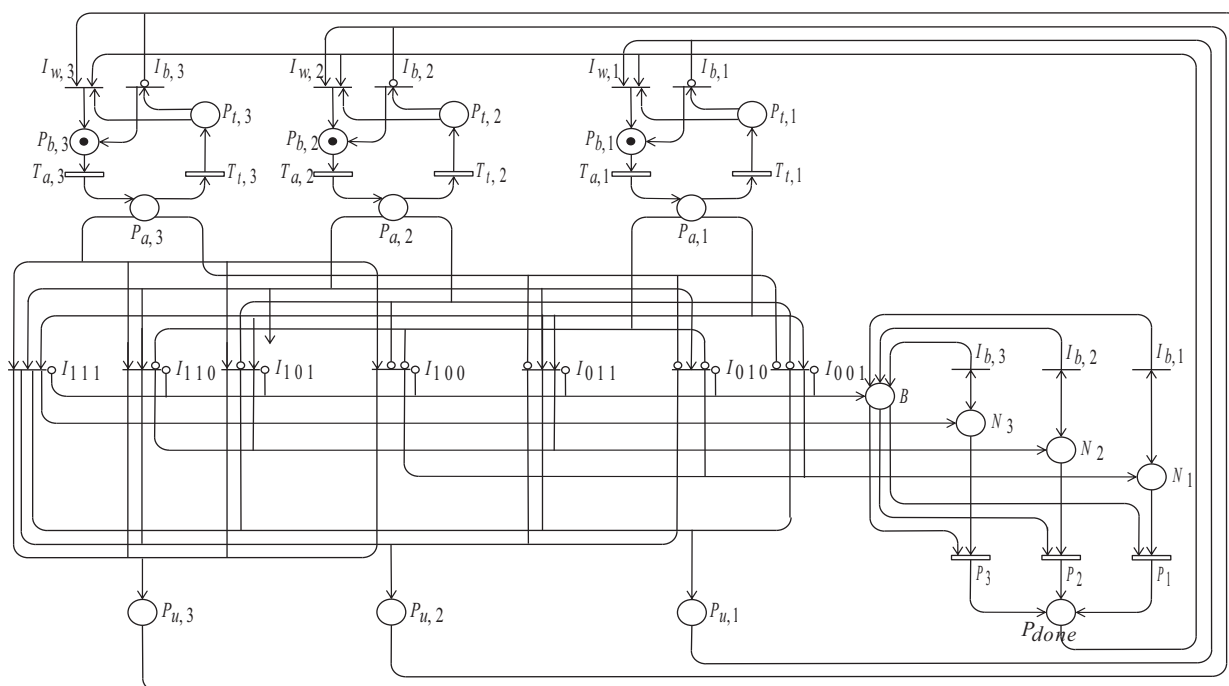


Figure 4. The TPN model of the task processing system.

As shown in Figures 3 and 4, thanks to the input/output gates, the models in TANs are more succinct than the TPN models. In order to have a comparison between the TPN and TAN models, in Table 8, we compare the size of the TPN and TAN models of the task processing system with N processors, where N ranges from 2 to 10. The model sizes are given in terms of the number of graph nodes in each model. As shown in Table 8, the TAN models are significantly more succinct than the TPN models. The sizes of the models are also shown for the case of the nonidentical processors. If the processors are nonidentical in the TPN models, additional places and transitions are needed to keep track of the set of available processors, and so on. However, for the TAN models, the model size is the same as the identical case.

Table 8. A comparison of the size of the TPN and TAN models for the task processing system.

N	#Nodes Identical processors		#Nodes Nonidentical processors	
	TPN	TAN	TPN	TAN
2	27	19	22	19
3	42	25	38	25
4	61	31	62	31
5	88	37	102	37
6	131	43	174	43
7	206	49	310	49
8	345	55	574	55
9	612	61	1094	61
10	1135	67	2126	67

3. Semantics of the TANs

After defining the basic concepts, now we concentrate on the semantics of the TAN models. A state (configuration) of a TAN model can be represented by a tuple (μ, v) , where μ is the marking of the net and v is the valuation of its timed activities. A valuation is a mapping $v \in (R_{\geq 0})^{TA}$ such that $\forall a \in TA$, $v(a)$ is the progress amount of the activity a since its activation. Note that $v(a)$ is meaningful only for the enabled timed activities. The process rate function $PROC$ gives for each activity a the speed at which $v(a)$ increases.

Definition 6 (State of the TAN) A state of a TAN is defined as a tuple $s = (\mu, v)$, where μ is the marking of the net and v is the valuation vector.

After the completion of an activity, some activities' valuation must be reset. In order to decide the set of timed activities whose valuation should be reset, the notion of newly enabled activities is defined as follows.

Definition 7 (Newly enabled activities) The function $\uparrow enabled(\mu, a) \in 2^{TA}$ denotes the set of newly enabled timed activities after the completion of the activity a in a marking μ . The completion of a will lead to the marking $next(\mu, a)$. An enabled activity $a' \neq a$ is newly enabled iff a' is disabled in the marking $pre(\mu, a)$ and is enabled in the marking $next(\mu, a)$ or its reactivation predicate at the marking $next(\mu, a)$ is true. In addition, the activity a is always newly enabled after its completion. Therefore, we have:

$$a' \in \uparrow enabled(\mu, a) \Leftrightarrow a' \in enabled(next(\mu, a)) \wedge (a' \notin enabled(pre(\mu, a)) \vee a' = a \vee \Pi(next(\mu, a), a)).$$

As mentioned before, $\Pi(\mu, a)$ is the reactivation predicate, which determines whether the activity a is reactivated by reaching the marking μ . When an activity is reactivated, its corresponding progress variable is set to zero.

Definition 8 (Semantics of the TAN) The semantics of TAN, T , is defined as the timed transition system (TTS) [31] $S_T = (S, s_0, \rightarrow)$, where

- $S = N^P \times (R^+)^{TA}$
- $s_0 = (\mu_0, \bar{0})$ where μ_0 is the initial marking and $\bar{0}$ is the zero vector.
- $\rightarrow \in S \times (TA \cup IA \cup R^+) \times S$ is the transition relation including continuous and discrete transitions:
 - $\forall t \in R^+$ the continuous transition represents the time progress:

$$(\mu, v) \xrightarrow{t} (\mu, v') \Leftrightarrow \begin{cases} \forall a \in TA \cap enabled(\mu), v'(a) = v(a) + PROC(\mu, a) \times t, \\ \forall a \in TA \cap enabled(\mu) \Rightarrow v'(a) \leq D_{max}(a), \\ enabled(\mu) \cap IA = \emptyset \end{cases}$$

- $\forall a_i \in IA$ the discrete transition relation is:

$$(\mu, v) \xrightarrow{a_i} (\mu', v') \Leftrightarrow \begin{cases} a_i \in enabled(\mu) \cap IA \\ \mu' = next(\mu, a) \\ \forall a \in TA, v'(a) = \begin{cases} 0 & \text{if } a \in \uparrow enabled(a_i, \mu) \\ v(a) & \text{Otherwise.} \end{cases} \end{cases}$$

- $\forall a_t \in TA$ the discrete transition relation is:

$$(\mu, v) \xrightarrow{a_t} (\mu', v') \Leftrightarrow \begin{cases} a_t \in \text{enabled}(\mu) \cap TA \\ \text{enabled}(\mu) \cap IA = \emptyset \\ \mu' = \text{next}(\mu, a) \\ D_{\min}(a) \leq v(a_t) \leq D_{\max}(a), \\ \forall a \in TA, v'(a) = \begin{cases} 0 & \text{if } a \in \uparrow \text{enabled}(a_t, \mu) \\ v(a) & \text{otherwise.} \end{cases} \end{cases}$$

An enabled timed activity $a_t \in TA$ can be completed in a state (μ, v) if and only if μ is a stable marking and $D_{\min}(a) \leq v(a_t) \leq D_{\max}(a)$. The impact of completion of a on the state of system is computed as follows: the new marking of the system is $\mu' = \text{next}(\mu, a)$ and for each activity a_i we have $v'(a_i)$ if a_i is newly enabled in μ' and $v'(a_i) = v(a_i)$ otherwise.

4. State space computation of the TAN models

In this section, our approach toward the computation of the state space of TAN models will be discussed. Due to the existence of real-valued progress variables corresponding to timed activities, the state space of a nontrivial TAN model is an infinite and uncountable set. Therefore, the set of states of a TAN cannot be exhaustively enumerated. However, it is possible to partition the infinite state space into a countable set of groups of states called state classes [32].

In time petri nets, which do not feature any stopwatch and varying process rate for transitions, state classes can be encoded using the well-known difference bound matrices (DBM) data structure [27]. However, the presence of *PROC* function makes it impossible to use DBM for state space computation of TANs, and so we have to use general polyhedral representation unless in case of over-approximation. However, in this paper, we stick to precise analysis instead of approximation approaches. Accordingly, the states of a TAN are classified into state classes with general polyhedral domains.

Definition 9 (State class) A state class of a TAN is a tuple $C = (\mu, D)$, where μ is the marking and D is the groups of valuation forming a convex polyhedron as the domain of a state class. Therefore, a domain of a TAN is $|TA|$ -dimensional convex polyhedron.

The symbolic state space of a TAN model is computed as a state class graph. The nodes of the state class graph are state classes. In finding the state class graph of a TAN, after each activity's completion, an already found state class may be created. This state class should not be added to the state class graph if it is exactly the same as or included in an already added one.

Definition 10 (State class inclusion) The state class $C_2 = (\mu_2, D_2)$ is included in $C_1 = (\mu_1, D_1)$ if and only if $\mu_1 = \mu_2$ and $D_2 \subseteq D_1$.

The obvious approach in finding the state class graph is to follow all possible activity completion paths starting from the initial state class $C_0 = (\mu_0, D_0)$. Accordingly, from any given state, all activities that can be completed should be considered.

Definition 11 (Completeness) An activity a in the state $s = (\mu, v)$ is completable if and only if:

$$a \in \text{completable}(s) \Leftrightarrow (a \in \text{enabled}(\mu) \cap IA) \vee \begin{cases} \text{enabled}(\mu) \cap IA = \emptyset \\ a \in \text{enabled}(\mu) \cap TA \\ D_{\min}(a) \leq v(a) \leq D_{\max}(a) \end{cases}$$

Moreover, an activity a is completable in the state class $C = (\mu, D)$ if and only if $a \in \text{completable}(s)$ for some state $s \in C$.

From a given state or state class a set of states can be reached by letting time progress toward the process rate vector \vec{PROC} . Based on this concept, the time closure of a state class is defined as follows:

Definition 12 (Time closure) Given a state class $C = (\mu, D)$, (forward) time closure $TC(C)$ of C is a class of states which are obtained by letting time progress from a state in C :

$$s' = (\mu, v') \in TC(C) \Leftrightarrow \exists s = (\mu, v) \in C \cdot \begin{cases} \exists t \in R^+. v' = v + t \times \vec{PROC}(\mu) \\ \text{enabled}(\mu) \cap IA = \emptyset \\ v' \in \bigcap_{a \in \text{enabled}(\mu) \cap TA} \llbracket v(a) \leq D_{\max}(a) \rrbracket \end{cases}$$

Based on the concepts of time closure and completeness, the whole state class graph can be obtained by computing all successor state classes from the initial state class using a breadth first search algorithm. As described before, in each moment, 2 types of system transitions can happen: continuous and discrete. With regard to continuous transitions, from the state class $C = (\mu, D)$, if no instantaneous activity is enabled, by letting time progress a state class $C' = TC(C) = (\mu, D')$ is obtained and added to the state space. From the state class C' , given the completable timed activity a (among all completable activities in this state class) after completion of a , state class $C'' = (\mu', D')$ is obtained. This state class is computed by the function $Successor(C', a)$. In the related state class graph, this transition is denoted by the edge $C' \xrightarrow{a} C''$ with the label a from the class C' to C'' .

Definition 13 (Successor(C, a)) If from a state class $C = (\mu, D)$, the timed activity a is completable then the successor state class of C by completion of a , $(\mu', D') = Successor(C = (\mu, D), a)$ is computed by following these steps:

- $\mu' = \text{next}(\mu, a)$
- D' is found from D by:
 - $\forall a \in \text{enabled}(\mu) - \text{enabled}(\mu'), \text{ set } v(a) = 0$
 - $\forall a \in \text{enabled}(\mu) \cap \uparrow \text{enabled}(\mu, a), \text{ set } v(a) = 0$

Accordingly, the state space generation of TAN from the initial state $C_0 = (\mu_0, \vec{0})$ is performed by algorithm 1 presented in Figure 5. In the following, we briefly discuss the correctness and complexity of the state space computation algorithm. The algorithm computes the set of reachable states by means of a forward depth-first search in the state class graph. Let us assume that the system is initially in the state $(\mu_0, \vec{0})$. The algorithm starts with an initial state class $C_0 = (\mu_0, \{\vec{0}\})$ containing only the state $(\mu_0, \vec{0})$. In line 4 of the

algorithm, an already obtained state class is popped from the stack and its future states are computed. In line 9 of the algorithm, if no immediate activity is enabled from the current class, the time closure of the current class is computed according to Definition 12. However, if some immediate activities are enabled from a class, in lines 10 and 11 of the algorithm, the successor state class of the current class with respect to the completion of each of those activities are obtained according to Definition 13 and put on stack for further processing.

Algorithm 1. Symbolic state space computation

Input: The initial state $C_0 = (\mu_0, \{\vec{0}\})$

Output: The state class graph, SCG

```

1:  $SCG := empty$ 
2:  $enqueue(C_0)$ 
3: do
4:    $C = (\mu, D) := dequeue()$ 
5:   if  $(enabled(\mu) \cap IA = \emptyset)$ 
6:      $C = (\mu, D) := TC(C)$ 
7:   if  $(C$  is not included in  $SCG)$ 
8:     add  $C$  to the  $SCG$ 
9:   if  $(enabled(\mu) \cap IA \neq \emptyset)$ 
10:    for each  $a \in enabled(\mu) \cap IA$  do
11:       $C' := successor(C, a)$ 
12:      if  $(C'$  is not included in  $SCG)$ 
13:         $enqueue(C')$ 
14:    else
15:      for each  $a \in completable(C)$  do
16:         $C' := successor((\mu, (D \cap \llbracket D_{min}(a) \leq v(a) \leq D_{max}(a) \rrbracket)), a)$ 
17:        if  $(C'$  is not included in  $SCG)$ 
18:           $enqueue(C')$ 
19: until  $queue$  is  $empty$ 
20: return  $SCG$ .
```

Figure 5. State class graph computation algorithm.

When a new state class is computed, in lines 7 and 8, and 17 and 18 of the algorithm, if it is not already contained in the state class graph, it is added to the state class graph. In lines 15 and 16 of the algorithm, according to Definition 11, the set of completable activities from the time closure class is obtained and the

successor state class of the current class with respect to completion of each of those activities is obtained. In order to do this, for the class (μ, D) , the successor of the following class is obtained:

$$C' := Successor((\mu, (D \cap \llbracket D_{min}(a) \leq v(a) \leq D_{max}(a) \rrbracket)), a),$$

which restricts the class (μ, D) to a subset of the valuation domain, D , where the activity is completable.

In generating the state class graph of TANs, the Parma polyhedra library (PPL) [33,34] for demonstrating and the required operations on the polyhedral domains, such as forward time closure, and successor, can be used. Convex polyhedra can be represented either as the set of solutions of a conjunction of linear inequalities or in a polar representation, called the system of generators [34]. As any operation on convex polyhedra may require a conversion from one representation to the other, all operations have a cost exponential with respect to the number of variables, both in terms of time and memory. Therefore, the complexity of computing each state class of the state class graph is at least exponential in the number of enabled activities. However, the complexity of the semi-algorithm is at least n times the complexity of computing a single state class, where n is the number of state classes in the state class graph.

The output of Algorithm 1 is a graph of all possible state of the system. Nodes of this graph are connected via edges labeled by activities. Therefore, untimed language of the TAN model can be extracted from this graph. Activities and state classes comprise the language's alphabets of a bounded TAN. If the notion of time is eliminated from the state space representation, we will not be able to model check timed reachability properties [27]. In order to illustrate this limitation, consider the TAN model of Figure 1. In this model, based on the state class graph, it can only be found out that the system has 7 state classes and the firing sequence $a_2, a_1, (a_3, a_4 \text{ or } a_4, a_3)$ as shown in Figures 6 and 7. After the completion of a_2 , a_1 can be completed at any time between 0 and 3. If it is completed at 0 the only possible completion sequence is a_3, a_4 . If a_1 is completed at 3, both a_3, a_4 and a_4, a_3 can happen. Therefore, based on the created graph, timed properties like this cannot be checked. Hence, an additional method should be utilized for further checking. The idea is to keep track of all progress variables such that no time information is lost.

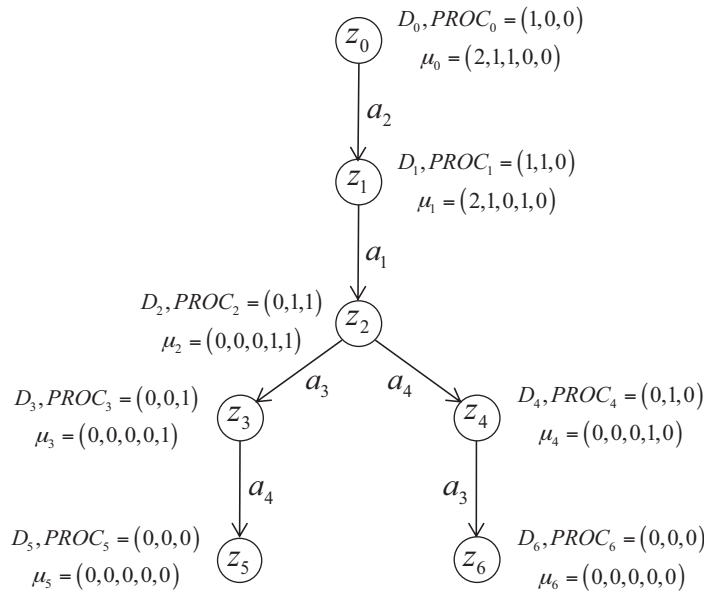


Figure 6. The state class graph of the TAN model in Figure 1.

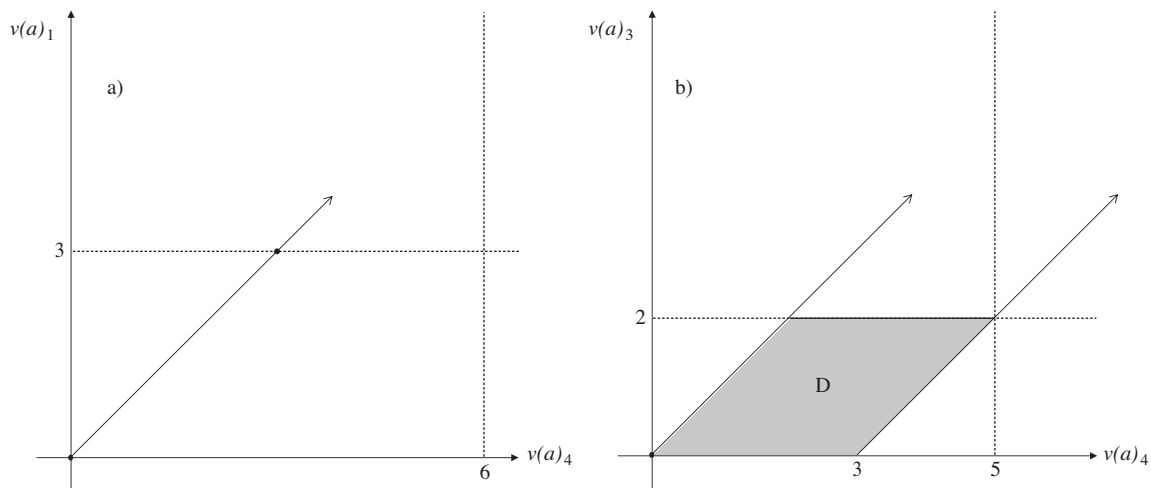


Figure 7. Computing the first state classes of the TAN model in Figure 1.

In Figure 7, the procedure for obtaining the first state classes of the TAN model in Figure 1 is shown.

5. Verification of TAN models

In this section, we describe possible verification and analysis techniques for TAN models. Two basic approaches can be employed for checking the time properties on an activity network based model: overloading and translation into automaton-based models. The former is done by using observers. The observer can be defined as a model that does not modify the behavior of the initial model, but changes the time properties checking issues to reachability checking concerns [35]. The important problem that hinders the use of observers is that for checking any property a specific observer is needed [36].

Therefore, in order to have a general approach for the verification of quantitative time properties (quantitative liveness and TCTL [6] properties), the translation approach remains useful. Especially when using TCTL as the verification framework, we are forced to translate our formalism into hybrid automata for which the existing model checking tools can be used. Translation notion can be used in 2 ways: structural translation and state space computation based translation [37]. In the case of the former method, different efforts have been conducted to translate various types of petri net-based formalisms, like TPN, into automaton-based models.

Cassez and Roux proposed a structural translation for TPN into TA that keeps the behavioral semantics unchanged [35]. In their approach, they have modeled each transition with one clock timed automaton, and so after translation we have the same number of simple automata as the number of transitions in the original TPN. The state of each transition is represented by the state of the corresponding timed automaton. The initial state of all automata is determined by the initial marking M_0 of TPN. The result is a composition of parallel automata supervised by an additional automaton. This automaton regulates state transition and behavior of all of the other automata. Moreover, other translation methods exist in the literature like Gu and Shin's method [38]. They have used time petri nets for modeling event-driven real-time systems, and have performed subsequent analysis via model checking by a translation into TA.

There exist other works done so far in this field, most of which have special purposes. Due to the similarity of TAN models to TPNs, the idea of the 2 methods mentioned above can be a useful inspiration for our translation, but these methods suffer from the large number of produced clocks. In the best case, the

number of clocks is the same as the number of activities. However, our focus is on state space based translation, especially the method firstly introduced by Lime and Roux in [32] and further completed in [7]. In their method, the state class graph is transformed into a hybrid automaton that accepts the same timed language as the original TPN. The most important advantage of this method is its simplicity and general usage, where translation is done once and all verifications can be done on it by using the existing techniques and tools.

We have employed the Parma polyhedral library in implementation of a tool for computation of the state class graph. All of the potential states that can be reached by the model are achieved, and so marking reachability along by time space reachability analyses can be done on its result by using untimed CTL. However, beside reachability analysis we are more interested in checking temporal properties that can be expressed by a standard temporal logic like LTL, CTL*, or TCTL. Using this notion we succeeded in translating our formalism into hybrid automata and exploit the existing tools like HYTECH [6].

Our approach towards using this method will be fully explained in the next section. Due to the similarity of TAN models with TPNs, based on the claimed experimental results of Lime and Roux, we expect to have much lower number of clocks than in the direct translation.

5.1. Verification using state class hybrid automata

In this subsection, we present our solution for model checking of TAN models using the state class hybrid automata notion introduced by Lime and Roux [32]. In the presence of the *PROC* function, the state class graph is translated into a time bisimilar linear hybrid automaton. Before going through it, let us have a glance at the definition of hybrid automaton and the required notations and concepts.

5.1.1. Linear hybrid automata

A hybrid automaton is a formal model for mixed discrete-continuous systems [10]. Indeed, hybrid automata are extensions of standard automata that add continuous variables to the original model.

Definition 14 (Hybrid automata) A hybrid automaton H consists of the following elements [10]:

- **Variables.** A finite set $X = \{x_1, x_2, \dots, x_n\}$ of real variables. n is called the dimension of H . $\dot{X} = \{\dot{x}_1, \dot{x}_2, \dots, \dot{x}_n\}$ is the set of derivatives of the variables in X representing the rate at which they change.
- **Control modes or locations.** There is a finite set of locations denoted by L .
- **Initial, invariant, and flow conditions.** These 3 functions assign to each location $l \in L$, the following 3 predicates:
 - The 2 predicates $init(l)$ and $inv(l)$, whose free variables are from X , determine the initial valuation of variables and the condition on the valuation of variables in location l , respectively.
 - The predicate $flow(l)$, whose free variables are from $X \cup \dot{X}$, determines the rate at which the variables change while the control mode is l .
- **Events.** A finite set Σ of alphabets of H .

- **Control switches or edges.** $E \subset L \times \varsigma(X) \times \Sigma \times 2^X \times X^X \times L$ is a finite set of edges. If $e = (l, \delta, \alpha, \Upsilon, \rho, l')$ is an edge of H from l to l' , then δ is its guard function, also known as the jump condition, α is relevant event occurred by this jump, Υ determines the set of clocks to be reset and ρ is the clock assignment function.

A hybrid automaton, H , is linear if the following conditions are met:

1. The predicates *init* and *inv* in each location $l \in L$ of H are Boolean combination of the linear inequalities.
2. The *flow* conditions governing the evolution of the variables X in each location $l \in L$ can be written as $A\dot{X} \leq B$, where A is a real matrix and B a real vector.

Since in the proposed model, based on the definition of the *PROC* function, the evolution of continuous variables depend only on the marking and ranges over Q^+ , then the *flow* predicate and also other constraints on continuous variables can be represented by linear expressions, independent of X and \dot{X} . Therefore, we are dealing with linear hybrid automata.

Definition 15 (*Semantics of hybrid automata*) *The semantics of a hybrid automaton H is defined as a TTS $S_H = (Q, Q_0, \xrightarrow{a})$, where $Q \subseteq L \times R^n$ is the set of states and $Q_0 \subseteq Q$ is the set of initial states of H , such that $\forall (l, v) \in Q, \text{inv}(l, v)$ is true and $\forall (l, v) \in Q_0$ both $\text{inv}(l, v)$ and $\text{init}(l, v)$ are true. Finally, \xrightarrow{a} is the transition function with the label $a \in \Sigma \cup R^+$, where*

- $(l, v) \xrightarrow{\sigma} (l', v')$ for $\sigma \in \Sigma$ if and only if there exists an edge $(l, \delta, \sigma, \Upsilon, \rho, l') \in E$ such that:
 - $\delta(v)$ is true,
 - $v' = v[\Upsilon][\rho]$, and
 - $\text{inv}(l', v')$ is true,
- $(l, v) \xrightarrow{t} (l, v')$ for each nonnegative real number t , where
 - $\forall x \in X; v'(x) = v(x) + t \times \dot{x}$, and
 - $\text{inv}(l, v')$ is true.

5.1.2. State class hybrid automata

In order to model check a TAN model, we keep the track of the time and variables' evolution in the generated state class graph. This idea is implemented via a linear hybrid automaton bisimilar to the original model, in which each location contains the corresponding location of the original graph information and other additional requirements. These additional requirements are the location's clocks and invariants of the related automata, which have the task of keeping the trace of time. Consequently, the extended state classes and the other related concepts should be mentioned and redefined here.

Definition 16 (Extended state class) An extended state class is a 4-tuple $(\mu, D, X, trans)$, where μ and D are the the marking and the domain, respectively. $X = X_T \cup \{x_I\}$ is the set of clocks, where X_T represents the set of clock corresponding to the enabled time activities and x_I is a virtual clock used for the enabled instantaneous activities. Finally, $trans \in (2^{TA \cup IA})^X$ is a mapping from X to the power set of $TA \cup IA$.

The function $trans(x)$ returns a set of enabled activities that are mapped onto the clock $x \in X$. Indeed, $trans$ assigns a set of activities to each $x \in X$, where x keeps the elapsed time for all of the activities in $trans(x)$ that have the same enabling time offset and progress rates. Considering that instantaneous activities have a zero completion time, by the completion of any instantaneous activity, a state class with a zero life time will exist. Accordingly, we add the clock x_I to our clocks; hence we have $trans(x_I) \in 2^{IA}$. This clock covers our need for instantaneous jumps in the bisimilar linear hybrid automata.

If in the original state class graph, 2 state classes, C and C' , are connected via a labeled edge e_a , and then there exist equivalent states and edges in the corresponding extended state class graph. Therefore, from the state class $C = (\mu, D, X, trans)$ via the edge e_a , C' will be reached, where $C' = (\mu', D', X', trans')$ is computed by Algorithm 2 and is shown in Figure 8.

Algorithm 2. SuccessorExtendedStateClass(C, a)

Input: Source extended state class C , activity a .

Output: The extended state class C' achieved by completion of activity a from C

- 1: μ' and D' are computed as before.
 - 2: $\forall x \in X, trans'(x) := trans(x) - \uparrow enabled(\mu, a) - disabled(\mu)$
 - 3: $\forall x \in X$ if $trans(x) = \emptyset; X := X - x$
 - 4: $\forall a \in \uparrow enabled(\mu, a) \cap IA; trans(x_I) := trans(x_I) \cup \{a\}$
 - 5: $\forall a \in \uparrow enabled(\mu, a) \cap TA$
 - 6: if $\exists x \in X \cdot val(x) = 0$ then
 - 7: $trans(x) := trans(x) \cup (\uparrow enabled(\mu, a) \cap TA)$
 - 8: else
 - 9: create clock x_i , where $val(x_i) = 0$, where i is the smallest available index in X
 - 10: $trans(x_i) := \uparrow enabled(\mu, a)$
 - 11: If the timed activities in $trans(x)$ have m different rates, then create $m - 1$ new variables with the smallest possible indices and add the related activities to the $trans$
 - 12: $\forall x \in X, \exists a \in trans(x)$, set $\dot{x} := PROC(\mu', a)$
-

Figure 8. Computing the extended state class C' achieved by the completion of activity a from C .

Since timed activities can have different rates, the sets of activities that are mapped to a single clock should have the same rate. Accordingly, if the timed activities in $trans(x)$ have m different rates, we create $m - 1$ new variables with the smallest possible indices and add the related activities to the function $trans$.

In the following, we briefly discuss the correctness and complexity of Algorithm 2. As the elapsed time for all of the disabled activities is not needed, in line 2 of the algorithm, for each disabled activity, its information is removed from the values of *trans* of all of the variables. In addition, the set of newly enabled activities are removed from the values of *trans* of all of the variables, and later in lines 5 through 10, all of the newly enabled activities are associated to a variable with valuation zero. Two cases can occur: in line 6, if a variable with valuation zero exists, all of the newly enabled activities are associated to that variable; otherwise, in line 10 of the algorithm, such a variable is added, and all of the newly enabled activities are associated to it. As for each variable $x \in X$, all of the activities in $trans(x)$ must have the same process rate. If after the completion of an activity, the timed activities in $trans(x)$ are partitioned into $m > 1$ groups, each with a different evolution rate, in lines 11 and 12 of the algorithm, $m - 1$ new variables are added and each of the m variables are associated with the activities in 1 group.

In computing the new valuation domain D' , in line 1 of the algorithm, the polyhedral operations are used. Therefore, the complexity of computing the new valuation domain is exponential in the number of activities. Consequently, since the rest of the algorithm is polynomial, the complexity of Algorithm 2 is exponential in the number of activities. However, since the translation procedure requires a repetitive application of Algorithm 2 (for each state class), it has n times the complexity of Algorithm 2, where n is the number of state classes of the state class graph.

Definition 17 (Clock similarity) Two extended state classes $C = (\mu, D, X, trans)$ and $C' = (\mu', D', X', trans')$ are clock-similar if they have the same marking and the same number of clocks that can be mapped by the following one-to-one function between them:

$$C \approx C' \Leftrightarrow \begin{cases} \mu = \mu' \\ |X| = |X'| \\ trans(x_I) = trans(x'_I) \\ \forall x \in X_T, \exists x' \in X'_T; \begin{cases} trans(x) = trans'(x') \\ \dot{x} = \dot{x}' \\ init(x) = init(x') \end{cases} \end{cases},$$

where $init(x)$ is the value of x at the moment that the state class C is created.

Definition 18 (Extended state class inclusion) $C' = (\mu', D', X', trans')$ is included in $C = (\mu, D, X, trans)$ and is shown by $C' \subseteq C$, if and only if $D' \subseteq D$ and $C \approx C'$.

After this definition and having extended the state class graph, we are ready to present the method for the creation of the state class hybrid automata for TANs.

Definition 19 (State class hybrid automata) A state class hybrid automaton is 7-tuple $H = (L, l_0, X, \Sigma, E, Inv, Flow)$, where

- L is the set of locations obtained from the set C_{ext} (the set of all of the extended state classes),
- $l_0 = C_0 = (\mu_0, D_0, X_0, trans_0)$ is the initial location,
- $X = \bigcup_{C=(\mu, D, X, trans) \in C_{ext}} X$,

- $\Sigma = TA \cup IA$.
- E is the set of edges, where

$$\forall C_i = (\mu_i, d_i, X_i, trans_i), C_j = (\mu_j, d_j, X_j, trans_j) \in C_{ext},$$

$$\exists e = (l_i, \delta, \sigma, \Upsilon, \emptyset, l_j) \in E \Leftrightarrow \begin{cases} \delta = \begin{cases} x_I = 0, & a \in IA \\ trans_i^{-1}(a) \geq D_{min}(a), & a \in TA \end{cases} \\ \sigma = a \\ \Upsilon = trans^{-1}(\uparrow enabled(\mu_i, a)) \\ C_i \xrightarrow{a} C_j \end{cases} .$$

- $\forall l = (\mu, D, X, trans) :$

$$inv(l) = \begin{cases} \bigwedge_{\forall x \in X, a \in trans(x)} (x \leq D_{max}(a)), & trans(x_I) \neq \emptyset \\ x_I = 0, & otherwise \end{cases} .$$

- $\forall l = (\mu, D, X, trans) :$

$$\forall x \in X ; \dot{x} = Flow(l)(x) = \begin{cases} PROC(a, \mu), & x \in X_T \wedge \exists a \in trans(x) \\ 1, & x = x_I \wedge trans(x_I) \neq \emptyset \end{cases} .$$

Configuration of the state class hybrid automata can be represented by $(l = (\mu_H, D_H, X_H, trans_H), v_H)$, where the vector v_H shows the values of the clocks.

5.2. Soundness of the translation

In this section, we prove the soundness of the translation from the TANs into a bisimilar hybrid automaton. For this purpose, in the following, we will show that the original TAN model is bisimilar with its corresponding state class hybrid automaton.

Definition 20 (Bisimulation) Let $S_1 = (Q_1, Q_0^1, \xrightarrow{a} \rightarrow_1)$ and $S_2 = (Q_2, Q_0^2, \xrightarrow{a} \rightarrow_2)$ be 2 timed transition systems and let R be a binary relation over $Q_1 \times Q_2$. We write $q_1 R q_2$ for $(q_1, q_2) \in R$, where $q_1 \in Q_1$ and $q_2 \in Q_2$. R is a bisimulation if:

- $q_1 R q_2$ and $q_1 \rightarrow q_1'$ implies that $\exists q_2' \in Q_2$, where $q_2 \rightarrow q_2'$ and $q_1' R q_2'$.
- Symmetrically, if $q_1 R q_2$, $\forall q_2' \in Q_2$, where $q_2 \rightarrow q_2'$ implies that $\exists q_1' \in Q_1$, where $q_1 \rightarrow q_1'$ and $q_1' R q_2'$.

Theorem 1 (Bisimulation of TAN and the state class HA) Let Q_T be the set of states of a TAN T and let Q_H be the set of states of its corresponding state class hybrid automaton $H = (L, l_0, X, \Sigma, E, Inv, Flow)$. We define a binary relation R over $Q_T \times Q_H$. Such that for all $q_T = (\mu_T, v_T) \in Q_T$ and $q_H = (l = (\mu_H, D_H, X_H, trans_H), v_H) \in Q_H$, $q_T R q_H$, if and only if:

1. $\mu_H = \mu_T$.
2. $\forall a \in enabled(\mu_T) \cap IA, a \in trans(x_I)$.
3. $\forall a \in enabled(\mu_T) \cap TA, \exists x \in X_T, v_T(a) = v_H(x)$ and $Flow(l)(x) = PROC(\mu_T, a)$.

R is bisimulation.

Proof Based on the previous definition, the proof of bisimulation is a 2 way procedure. Suppose that $q_T Rq_H$, then:

Step 1. If $q_T \xrightarrow{\sigma} q'_T, \exists q'_H$, where $q_H \xrightarrow{\sigma} q'_H$ and $q'_T Rq'_H$.

- If $\sigma=t \in R^+$, so it is a continuous transition in the TAN ($\mu_T = \mu'_T$), so $enabled(\mu) \cap IA = \emptyset$ and $\forall a \in enabled(\mu_T) \cdot (v_T(a) + t \times PROC(\mu_T, a)) \leq D_{max}(a)$. Hence:

$$- \forall a \in enabled(\mu_T) \cdot v_T(a) = v_H(x) \Rightarrow v'_H(x) = (v_H(x) + t \times PROC(\mu_T, a)) \leq D_{max}(a) \Rightarrow inv(l, v_H + \dot{X} \times t) = true.$$

- Hence, H can have a continuous transition with the time elapse t . Due to the fact that H is evolving in the same location, the extended state is the same, and so is the related marking ($\mu_H = \mu'_H$). Therefore, $Flow(l)(x) = PROC(\mu_T, a)$ stays unchanged and results in $q'_T Rq'_H$.

- $\sigma = a \in IA$, so an instantaneous activity a is enabled in q_T . That means $a \in trans(x_I)$ and $inv(l) = \llbracket x_I = 0 \rrbracket$, so:

- $\exists q'_H \cdot q_H \xrightarrow{\sigma} q'_H$. Since q_T and q_H have the same marking and have the same rules for finding new a marking from the same activity completion, then q'_H and q'_T have the same marking ($\mu'_T = \mu'_H$).

- Based on Algorithm 2, after the completion of a if $\exists a' \in \uparrow enabled(\mu_T, a) \cap IA$, then $trans'(x_I) = trans(x_I) \cup a'$, so: $\forall a \in enabled(\mu'_T) \cap IA, a \in trans'(x_I)$.

- $\forall a' \in \uparrow enabled(\mu_T, a) \cap TA$, a' is assigned to a clock $x' \in X_T$ with the same value and progress rate ($v'_T(a') = v_H(x)$ and $Flow(l')(x') = PROC(\mu'_T, a')$).

- As a result of this, we have $q'_T Rq'_H$.

- $\sigma = a \in TA$, so a timed activity a is enabled in q_T (and $enabled(\mu) \cap IA = \emptyset$), so the edge e with the jump condition $trans^{-1}(a) \geq \alpha(a)$ exists in H that led to q'_H . Since the approach for finding the marking, the clock's rate, and the values does not differ for $a \in TA$ or $a \in IA$, like before, it can be easily proved that $q'_T Rq'_H$.

Step 2. In this step, it should be proved that if $q_H \xrightarrow{\sigma} q'_H$ then $\exists q'_T, q_T \xrightarrow{\sigma} q'_T$ and $q'_T Rq'_H$.

- $\sigma=t \in R^+$, so it is a continuous transition in H ($\mu_H = \mu'_H$ and so $\mu'_H = \mu_T$). Based on the definition of the state class hybrid automata, we have:

$$- enabled(\mu) \cap IA = \emptyset$$

$$- inv(l') = true \Rightarrow$$

$$\left\{ \begin{array}{l} \forall a \in enabled(\mu'_H = \mu_T) \cap trans(x) \cdot v'_H(x) = (v_H(x) + t \times Flow(l)(x)) \leq D_{max}(a) \\ Flow(l)(x) = PROC(a, \mu'_H = \mu_T) \\ q_T Rq_H \Rightarrow \forall a \in enabled(\mu_T) \cap trans(x) \cdot v_H(x) = v_T(a) \end{array} \right.$$

$\Rightarrow \forall a \in enabled(\mu_T) \cdot (v_T(a) + t \times PROC(\mu_T, a)) \leq D_{max}(a)$, which means that T can have a continuous transition with the time elapse t . In other words, $\exists q'_T, q_T \xrightarrow{\sigma=t} q'_T$. Based on the definition of continuous transition in the TAN, $\mu_T = \mu'_T$ and obviously $\mu'_T = \mu'_H$.

- In addition, $enabled(\mu_T) = enabled(\mu'_T)$, which means X_T stays unchanged, so condition 3 of Theorem 1 is also satisfied, and accordingly $q'_T R q'_H$.

• $a \in IA$, so based on the definition of the state class hybrid automata, we have: $\exists C_i \xrightarrow{a} C_j \cdot a \in IA$. Hence:

- $\exists a \in IA \cap enabled(\mu_H = \mu_T) \Rightarrow \exists q_T \xrightarrow{\sigma=a} q'_T$. Since q_T and q_H have the same marking and have the same rules for finding new markings from same activity completion, then q'_H and q'_T will have the same marking ($\mu'_T = \mu'_H$).

- Based on Algorithm 2, after the completion of a , if $\exists a' \in \uparrow enabled(\mu_T, a) \cap IA$, then: $trans'(x_I) = trans(x_I) \cup a'$, so: $\forall a \in enabled(\mu'_T) \cap IA, a \in trans'(x_I)$.

- $\forall a' \in \uparrow enabled(\mu_T, a) \cap TA$, a' is assigned to a clock $x' \in X_T$ with the same value and progress rate, which means:

$$\forall a' \in enabled(\mu'_T) \cdot \exists x' : v'_T(a') = v_H(x') \text{ and } Flow(l')(x') = PROC(\mu'_T, a').$$

- As a result, we have $q'_T R q'_H$.

• $\sigma = a \in TA$, so based on the definition of the state class hybrid automata $\exists C_i \xrightarrow{a} C_j \cdot a \in TA$, so the edge e with the jump condition $trans^{-1}(a) \geq \alpha(a)$ exists in H , which leads to q'_H . Since the approach for finding the marking, clock's rate, and values from state q_T does not differ for $a \in TA$ or $a \in IA$, like before, it can easily be proved that $q'_T R q'_H$.

□

In this section, we proved that using the proposed approach, a given TAN model can be translated into a bisimilar hybrid automaton that accepts the same timed language. The resulting automata can be used for the further checking of timed properties, which was impossible for the original TAN.

6. Related works and comparison

TA [4] may be the most successful model of real-time systems. An extension of TA, stopwatch automata (SWA) are TA augmented with stopwatches that can be stopped and resumed. In [39], the expressive power of SWA with an unobservable behavior was investigated and it was proved that it has the same expressive power as linear hybrid automata (LHA) in the sense that any finite or infinite timed language accepted by an LHA is also acceptable by a SWA.

Suspension automata [28] are another extension of TA with a bounded subtraction, in which the clocks may be updated by subtractions within a bounded zone. If all of the suspension durations are fixed and integral, the model is a decidable class of TA. The approach of suspension automata is further extended in task automata

[29]. Task automata are extensions of TA for the schedulability analysis of real-time systems with nonuniformly recurring computation tasks that are triggered by events. A task is an executable program characterized by its best-case and worst-case execution time, deadline, priority, and so on.

Several extensions of petri nets have been used for specifying timed systems [27,40,41]. Time petri net (TPN) [40] is a classical timed extension of petri nets, in which a firing interval is associated with each transition. This extension takes into account the scheduling of the software tasks that are assigned to the processors of a multiprocessor system. Moreover, in [27,32], a scheduling-TPN for schedulability verification was proposed that is appropriate for specifying different scheduling policies. In [27], when n tasks with the same priority share a single processor in a round-robin policy, the evolution rate of the corresponding variables was $1/n$. In order to model round-robin scheduling easily, in the approach proposed in [42], groups of transitions were defined together with execution speeds. The transitions in a group can be executed at the same time and each rate is divided by the sum of the execution speeds.

TANs inherit all of the advantages of petri nets, such as the ability to capture behaviors, including concurrency, synchronization, priorities, and conflicts. Due to the usage of the input and output gates, the TAN models, proposed in this paper, are more flexible than the petri net-based models mentioned above. In addition, due to the marking-dependent process rate functions of TANs, these models are more flexible than the models introduced in [27,32,42].

7. Conclusions

In this paper, we introduced TANs, based on the well-known SANs, for the modeling and verification of a timed system. We presented the informal and formal definitions, semantics, and analysis techniques of TAN models.

TANs take the advantages of activity networks, which are using the input/output gates to construct more compact petri net-based models for timed systems. In addition, TANs allow the assignment of time intervals to timed activities. Despite petri net-based timed formalisms, TANs are capable of the easily modeling of systems where the activity process rates are state dependent. Examples of such systems are stream processing systems and scheduling systems. On the other hand, due to the naivety of the enabling and firing rules in petri net-based models, describing many systems using TAN models is much easier than petri net-based formalisms. Thanks to the presence of the input/output gates in TANs, this formalism is capable of modeling systems with complicated enabling rules and state-dependent activity completion rates. These features make TANs a better model for timed systems.

A great benefit of this formalism is the similarity of its basis with SAN models, which allows easily for obtaining or converting the existing SAN models to TAN models, which are appropriate for timing analysis.

As we presented in this paper, TAN models can be translated into LHA. Therefore, the existing model checking techniques and tools can be employed with TAN models after their transformation into LHA models.

We are currently developing a tool for the direct model checking of TANs. In future, we intend to extend TANs with some other useful features, such as time-dependent activity rates, for better application on modeling timed systems.

Acknowledgments

This research was supported by the Research Institute for ICT of Iran.

References

- [1] A. Movaghar, "Stochastic activity networks: a new definition and some properties", *Scientia Iranica*, Vol. 8, pp. 303–311, 2001.
- [2] M. Uzam, A.H. Jones, "A new Petri-net-based synthesis technique for supervisory control of discrete event systems", *Turkish Journal of Electrical Engineering & Computer Sciences*, Vol. 10, pp. 85–109, 2002.
- [3] R. Samet, O.F. Duman, "Behaviors of real-time schedulers under resource modification and a steady scheme with bounded utilization", *Turkish Journal of Electrical Engineering & Computer Sciences*, Vol. 18, pp. 1115–1130, 2010.
- [4] R. Alur, D. Dill, "Automata for modelling real-time systems", *Lecture Notes in Computer Science*, Vol. 443, pp. 322–335, 1990.
- [5] G. Behrmann, J. Bengtsson, A. David, K. Larsen, P. Pettersson, W. Yi, "UPPAAL implementation secrets", *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*, pp. 3–22, 2002.
- [6] T.A. Henzinger, P. Ho, H. Wong-Toi, "HyTech: a model checker for hybrid systems", *Proceedings of the 6th International Conference on Computer Aided Verification*, pp. 460–463, 1997.
- [7] F. Laroussinie, K. Larsen, "CMC: a tool for compositional model-checking of real-time", *Proceedings of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification*, pp. 439–456, 1998.
- [8] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, S. Yovine, "Kronos: A model-checking tool for real-time systems", *Proceedings of the 10th International Conference on Computer Aided Verification*, pp. 546–550, 1998.
- [9] J. Srba, "Comparing the expressiveness of timed automata and timed extensions of Petri nets", *Proceedings of the 6th International Conference on Formal Modelling and Analysis of Timed Systems*, pp. 15–32, 2008.
- [10] T. Henzinger, "The theory of hybrid automata", *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pp. 278–292, 1996.
- [11] P. Merlin, "A study of recoverability of computing systems", PhD Thesis, Department of Computer Science, University of California, Irvine, 1974.
- [12] P. Merlin, D. Farber, "Recoverability of Communication protocols: implications of a theoretical study", *IEEE Transactions on Communications*, Vol. 24, pp.1036–1043, 1976.
- [13] B. Bérard, F. Cassez, S. Haddad, D. Lime, O. Roux, "Comparison of the expressiveness of timed automata and time Petri nets", *Proceedings of the 3rd International Conference on Formal Modelling and Analysis of Timed Systems*, pp. 211–225, 2005.
- [14] M. Abdollahi Azgomi, A. Movaghar, "A modelling tool for hierarchical stochastic activity networks", *Simulation Modelling Practice and Theory*, Vol. 13, pp. 505–524, 2005.
- [15] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets*, New York, Wiley, 1995.
- [16] A. Abbarin, M. Firuzabad, A. Özdemir, "An extended component-based reliability model for protective systems to determine routine test schedule", *Turkish Journal of Electrical Engineering & Computer Sciences*, Vol. 19, pp. 303–315, 2011.
- [17] M. Cakiroğlu, A. Özcerit, "Design and evaluation of a query-based jamming detection algorithm for wireless sensor networks", *Turkish Journal of Electrical Engineering & Computer Sciences*, Vol. 19, pp. 1–19, 2011.
- [18] W.H. Sanders, W. Obal II, M. Qureshi, F. Widjanarko, "The UltraSAN modeling environment", *Performance Evaluation*, Vol. 24, pp. 89–115, 1995.
- [19] D.D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. Doyle, W.H. Sanders, P. Webster, "The Möbius framework and its implementation", *IEEE Transactions on Software Engineering*, Vol. 28, pp. 956–969, 2002.

- [20] A. Movaghar, J.F. Meyer, “Performability modeling with stochastic activity networks”, Proceedings of the 1984 Real-Time Systems Symposium, pp. 215–224, 1984.
- [21] A. Movaghar, “Performability modeling with stochastic activity networks”, PhD Dissertation, Department of Electrical Engineering, University of Michigan, 1985.
- [22] M. Abdollahi Azgomi, A. Movaghar, “An introduction to high-level stochastic activity networks” International Reviews on Computers and Software, Praise Worthy Prize, Vol. 1, pp. 20–30, 2006.
- [23] M. Abdollahi Azgomi, A. Movaghar, “Modeling and evaluation with object stochastic activity networks”, Proceedings of the 1st International Conference on Quantitative Evaluation of Systems, pp. 326–327, 2004.
- [24] W.H. Sanders, J.F. Meyer, “Stochastic activity networks: formal definitions and concepts”, Lecture Notes in Computer Science, Vol. 2090, pp. 315–343, 2001.
- [25] M. Abdollahi Azgomi, A. Movaghar, “A modeling tool for a new definition of stochastic activity networks”, Iranian Journal of Science and Technology, Transaction B (Technology), Vol. 29, pp. 79–92, 2005.
- [26] A. Khalili, A. Jalaly Bidgoly, M. Abdollahi Azgomi, “PDETool: a multi-formalism modeling tool for discrete-event systems based on SDES description”, Lecture Notes in Computer Science, Vol. 5606, pp. 343–352, 2009.
- [27] D. Lime, O. Roux, “Formal verification of real-time systems with preemptive scheduling”, Real-Time Systems, Vol. 41, pp. 118–151, 2009.
- [28] J. McManis, P. Varaiya, “Suspension automata: a decidable class of hybrid automata”, Proceedings of the 6th International Conference on Computer Aided Verification, pp. 105–117, 1994.
- [29] E. Fersman, P. Krcál, P. Pettersson, W. Yi, “Task automata: schedulability, decidability and undecidability”, Information and Computation, Vol. 205, pp. 1149–1172, 2007.
- [30] O. Roux, D. Lime, “Time Petri nets with inhibitor hyperarcs: formal semantics and state space computation”, Proceedings of the 25th International Conference on Applications and Theory of Petri Nets, pp. 371–390, 2004.
- [31] T. Henzinger, Z. Manna, A. Pnueli, “Timed transition systems”, Proceedings of the REX Workshop Real-Time: Theory in Practice, Lecture Notes in Computer Science, Vol. 600, pp. 226–251, 1992.
- [32] D. Lime, O. Roux, “Model-checking of time Petri nets using the state class timed automaton”, Discrete Event Dynamic Systems, Vol. 16, pp. 179–205, 2006.
- [33] R. Bagnara, P. Hill, E. Zaffanella, “The Parma polyhedra library: toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems”, Science of Computer Programming, Vol. 72, pp. 3–21, 2008.
- [34] R. Bagnara, P.M. Hill, E. Zaffanella, “The Parma polyhedra library user’s manual”, Department of Mathematics, University of Parma, Parma, Italy, Version 0.11.2, Available: <http://www.cs.unipr.it/ppl/>, 2011.
- [35] F. Cassez, O. Roux, “Structural translation from time Petri nets to timed automata”, Journal of Systems and Software, Vol. 79, pp. 1456–1468, 2006.
- [36] J. Toussaint, F. Simonot-Lion, J. Thomesse, “Time constraints verification methods based on time Petri nets”, Proceedings of the 6th IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems, pp. 262–269, 1997.
- [37] D. Lime, O. Roux, C. Seidner, L. Traonouez, “Romeo: a parametric model-checker for Petri nets with stopwatches”, Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 54–57, 2009.
- [38] Z. Gu, K.G. Shin, “Analysis of event-driven real-time systems with time Petri nets: a translation-based approach”, Proceedings of the IFIP 17th World Computer Congress - TC10 Stream on Distributed and Parallel Embedded Systems: Design and Analysis of Distributed Embedded Systems, pp. 31–40, 2002.
- [39] F. Cassez, K. Larsen, “The impressive power of stopwatches”, Proceedings of the 11th International Conference on Concurrency Theory, pp. 138–152, 2000.

- [40] O. Roux, A. Déplanche, “A T-time Petri net extension for real-time task scheduling modelling”, *European Journal of Automation*, Vol. 36, pp. 973–987, 2002.
- [41] O. Roux, D. Lime, “Time Petri nets with inhibitor hyperarcs: formal semantics and state space computation”, *Proceedings of the 25th International Conference on Applications and Theory of Petri Nets*, Vol. 3099, pp. 371–390, 2004.
- [42] Y. Okawa, T. Yoneda, “Schedulability verification of real-time systems with extended time Petri nets”, *International Journal of Mini and Microcomputers*, Vol. 18, pp. 148–156, 1996.