RT-Studio: A tool for modular design and analysis of realtime systems using Interpreted Time Petri Nets

Rachid Hadjidj and Hanifa Boucheneb

Abstract. RT-Studio (Real Time Studio) is an integrated environment for modeling, simulation and automatic verification of realtime systems modeled as networks of Interpreted Time Petri Nets (ITPNs). The tool allows to construct several abstractions for ITPNs suitable to verify reachability, linear and branching properties, in addition to a TCTL model checker. In this paper we describe the ITPN model as a proposed extension to the classical TPN model and describe the modular modeling capabilities of RT-Studio. The classical railroad crossing model is used to illustrate these capabilities.

Keywords: Formal verification, Time Petri Nets, timed properties, model checking, simulation

1 Introduction

Interpreted Time Petri Nets (ITPNs) are Time Petri Nets (TPNs) [15, 8, 11] we extended with bounded data variables to increase their modeling power and expressiveness. TPNs are Petri nets extended with temporal constraints in the form of time intervals associated with their transitions [15]. A transition can fire if it is enabled and if the time elapsed since it became enabled most recently is within its time interval, but must be fired or disabled if this elapsed time reaches the upper bound of its time interval. With this extension, TPNs are powerful enough to model realtime systems. However, their analysis is much more complicated than simple Petri nets. In fact, the state space of the TPN model is in general infinite due to the continuous aspect of time. Furthermore boundedness is undecidable for this model [5, 4]. Hopefully, a subclass of TPNs called bounded TPNs, for which reachability is decidable, allows to model most useful systems. The analysis and verification of the TPN model is generally performed using model checking techniques. Model checking requires first to represent the behavior of a system as a finite *state transition system*[1], then specify properties of interest in a temporal logic (LTL, CTL, CTL*,...), and finally explore the state transition system to determine whether these properties hold or not [8, 11–13].

In the field of realtime systems modeling and verification, UPPAAL is one of the best tools available [3]. UPPAAL allows for a modular design of realtime systems using an extension of the Timed Automata (TA) model [2] instead of the TPN model. In this extension, variables can be defined, and both guards and actions can be associated with transitions. In addition to its modeling capabilities, UPPAAL integrates an efficient on the fly model checker (a key success component) for a subset of TCTL [1].

[1] Also called *state graph* or *state space*.

For the TPN model, many tools exist for modeling, analysis and verification. Some well known tools are *Roméo* [10] and *TINA* [6]. But to our knowledge, there is no tool for the TPN model that compares to UPPAAL in terms of modular design, extension with variables and verification performance. TAPAAL2 [9], as an inspiration from UPPAAL, is an interesting reascent addition to the pool of tools dedicated to the modeling and verification of real time systems using a timed extension of Petri Nets. In this extension clocks are associated with tokens and time intervals with arcs. A token cannot pass trough an arc unless its clock is within the time interval of the arc. Time invariants associated with places allow to model urgency. The tool, has good verification performance and a nice graphical user interface capable of component based design. Even if it is a great step forward toward tightening the gap with UPPAAL in terms of providing a similar user experience and modeling power, UPPAAL still has a more powerful modular design capability in addition to the use of variables which TAPAAL2 lacks. RT-Studio presented here, attempts to tighten the gap with UPPAAL even more, by allowing for a similar modular design capability based on the ITPN model. RT-Studio also allows to simulate these models and analyze them using several means. In the ITPN model, each transition has, in addition to a time interval, a guard and a set of update operations on user defined data variables and places markings. An enabled transition can be fired if it is enabled and the guard associated with it is true in the current state of the model. When a transition is fired, its updates are applied.

2 Interpreted Time Petri Nets

To increase the modeling power and expressiveness of TPNs , we extended the TPN model with bounded integer variables and associate guards and update actions with transitions. The resulting model is called Interpreted TPN (ITPN). As a consequence, in addition to the normal transition firing requirements, a transition requires also that its guard is true in the current ITPN state. A guard is a condition on the ITPN state defined on its marking, firing intervals of transitions, and the state of associated variables. After a transition t is fired, associated updates are performed. Updates can target variables associated with the ITPN, but also its marking. An update can only target places not attached to t for not conflicting with the usual operational semantics of transition firing. With the new extension, inhibitor arcs and priority on transitions firing can be modeled. Note that if variables associated with an ITPN are always positive then they can be implemented as normal places.

Let $\mathbb{Q}^+$, $\mathbb{R}^+$ and $\mathbb{Z}$ be respectively the set of positive rational numbers, the set of positive real numbers and the set of integers. Let $\mathbb{Q}^+_{[\,]}$ be the set of non empty intervals of $\mathbb{R}^+$ which bounds are respectively in $\mathbb{Q}^+$ and $\mathbb{Q}^+ \cup \{\infty\}$. For an interval $I \in \mathbb{Q}^+_{[\,]}$, $\downarrow I$ and $\uparrow I$ denote respectively its lower and upper bounds.

Definition 1. ***Interpreted Time Petri Net (ITPN)***
An ITPN $\mathcal{P}$ is a tuple $(P, T, V, Pre, Post, m_0, v_0, Is, G, U)$ where:

- *P is a finite set of places,*
- *T is a finite set of transitions, with $P \cap T = \emptyset$,*
- *V is a finite set of integer variables, with $(P \cup T) \cap V = \emptyset$,*

- Pre *and* $Post$ *are respectively the backward and forward incidence functions:* $P \times T \to \mathbb{N}$*, where* $\mathbb{N}$ *is the set of nonnegative integers,*
- $m_0 : P \to \mathbb{N}$*, is the initial marking,*
- $v_0 : V \to \mathbb{Z}$*, is the initial valuation on* $\mathcal{P}$ *variables.*
- $Is : T \to \mathbb{Q}^+_{[\,]}$ *associates with each transition* t *an interval* $[\downarrow Is(t), \uparrow Is(t)]$ *called its* static firing interval*. The bounds* $\downarrow Is(t)$ *and* $\uparrow Is(t)$ *are called the* minimal *and* maximal static firing delays *of* t*.*
- $G : T \to Bool(\mathcal{P})$*, associate with each transition a guard from the set* $Bool(\mathcal{P})$*.* $Bool(\mathcal{P})$ *is the set of boolean functions on the set* $\mathcal{S}_\mathcal{P}$ *of states of* $\mathcal{P}$ *(The ITPN state is defined next).*
- $U : T \to Update(\mathcal{P})$*, where* $Update(\mathcal{P})$ *is the set of integer functions on the set* $\mathcal{S}_\mathcal{P} \times (P \cup V)$*, associates with each transition* t *an update function on places and variables associated with* $\mathcal{P}$*.*

Let M be the set of all markings of $\mathcal{P}$. Let $m \in M$ be a marking, and $t \in T$ a transition of $\mathcal{P}$. t is said to be *enabled* in m, iff all tokens required for its firing are present in m, i.e.: $\forall p \in P, m(p) \geq Pre(p,t)$. We denote by $En(m)$ the set of all transitions enabled in m. If m results from firing transition t_f from another marking, $New(m, t_f)$ denotes the set of all newly enabled transitions in m, i.e.: $New(m, t_f) = \{t \in En(m) | \exists p, m(p) - Post(p, t_f) < Pre(p, t)\}$.

Definition 2. ***ITPN state***
The state of ITPN $\mathcal{P}$ *is a couple* (m, v, I)*, where* m *is a marking,* v *is a valuation on the variables of* $\mathcal{P}$*, and* I *is an interval function* $I : En(m) \to \mathbb{Q}^+_{[\,]}$ *[5].*

For a state $s = (m, v, I)$, and $t \in En(m)$, $I(t)$ is called the *firing interval* of t. It is the interval of time where t can fire. The initial state of $\mathcal{P}$ is $s_0 = (m_0, v_0, I_0)$, where $I_0(t) = Is(t)$, for all $t \in En(m_0)$. The state of $\mathcal{P}$ evolves either by time progression or by firing transitions. When a transition t becomes enabled, its firing interval is set to its static firing interval $Is(t)$. The bounds of this interval decrease synchronously with time, until t is fired or disabled by another firing. t can fire, if the lower bound $\downarrow I(t)$ of its firing interval reaches 0 and its guard $G(t)$ evaluates to true in the current state of $\mathcal{P}$, but must be fired, without any additional delay if the upper bound $\uparrow I(t)$ of its firing interval reaches 0 while its guard evaluates true. The firing of a transition takes no time.

Let $s = (m, v, I)$ and $s' = (m', v', I')$ be two states of $\mathcal{P}$. We write $s \xrightarrow{\theta} s'$, iff state s' is reachable from state s after a time progression of θ time units (s' is also denoted $s + \theta$), i.e.:

$$\begin{cases} \exists \theta \in \mathbb{R}^+, \ \bigwedge_{t \in En(m)} \theta \leq \uparrow I(t), \\ m' = m, v' = v, \\ \forall t' \in En(m'), \ I'(t') = [max(\downarrow I(t') - \theta, 0), \uparrow I(t') - \theta]. \end{cases}$$

We write $s \xrightarrow{t} s'$ iff state s' is immediately reachable from state s by firing transition t. i.e.:

$$\begin{cases} t \in En(m), \\ \downarrow I(t) = 0 \wedge G(t,s) = true, \\ \forall p \in P \begin{cases} m'(p) = m(p) - Pre(p,t) + Post(p,t) \; if \; Pre(p,t) \neq 0 \vee Post(p,t) \neq 0, \\ m'(p) = U(t,s,p) \qquad\qquad\qquad\qquad\quad otherwise, \end{cases} \\ \forall x \in V, \; v'(x) = U(t,s,x), \\ \forall t' \in En(m') \begin{cases} I'(t') = Is(t') \; if \; t' \in New(m',t), \\ I'(t') = I(t) \quad otherwise. \end{cases} \end{cases}$$

Definition 3. ***ITPN state space***
The state space of an ITPN *model $\mathcal{P}$ is the structure $(\mathcal{S}, \rightarrow, s_0)$, where:*

- *$s_0 = (m_0, v_0, I_0)$ is the initial state of $\mathcal{P}$,*
- *$s \rightarrow s'$ iff either $s \xrightarrow{\theta} s'$ for some $\theta \in \mathbb{R}^+$ or $s \xrightarrow{t} s'$ for some $t \in T$,*
- *$\mathcal{S} = \{s | s_0 \xrightarrow{*} s\}$, where $\xrightarrow{*}$ is the reflexive and transitive closure of $\rightarrow$, is the set of reachable states of $\mathcal{P}$.*

3 Modular design of realtime system

RT-Studio uses the concept of programming project[2] to allow for a modular design of a realtime system. A project is basically a collection of ITPN components, and a *system description file*. An ITPN component is actually a template with possible parameters. The system description file is where the system configuration is specified. It consists of a list shared variable declarations and instantiations of ITPN components.

Instances of ITPN components within the same project can communicate and synchronize using data variables defined in the system description file, and using shared transitions and shared places. A transition or a place is shared between several ITPNs if it has the same name and set as external (not local) in each one of them. When synchronizing ITPN components instances, shared places with the same name are merged in one single place. The marking of that place is the maximum marking of synchronized places. For transitions, the synchronization is little bit more demanding. A transition meant for synchronization is implicitly associated with a signaling channel having the same name. Transitions which names end with and exclamation mark '!' represent send actions on associated channels. Those with names ending with an interrogation mark '?' represent receiving actions on associated channels. When instances of ITPN components are synchronized, transitions representing complimentary actions are synchronized by merging each sending transition with a copy of a receiving transition having the same name. A copy of a transition is created by duplicating the transition and all its ingoing and outgoing arcs.

Each ITPN component element (place, transition, arc), including the ITPN itself has an extendable list of attributes (properties). When an ITPN component or an ITPN component element is created, a default list of attributes is associated with it. These attributes describe its different features, like the number of tokens for a place, the guard and update actions for a transition, the list of parameters for the ITPN component,

[2] A collection of files.

including the visual appearance of ITPN elements like the color, the shape, the border size, etc. The *Attribute viewer* is a panel for editing, modify and adding new attributes for the currently selected ITPN element. Added attributes can be referred to in guards and updates actions of transitions. Three attribute types are supported in the current version of RT-Studio: number (real value), boolean and string.

To analyze a system designed as a project, its system description file need to be compiled. The compilation consists in synchronizing all ITPN components instances in one single ITPN. In the case of any error, error messages are displayed in a message pane at the bottom of the main window, otherwise the synchronized ITPN is generated and displayed in a separate panel and ready to be analyzed if it is self contained[3]. Note that a self contained ITPN components can be analyzed without compilation.

4 Functionality

RT-Studio's verification and analysis capabilities can be grouped in three categories: abstract state spaces construction, model checking and simulation. Both known characterizations of the TPN state (interval and clock characterizations) [11] can be used to construct abstractions for ITPN models. For the interval characterization, computed abstractions are: the classical State Class Graph (SCG), the Strong State Class Graph (SSCG) and the Atomic State Class Graph (ASCG) [7]. Both the SCG and the SSCG preserve linear properties, but the SCG is a better alternative for linear properties as it is smaller and faster to compute. On the other hand, the SSCG is used as a starting point in a refinement process to generate the ASCG which preserves branching properties (CTL*). During the refinement, state classes are split by linear constraints so that each state captured in a sub class has a successor in each one of the following classes. Such sub classes are said to be *atomic*, and atomicity of all classes ensures preservation of CTL* properties [8]. For the clock characterization of states, computed abstractions are: the Concrete State Zone Graph (CSZG) for linear properties [11], and its atomic version, the ACSZG [11] for CTL* properties. Compared with the ASCG, the ACSZG is in general smaller and faster to compute. For reachability properties, RT-Studio implements three contraction techniques (by inclusion, by convex combination and by convex hull) to rapidly generate contracted versions of the SCG, SSCG and CSZG which are suitable to verify reachability properties [11, 14]. It also implements several post contraction operations to further contract abstractions after they are constructed, and a minimizer under bisimulation. After computing any abstraction, RT-Studio allow the user to explore and edit it graphically. It also generates some statistics about the abstraction like its size (the number of nodes and edges) and the generation time. To verify properties, RT-Studio implements two model-checkers: a classical CTL model-checker and an innovative TCTL model-checker based on the forward on the fly verification technique described in [12, 13]. Finally, the simulator implemented in RT-studio allows for an assisted and interactive generation of any abstraction. Using the simulator, the user can intervene during the generation process, guide the generator to explore any branch of the state space graph being constructed, and even alter the model

[3] A self contained ITPN has no parameters and does not need to be synchronized with other ITPNs.

during the simulation if needed. RT-Studio stores a project in a single XML file. It also allows to import or export self contained ITPNs to a text file in a simple format accepted by the TINA toolbox [6]. Generated state spaces can also be exported in a text format supported by many model checkers.

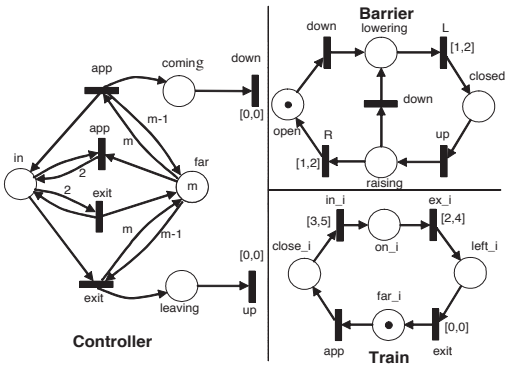

Fig. 1. The level crossing TPN model

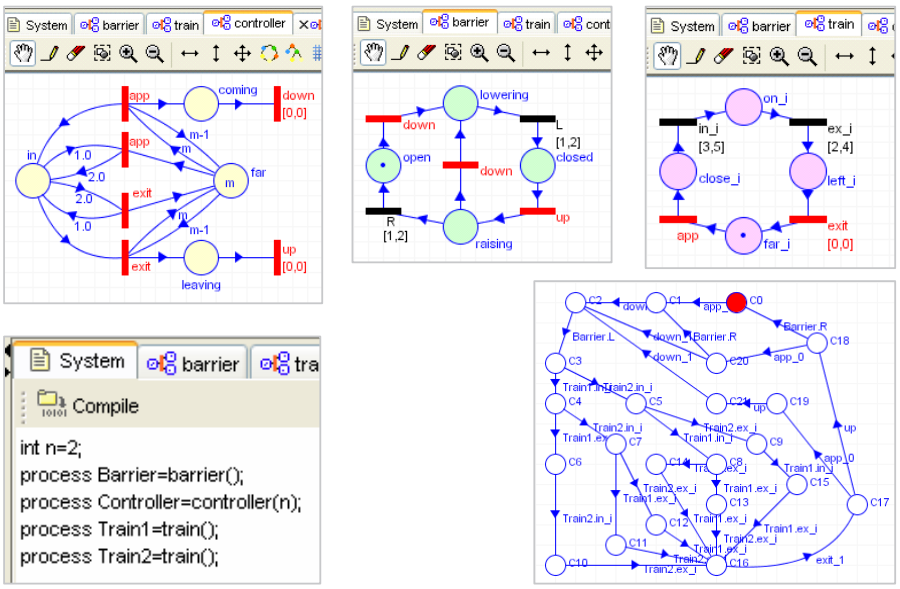

Fig. 2. Screen shots from the project "level crossing model": the controller, the barrier, the train. The picture at the right bottom is the SCG of the model

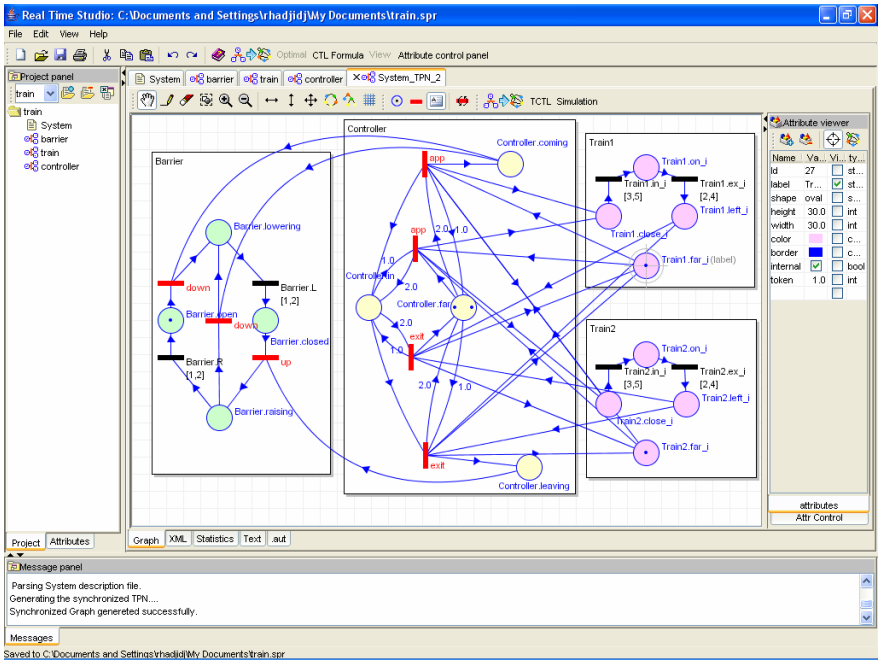

Fig. 3. RT-Studio interface

5 An illustrative example

As an illustrative example, we consider the classical *Railroad Crossing* model. Components of this system are shown in Figure 1 and also in Figure 2 as screen shots from RT-Studio. The ITPN model of n trains crossing concurrently the road is obtained by synchronously composing the controller model with its parameter m[4] set to n, the barrier model, and n instances of the train model. In Figure 2, we can see the system description file for the railroad crossing project where two instances of the train model (Train1 and Train2) are synchronized with one instance of the barrier model (Barrier) and one instance of controller model (Controller). After compiling the project, we obtain the synchronized ITPN of the whole system shown in Figure 3. Note that for clarity, in Figure 3 each transition *app* is actually the superposition of two *app* transitions; one synchronized with first train instance, the other one with the second train instance. At the right bottom of Figure 2, we can see the SCG of the synchronized ITPN computed by RT-studio.

[4] m is a number of tokens.

6 Installation

RT-Studio can be downloaded from *http://faculty.qu.edu.qa/rhadjidj/rt-studio.aspx*. The tool comes in a Zip file that includes three files : *RT-studio.jar*, *realtime.exe*, *readme.txt*, and a directory for examples. *RT-studio.jar* is a Java executable representing the graphical interface of the tool. *realtime.exe* is the engine written in C++ for performance. Installing the tool consists in simply unziping the zip file in chosen directory. Double clicking on the file *RT-studio.jar* launches the tool.

On the download web page there are video tutorials that explain how to install and use the tool to model, simulate and verify properties.

References

1. R. Alur, C. Courcoubetis, and D. Dill. Model checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.
2. R. Alur and D. Dill. Automata for modelling real-time systems. In *In Proc. Of ICALP'90*, volume 443 of LNCS, pages 322–335. Springer-Verlag, 1990.
3. G. Behrmann, J. Bengtsson, A. David, K. G. Larsen, P. Pettersson, and W. Yi. Uppaal implementation secrets. In *In Proc. of FTRTFT-02*, pages 3–22, 2002.
4. B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Trans. on Software Eng*, 17(3):259–273, 1991.
5. B. Berthomieu and M. Menasche. An enumerative approach for analyzing time petri nets. In *In Proc. of IFIP83*, volume 9, pages 41–46, September 1983.
6. B. Berthomieu, P.O. Ribet, and F. Vernadat. The tool tina – construction of abstract state spaces for petri nets and time petri nets. In *International Journal of Production Research*, 2004.
7. B. Berthomieu and F. Vernadat. State class constructions for branching analysis of time petri nets. In *In Proc. of TACAS'03*, volume 2619 of LNCS, pages 442–457. Springer-Verlag, 2003.
8. H. Boucheneb and R. Hadjidj. Ctl* model checking for time petri nets. *Theoretical Computer Science*, TCS 353(1-3):208–227, 2006.
9. Alexandre David, Lasse Jacobsen, Morten Jacobsen, Kenneth Yrke Jørgensen, Mikael H. Møller, and Jirí Srba. Tapaal 2.0: Integrated development environment for timed-arc petri nets. In *TACAS*, pages 492–497, 2012.
10. D. Lime G. Gardey, M. Magnin, and O.H. Roux. Roméo: A tool for analyzing time petri nets. In *In 17th International Conference on Computer Aided Verification (CAV'05)*, 2005.
11. R. Hadjidj. Analyse et validation formelle des systèmes temps réel. *Ph.D. Theses,University of Montreal (Ecole plytechnique)*, 2006.
12. R. Hadjidj and H. Boucheneb. On the fly TCTL model checking for time petri nets using the state class method. In *In Proc of ACSD'06*, pages 111–120. IEEE Computer Society Press, 2006.
13. R. Hadjidj and H. Boucheneb. On the fly TCTL model checking for time petri nets. *Theoretical Computer Science*, TCS 410(42):4241–4261, 2009.
14. R. Hadjidj and H. Boucheneb. Efficient reachability analysis for time petri nets. *IEEE Transaction on Computers*, 60(8):1085–1099, 2011.
15. P. Merlin and D. J. Farber. Recoverability of communication protocols - implication of a theoretical study. *IEEE Trans. on Communications*, 24(9):1036–1043, 1976.