Testing Real-time Systems Using TINA

Noureddine Adjir [1], Pierre De Saqui-Sannes[2], Kamel Mustapha Rahmouni[3]
[1] University of Saida - BP 138 – 20001 Ennasr Saida, Algeria
[2]University of Toulouse ISAE, 31 Avenue Edouard Belin -BP 44032- 31055, cedex 4 France
[3] University of Oran –BP 1524 - El mnaouar 31000 Oran, Algeria
Adjir_nourd@yahoo.fr, pdss@isae.fr, kamel_rahmouni@yahoo.fr

Abstract. The paper presents a technique for model-based black-box conformance testing of real-time systems using the Time Petri Net Analyzer TINA. Such test suites are derived from a prioritized time Petri net composed of two concurrent sub-nets specifying respectively the expected behaviour of the system under test and its environment.We describe how the toolbox TINA has been extended to support automatic generation of time-optimal test suites. The result is optimal in the sense that the set of test cases in the test suite have the shortest possible accumulated time to be executed. Input/output conformance serves as the notion of implementation correctness, essentially timed trace inclusion taking environment assumptions into account. Test cases selection is based either on using manually formulated test purposes or automatically from various coverage criteria specifying structural criteria of the model to be fulfilled by the test suite. We discuss how test purposes and coverage criterion are specified in the linear temporal logic SE-LTL, derive test sequences, and assign verdicts.

Keywords: real-time system; Prioritized Time Petri Nets; conformance testing; time optimal test cases.

1 Introduction

Real-Time systems are characterized by their capacity to interact with their surrounding environment and to provide the latter the expected output at the right date i.e. the timely reaction is just as important as the kind of reaction. Testing real-time systems is even more challenging than testing untimed reactive systems, because the tester must consider when to stimulate system, when to expect responses, and how to assign verdicts to the observed timed event sequence. Further, the test cases must be executed in real-time, i.e., the test execution system itself becomes a real-time system.

Model-based testing has been proposed as a technique to automatically verify that a system conforms to its specification. In this technique, test cases are derived from a formal model that specifies the expected behaviour of a system. In this paper, we propose a technique for automatically generating test cases and test suites for embedded real time systems based on Prioritized Time Petri Nets.

We focus on conformance testing i.e. checking by means of execution whether the behaviour of some black-box system, or a system part, called SUT (system under test), conforms to its specification. This is typically done in a controlled environment

where the SUT is executed and stimulated with input according to a test specification, and the responses of the SUT are checked to conform to its specification.

An important problem is how to select a very limited set of test cases from the extreme large number (usually infinitely many) of potential ones. So, a very large number of test cases (generally infinitely many) can be generated from even the simplest models. The addition of real-time adds another source of explosion, i.e. when to stimulate the system and expect response. Thus, an automatically generated test suite easily becomes costly to execute. To guide the selection of test cases, a test purpose or coverage criterions are often used. The paper demonstrates how it is possible to generate time-optimal test cases and test suites, i.e. test cases and suites that are guaranteed to take the least possible time to execute. The test cases can either be generated using manually formulated test purposes or automatically from several kinds of coverage criterion—such as transition or place or marking coverage– of the PrTPN model. The coverage approach guarantees that the test suite is derived systematically and that it guarantees a certain level of thoroughness. We describe how the real-time model checker *selt* and the path analysis tool *plan* of the toolbox TINA have been used to support automatic generation of time-optimal test suites for conformance testing i.e. test suites with optimal execution time. Such test suites are derived from a PrTPN composed of two subnets specifying respectively the expected behavior of the SUT and its environment. Especially, the required behaviour of the SUT is specified using a Deterministic Input enabled and Output Urgent PrTPN (DIOU-PrTPN). Time optimal test suites are interesting for several reasons. First, reducing the total execution time of a test suite allows more behaviour to be tested in the (limited) time allocated to testing; this means a more thorough test. Secondly, it is generally desirable that regression testing can be executed as quickly as possible to improve the turn around time between changes. Thirdly, it is essential for product instance testing that a thorough test can be performed without testing becoming the bottleneck, i.e., the test suite must be applied to all products coming of an assembly line. Finally, in the context of testing of real-time systems, we hypothesize that the fastest test case that drives the SUT to some state, also has a high likelihood of detecting errors, because this is a stressful situation for the SUT to handle. To know other advantages on Time optimal test suites, the reader can see [31].

The main contributions of the paper are: Re-implement the toolbox Tina and add functionality to support the composition of PrTPN's, definition of a subclass of PrTPN from which the diagnostic traces of *selt* can be used as test cases; application of time optimal paths analysis algorithms to the context of test case generation; a technique to generate time optimal covering test suites.

The rest of the paper is organized as follows: Section 2 surveys related work. Section 3 presents the LPrTPN model (syntax and semantics). In section4, we present test case generation based on the DIOU-PrTPN model and we describe how to encode test purposes and test criteria. Section 4 concludes the paper.

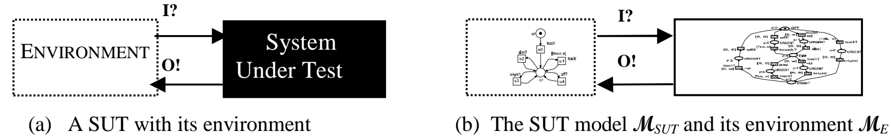
2 Motivation and Related Works

Among the models proposed for the specification and verification of real-time systems, two are prominent and widely used: Time Petri Nets (TPN) [39] and Timed Automata (TA) [2]. The TA formalism has become a popular and widespread formalism for specifying real-time systems. It has a rich theory and is cited in important research works e.g. fundamentals aspects, model checking, testing…etc. TPN are characterized by their condensed expression power of parallelism and concurrency, and the conciseness of the models. In addition, the efficient analysis methods proposed by [5] have contributed to their wide use. Many other extensions of Petri Nets exist e.g. p-time Petri Nets [29] and timed Petri Nets [43] but none of them has the success of TPN. Much research works compare TPN and TA in terms of expressivity w.r.t. language acceptance and temporal bisimilarity or propose translation from TA to TPN or vice versa e. g. [3], [4], [7], [14], [18] and [37]. It was shown in [14] that bounded TPN are equivalent to TA in terms of language acceptance, but that TA are strictly more expressive in terms of weak timed bisimilarity. Adding priorities to TPN (PrTPN) [7] preserves their expressiveness in terms of language acceptance, but strictly increases their expressiveness in terms of weak timed bisimilarity: it is proven in [7] that priorities strictly extend the expressiveness of TPN, and in particular that Bounded PrTPN can be considered equivalent to TA, in terms of weak timed bisimilarity i.e. that any TA with invariants is weak time bisimilar to some bounded PrTPN, and conversely. The TPN state space abstractions were prior to those of TA and TPN are exponentially more concise than classical TA [14]. In addition, interestingly, and conversely to the constructions proposed for model checking Prioritized TA the constructions required for PrTPNs preserve convexity of state classes; they do not require to compute expensive polyhedra differences [8]. Although, not many papers propose TPN for testing real-time systems e.g. [1] and [38]. So, until this paper, no test tool based on TPN, in particular conformance testing, is available. On the other hand, a lot of works on model based testing is based on TA or their extensions e.g. [15], [16], [17], [21], [24], [26], [28], [31], [32], [33], [34], [35], [36], [40], [41], [42], [45]; and there exist many tools for testing real-time systems based on TA more then ten years e.g. [24], [31], [36], and [40].

Many algorithms for generating test suites following test purposes or a given coverage criterion have also been proposed [29,22,18,13], including algorithms producing test suites optimal in the number of test cases, in the total length of the test suite, or in the total time required to execute the test suite. In this paper, we study test suite generation inspired by the analysis technique used in the State-Event LTL model-checker *selt* [8]. The schedules computed by the path analysis tool *plan*, in particular the fastest schedules and the shortest paths, associated to the diagnostic sequences (counterexamples), exhibited by *selt*, will be exploited to compute the optimal-time test suites.

3 Modeling the System and its Environment

A major development task is to ensure that an embedded system works correctly in its real operating environment and it's only necessary to establish its correctness under the modelled (environment) assumptions (Figure 1(a)); otherwise the environment model can be replaced with a completely unconstrained one that allows all possible interaction sequences. But, due to lack of resources it is not feasible to validate the system for all possible (imaginary) environments. However, the requirements and the assumptions of the environment should be clear and explicit. We assume that the test specification is given as an LPrTPN composed of two subnets: the first models the expected behaviour of the SUT, noted $\mathcal{M}_{SUT}$, while the second models the behaviour of its environment, and noted $\mathcal{M}_E$ (Figure 1(b)).



(a) A SUT with its environment (b) The SUT model $\mathcal{M}_{SUT}$ and its environment $\mathcal{M}_E$

Figure 1. A SUT with its environment. The SUT model $\mathcal{M}_{SUT}$ and its environment $\mathcal{M}_E$

3.1 Labeled Prioritized Time Petri Nets

Time Petri Nets (TPN*)*, introduced in [39], are obtained from PN by associating a temporal interval [tmin, tmax] with each transition, specifying firing delays ranges for the transitions. tmin and tmax respectively indicate the earliest and latest firing times of the transition (after the latter was enabled). Suppose that a transition *t* become enabled for the last one at the time θ, then *t* can't be fired before θ+tmin and it must be done at the latest at θ+tmax, unless disabled by firing some other transition. Prioritized TPN (PrTPN) extend TPN with a priority relation on the transitions; so a transition is not allowed to fire if some transition with higher priority is firable at the same instant. Such priorities increase the expressive power of TPN. Since we address the testing of reactive systems, we add an alphabet of actions *A* and a labelling function for transitions. *A* is partitioned in two separate subsets: input actions A_{in} and output actions A_{out}. Inputs are the stimuli received by the system from the environment. Outputs are the actions sent by this system to its environment. An input (output) is post fixed by ? (!). In addition, we assume the existence of internal actions denoted τ $(\tau \notin A)$. An internal action models the internal events of a system that are not observed by the tester.

Let I^+ be the set of nonempty real intervals with nonnegative rational endpoints. For $i \in \mathrm{I}^+$, $\downarrow i$ represent its lower endpoint, and $\uparrow i$ its superior endpoint (if it exists) or ∞. For any $\theta \in R^+, i \dot{-} \theta$ denotes the interval $\{ x-\theta | \; x \in i \wedge x \geq \theta \}$.

Syntax. Formally, a Labelled Prioritized Time Petri Net (LPrTPN in short) is a 9-uplet $(P,T,\text{Pre},\text{Post},m_0,I,\prec,A_\tau,L)$ where:

- $(P,T,\text{Pre},\text{Post},m_0)$ is a Petri Net where P is the set of places, T is the set of transitions, $m_0 : P \to \mathbb{N}^+$ is the initial marking and $\text{Pre}, \text{Post} : T \to P \to \mathbf{N}^+$ are the precondition and post-condition functions.
- $I_s : T \to \mathrm{I}^+$ is the static interval function which associates a temporal interval $I_s(t) \in \mathrm{I}^+$ with every transition in the net. The rational $\downarrow I_s(t)$ and $\uparrow I_s(t)$ are the static earliest firing time and the static latest firing time of *t*, respectively. In this paper, intervals $[0,\infty[$ are omitted and ***w*** in the right end point of an interval denotes ∞.
- $\prec \subseteq T \times T$ is the priority relation, assumed irreflexive, asymmetric and transitive, between transitions. $t_1 \succ t_2$ or $t_2 \prec t_1$ means t_1 has priority over t_2.
- $A_\tau = A_{in} \cup A_{out} \cup \{\tau\}$ is a finite set of actions
- $L : T \to A_\tau$ is the labelling function that associates to each transition an operation.

A marking is a function $m : P \to \mathbb{N}^+$. A transition t is enabled at marking m iff $m \geq \text{Pre}(t)$. The set of transitions enabled at m are denoted by $En(m) = \{\, t \mid \text{Pre}(t) \leq m \}$.

The predicate specifying when k is newly enabled by the firing of an internal transition t from marking m is defined by:

$$NS_{(t,m)}(k) = k \in En(m - \text{Pre}(t) + \text{Post}(t)) \wedge (k \notin En(m - \text{Pre}(t)) \vee k = t).$$

The predicate specifying when k is newly enabled by the firing of a couple of complementary transitions (t,t') from marking m is defined by:

$$NS_{(t,t',m)}(k) = k \in En\big(m - (\mathbf{Pre}(t) + \mathbf{Pre}(t')) + \mathbf{Post}(t) + \mathbf{Post}(t')\big) \wedge \big(k \notin En\big(m - (\mathbf{Pre}(t') + \mathbf{Pre}(t'))\big) \vee k = t \vee k = t'\big)$$

The sets of internal, input and output transitions of the net are defined respectively by: $T_\tau = \{\, t \in T \,/\, L(t) = \tau \}$, $T_{in} = \{\, t \in \overline{T_\tau} \,/\, L(t) \in A_{in} \}$ and $T_{out} = \{ t \in \overline{T_\tau} \,/\, L(t) \in A_{out} \}$ (with $\overline{T_\tau} = T - T_\tau = T_{in} \cup T_{out}$).

The set of environment model transitions which complement a transition t of the SUT model is noted $CT(t) = \{ t' \in \mathcal{M}_E \mid \text{ if } t = a! \text{ (resp. } a?) \text{ then } t' = a? \text{ (resp. } a!) \}$.

A state of an LPrTPN is a pair $e = (m, I)$ in which m is a marking of the net and $I : T \to \mathrm{I}^+$, a partial function called the interval function, associates exactly a temporal interval in I^+ with every enabled transition $t \in En(m)$. The initial state is $e_0 = (m_0, I_0)$, where I_0 is I_S restricted to the transitions enabled at m_0. The temporal information in states will be seen as firing domains, instead of intervals functions. The initial state $e_0 = (m_0, D_0)$ of the LPrTPN of Figures 2 and 3.a is defined by:

$$m_0 : p_0,\ q_0 \quad \text{and } D_0: \quad \begin{aligned} &0 \leq t_0 \\ &\text{Tidle} \leq t_8 \\ &0 \leq s_0 \end{aligned}$$

Semantics. The semantic of an LPrTPN $N=(P,T,\mathrm{Pre},\mathrm{Post},m_0,Is,\prec,A_\tau,L)$ is the Timed Transition System $E_N=(E,e_0,A_{in},A_{out},\rightarrow)$ where E is the set of states (m,I) of the LPrTPN and $e_0=(m_0,I_0)$ its initial state. $A_{in}=L(T_{in})$ and $A_{out}=L(T_{out})$.

$\longrightarrow\subseteq E\times T\cup\mathbb{R}_{\geq 0}\times E$ is the transition relation between states. It corresponds to two kinds of transitions witch includes discrete transitions (labelled with synchronized or internal actions) and temporal or continuous transitions (labelled by real values).

The continuous (or delay) transitions are the result of time elapsing. We have $(m,I)\xrightarrow{d}(m,I')$ iff $d\in\mathbb{R}_{\geq 0}$ and:

1. $(\forall t\in T)\ \ (t\in En(m)\Rightarrow d\leq\uparrow I(t)\ )$
2. $(\forall t\in T)\ \ (t\in En(m)\Rightarrow\ I'(t)=I(t)\ \dot{-}\ d\ )$

A continuous transition of size *d* is possible iff *d* is not greater then the latest firing time of all enabled transitions. All firing intervals of enabled transitions are shifted synchronously towards the origin as time elapses, and truncated to non negative times.

Discrete transitions are the result of the transitions firing of the net. They may be further partitioned into purely SUT or ENV transitions (hence invisible for the other part) or synchronizing transitions between the SUT and the ENV (hence observable for both parties). Internal transitions are fired individually while synchronizing transitions are fired by complementary actions couples (e.g. a? and a! are complementary synchronization actions). The first component of the couple is a transition of the SUT model, labelled by an input (resp. output) action, and the second component is an environment transition and labelled by an output (resp. input) action.

The discrete internal transitions: we have $(m,I)\xrightarrow{t}(m',I')$ iff $L(t)=\tau$ and :

1. $t\in En(m)$
2. $0\in I(t)$
3. $(\forall k\in T_\tau)\ \ (k\in En(m)\wedge(k\succ t)\Rightarrow 0\notin I(k))$
4. $(\forall k\in\overline{T_\tau},\forall k'\in TC(k))(k,k'\in En(m)\wedge(k\succ t)\Rightarrow 0\notin I(k)\wedge 0\notin I(k'))$
5. $m'=m\ -\ \mathrm{Pre}\ (t)+\mathrm{Post}(t)$
6. $(\forall k\in T)\big(m'\geq \mathrm{Pr}e(k)\Rightarrow I'(k)=\text{if}\ \ NS_{(t,m)}(k)\ \text{then}\ I_s(k)\ \text{else}\ \ I(k)$

An internal transition *t* may fire from the state (m,I) if it is enabled at *m* (1), immediately firable (2) and no transition with higher priority satisfies these conditions (3 & 4). In the target state, the transitions that remained enabled while *t* fired (*t* excluded) retain their intervals, the others are associated with their static intervals (6).

The discrete synchronizing transition: we have:
$(m,I)\xrightarrow{L(t),L(t')}(m',I')$ iff $t,t'\in\overline{T_\tau},\ t'\in TC(t)$ and :

1. $t,t'\in En(m)$
2. $0\in I(t)\wedge 0\in I(t')$

3. $(\forall k \in T_\tau)\ (k \in En(m) \wedge (k \succ t) \Rightarrow 0 \notin I(k))$

4. $(\forall k \in T, \forall k' \in TC(k))(k, k' \in En(m) \wedge (k \succ t \vee k' \succ t') \Rightarrow 0 \notin I(k) \wedge 0 \notin I(k'))$

5. $m' = m - (\mathrm{Pre}(t) + \mathrm{Pre}(t')) + \mathrm{Post}(t) + \mathrm{Post}(t')$

6. $(\forall k \in \overline{T_\tau})(m' \geq \mathrm{Pr}e(k) \Rightarrow I'(k) = \text{ if } NS_{(t,t',m)}(k) \text{ then } I_s(k) \text{ else } I(k)$

The complementary transitions t and t' may fire from the state (m, I) if they are enabled (1), immediately firable (2) and neither internal transition (3) nor couple of complementary transitions with higher priority satisfies these conditions (4). In the target state, the transitions that remained enabled while t and t' fired (t and t' excluded) retain their intervals, the others are associated with their static intervals (6).

If the light controller and its environment (Figure 2 and 3) are in their initial state and make a delay of 0.6 time unites ($e_0 \xrightarrow{0.6} e_1$). The new state $e_1 = (m_0, D_1)$ will be:

$m_0 : p_0,\ q_0$ and D_1: $0 \leq t_0$

$\mathrm{Tidle} - 0.6 \leq t_8$

$0 \leq s_0$

The firing of the synchronizing transition (t_0, s_0) from the state e_1 leads to the state e_2 ($e_1 \xrightarrow{touch?, touch!} e_2$). The new state $e_2 = (m_1, D_2)$ will be:

$m_1 : p_1,\ q_1$ and D_2: $0 \leq t_1 \leq 0$

$\mathrm{Treact} \leq s_1 \leq \infty$

$0 \leq s_2 \leq \infty$

$0 \leq s_3 \leq \infty$

$0 \leq s_4 \leq \infty$

A firing schedule, or a time transitions sequence, is a sequence alternating delay and discrete transitions $d_1\alpha_1 d_2\alpha_2 \cdots d_n\alpha_n$. α_i is a pure transition $(\alpha_i = k \in T_\tau)$ or a synchronizing transition $(\alpha_i = (t, t') |\ t \in \overline{T_\tau} \wedge t' \in TC(t))$ and d_i are the relative firing times. A schedule is realisable from the state e if the discrete transitions of the sequence $\sigma = \alpha_1\alpha_2 \cdots \alpha_n$ are successively firable from e at the associated relative firing times $d_1, d_2, \cdots, d_n$. The sequence σ is called its support.

If the pausing time Tidle and the switching time Tsw are respectively equal to 20 and 4 time units then the following time sequence is a realisable schedule $20.(touch?, touch!).0.(bright!, bright?).5.(touch?, touch!).0.(\mathrm{dim}!, \mathrm{dim}?).4.(touch?, touch!).0.(off!, off?)$

3.2 Tina (TIme Petri Net Analyzer)

Tina is a software environment for editing and analyzing TPN [6]. It includes the tools:

– *nd* (NetDraw) : an editor for graphical or textual description of TPN.

– *tina* : For analysing LPrTPN models, it's necessary to finitely represent the state spaces by grouping some sets of states. *tina* builds the strong state classes graph

(SSCG in short), proposed in [8], which preserves states and maximal traces of the state graph, and thus the truth value of all the formulae of the SE-LTL logic.
– *plan* is a path analysis tool. It computes all, or a single, timed firing sequence (schedule) over some given firing transitions sequence. In particular, it computes the fastest schedules and shortest paths. Accordingly, the latter schedules are used for test case generation.
– *selt*: is a model checker for an enriched version of state-event LTL [19], a linear temporal logic supporting both state and transition properties. For the properties found false, *selt* produces a timed counter example. It's called a diagnostic schedule of the property. The realization of this schedule from the initial state satisfies the property.

A diagnostic sequence of a property ϕ is a sequence of discrete transitions (internal and/or complementary transitions). The successive firing of these transitions, from m_0, at the corresponding dates, allows satisfying the property ϕ. A diagnostic trace is a schedule whose support is a diagnostic sequence.

3.3 Deterministic, Input Enabled and Output Urgent LPrTPN

To ensure time optimal testability, the following semantic restrictions turn out to be sufficient. Following similar restrictions as in [31] and [45], we define the notion of deterministic, input enabled and output urgent LPrTPN, DIEOU-LPrTPN, by restricting the underlying timed transition system defined by the LPrTPN as follows: (1) Deterministic: For every semantic state $e=(m,D)$ and an action $\gamma \in A \cup \{\mathbb{R}_{\geq 0}\}$, whenever $e \overset{\gamma}{\rightarrow} e'$ and $e \overset{\gamma}{\rightarrow} e''$ then $e' = e''$. (2) (Weak) input enabled: whenever $e \overset{d}{\rightarrow}$ for some delay $d \in \mathbb{R}_{\geq 0}$ then $\forall a \in A_{in}$, $e \overset{a}{\rightarrow}$. (3) Isolated outputs: $\forall \alpha \in A_{out} \cup \{\tau\}$, $\forall \beta \in A_{out} \cup A_{in} \cup \{\tau\}$ whenever $e \overset{\alpha}{\rightarrow}$ and $e \overset{\beta}{\rightarrow}$ then $\alpha = \beta$. (4) Output urgency: whenever $e \overset{\alpha}{\rightarrow}$, $\forall \alpha \in O \cup \{\tau\}$ then $e \overset{d}{\nrightarrow}, d \in \mathbb{R}_{\geq 0}$.

We assume that the tester can take the place of the environment and control the SUT via a distinguished set of observable input and output actions. For the SUT to be testable the LPrTPN modelling it should be controllable in the sense that it should be possible for an environment to drive the model through all of its syntactical parts (transitions and places). We therefore assume that the SUT specification is a DIEOU-LPrTPN, and that the SUT can be modeled by some unknown DIEOU-LPrTPN. The environment model need not be a DIEOU-LPrTPN. These assumptions are commonly referred to as the testing hypothesis.

Figure 2 shows an LPrTPN modelling the behaviour of a simple light-controller (this example is taken from [31]). The user interacts with the controller by touching a touch sensitive pad. The light has three intensity levels: `OFF`, `DIMMED`, and `BRIGHT`. Depending on the timing between successive touches, the controller toggles the light levels. For example, in dimmed state, if a second touch is made quickly (before the switching time $T_{sw} = 4$ time units) after the touch that caused the controller to enter dimmed state (from either off or bright state), the controller increases the level to

bright. Conversely, if the second touch happens after the switching time, the controller switches the light off. If the light controller has been in off state for a long time (longer than or equal to $T_{idle} = 20$), it should reactivate upon a touch by going directly to bright level. We leave to the reader to verify for herself that the conditions of DIEOU-LPrTPN are met by the given model.

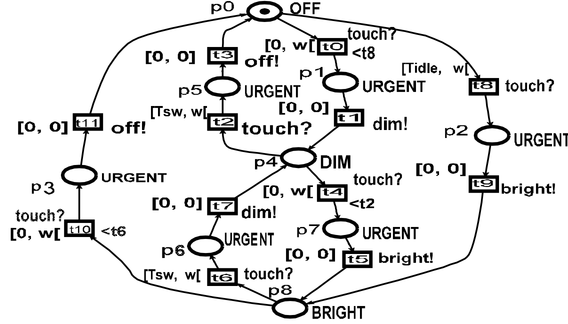
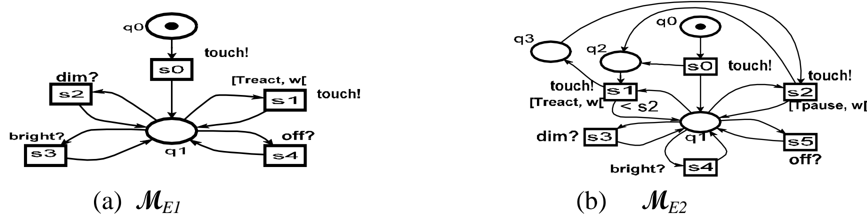


Figure 2. $\mathcal{M}_{SUT}$: the light controller model

Figure 3 shows two possible environment models for the simple light controller. Figure 3(a) models a user capable of performing any sequence of touch actions. When the constant *Treact* is set to zero he is arbitrarily fast. A more realistic user is only capable of producing touches with a limited rate; this can be modeled setting *Treact* to a non-zero value. Figure 3(b) models a different user able to make two quick successive touches, but which then is required to pause for some time (to avoid cramp), e.g., *Tpause* = 5. The LPrTPN shown in Figure 2 and Figure 3 respectively can be composed in parallel on actions A_{in} = {touch} and A_{out} = {off, dim, bright}.



(a) $\mathcal{M}_{E1}$ (b) $\mathcal{M}_{E2}$

Figure 3. Two light switch controller environment models

The firing of (t_1, s_2) from the state $e_2 = (m_1, D_2)$ leads to the state $e_3 = (m_2, D_3)$ ($e_2 \xrightarrow{\text{dim?,dim!}} e_3$):

$m_2 : p_4, q_1$ and D_2: $\text{Tsw} \le t_2 \le \infty$

$0 \le t_4 \le \infty$

$\text{Treact} \le s_1 \le \infty$

$0 \le s_2 \le \infty$

$0 \le s_3 \le \infty$

$0 \le s_4 \le \infty$

4 Test Generation

4.1 From Diagnostic Traces to Test Cases

Let $\mathcal{M}$ be the LPrTPN model of the SUT together with its intended environment ENV; and ϕ the property, formulated in SE–LTL, to be verified over $\mathcal{M}$. As SE–LTL evaluate the properties on all possible executions, we consider the formula $\neg\phi$ then we submit it to *selt*. If the response is negative, i.e. all the executions don't satisfy $\neg\phi$, so at least one satisfy its negation ϕ. *selt* provide simultaneously a counter example for $\neg\phi$, i.e. a diagnostic sequence that demonstrates that property ϕ is satisfied. This sequence is submitted to the tool *plan* for computing a schedule, or all the schedules having this sequence as support. This schedule is an alternating sequence of discrete transitions, synchronization (or internal) actions, performed by the system and its environment, and temporal constraints (or transitions firings time-delays) needed to reach the goal (the desirable marking or event).

Once the diagnostic trace is obtained, it's convenient to construct the associated test sequences. For DIEOU-LPrTPN, a test sequence is an alternating of sequence of concrete delay actions and observable actions (without internal actions). From the diagnostic trace above a test sequence, λ, may be obtained simply by projecting the trace to the environment component, $\mathcal{M}_E$, while removing invisible transitions, and summing adjacent delay actions. Finally, a test case to be executed on the real SUT implementation may be obtained from λ by the addition of verdicts. Adding the verdicts depends on the chosen conformity relation between the specification and SUT. In this paper, we require timed trace inclusion, i.e. that the timed traces of the SUT are included in the specification. Thus after any input sequence, the SUT is allowed to produce an output only if the specification also able to produce that output. Similarly, the SUT may delay (staying silent) only if the specification also may delay. The test sequences produced by the technique proposed in this paper are derived from the diagnostic traces, and are thus guaranteed to be included in the specification.

To clarify the construction we may model the test case itself as an LPrTPN $\mathcal{M}_\lambda$ for the test sequence λ. Places in M_λ are labelled using two distinguished labels, **Pass** and **Fail**. The execution of a test case is formalized as a parallel composition of the test case Petri net $\mathcal{M}_\lambda$ and SUT $\mathcal{M}_{SUT}$.

$$\text{SUT } \textbf{\textit{passes}}\ \mathcal{M}_\lambda \ \text{ iff } \mathcal{M}_\lambda \| \ \mathcal{M}_{SUT} \not\rightarrow \textbf{\textit{fail}}$$

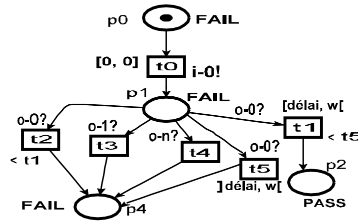


Figure 4. Test case LPrTPN $\mathcal{M}_\lambda$ for the sequence $\lambda = i_0! \,.\, delai \,.\, o_0\,?$

$\mathcal{M}_\lambda$ is constructed such that a complete execution terminates in a **Fail** state (the place FAIL will be marked) if the SUT cannot perform λ and such that it terminates in a **Pass** state (the place PASS will be marked) if the SUT can execute all actions of λ. The construction is illustrated in Figure 4.

4.2 Single Purpose Test Generation

A common approach to the generation of test cases is to first manually formulate a set of informal test purposes and then to formalize these such that the model can be used to generate one or more test cases for each test purpose. Because we use the diagnostic trace facility of the model-checker *selt*, the test purpose must be formulated as a SE-LTL property that can be checked by reachability analysis of the combined model $\mathcal{M}$. The test purpose can be directly transformed into a simple state or event reachability check. Also, the environment model can be replaced by a more restricted one that matches the behaviour of the test purpose only.

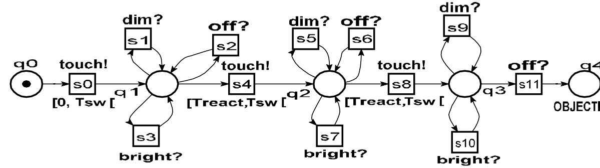


Figure 5. $\mathcal{M}_{E3}$, test environment for TP2

TP1: check that the light can become bright.
TP2: check that the light switches off after three successive touches.

TP1 can be formulated as a simple SE-LTL property $\phi_1 = \Diamond \text{BRIGHT}$ (state property) or $\phi_2 = \Diamond BRIGHT!$ (event property) i.e. eventually in some future the place bright of the light controller Petri net will be marked or the event bright! will be executed.

Among all diagnostic sequences exhibited by *selt* that satisfy the property ϕ_1 (or ϕ_2), two sequences are more interesting: the shortest and the fastest sequences. The second is selected as follows: first, we compute the fastest schedule associated for each obtained sequence, and then we keep only the schedule with the smallest accumulated time. Finally, the two schedules associated to the two selected sequences will be transformed to test cases as explained in 4.1. The execution time for each of these test cases is optimal.

For the light controller, the shortest diagnostic trace is $(touch?, touch!)\,(bright!, bright?)$. It results in the test sequence $20.touch!.0.bright?$. However, the fastest sequence satisfying ϕ_1 is $0.(touch?,touch!).0.(dim!,dim?).0.(touch?,touch!)$. It results in the test sequence $0.touch!.0.\text{dim}?.0.touch!.0.bright?$

TP2 can be formalized using the property $\mathcal{M}_{E3} \models \phi_3 = \Diamond \text{OBJECTIF}$ with $\mathcal{M}_{E3}$ is the restricted environment model in Figure 5. The fastest test sequence is:
$0.touch!.0.\text{dim}?.0.touch!.0.bright?.0.touch!.0.off?$

4.3 Coverage Based Test Generation

A large suite of coverage criteria may be proposed, such as statement, transition, states, and classes, each with its merits and application domain. We explain how to apply some of these to TPN models. In this paper, we use three coverage criteria of the LPrTPN model of the SUT:

Transition Coverage. A test sequence satisfies the transition-coverage criterion if, when executed on the model, it fires every transition of the net. Transition coverage can be formulated by the property $\phi_t = \bigwedge_{i=1}^{n} \Diamond t_i$, where n is the number of transitions of the net. The obtained counter example of the non satisfaction of the property $\neg\phi_t$ ensures transition coverage. Once the diagnostic sequences are obtained, we compute the two schedules: (1) the fastest schedule which has as support the shortest sequence (2) the fastest schedule among all schedules exhibited by *selt*. We transform these schedules in test cases as is indicated in 4.1.

When the environment can touch arbitrarily, the generated fastest transition covering test has the accumulated execution time 28. The solution (there might be more traces with the same fastest execution time) generated by *plan* is:

TC: *0.touch!.0.dim?.0.touch!.0.bright?.0.touch!.0.off?.20.touch!.0.bright?.4.touch!.0.dim?.4. touch!.0.off?*

Place Coverage. A test sequence satisfies the place-coverage criterion if, when executed on the model, it marks every place of the net. Place coverage can be formulated by the property $\phi_P = \bigwedge_{i=1}^{m} \Diamond p_i \geq 1$, where m is the number of places of the net.

Marking Coverage. A test sequence satisfies the marking-coverage criterion if, when executed on the model, it generates all the markings of the net. The test sequences which ensure the marking-coverage are generated by selecting the transition sequence(s), from the SSCG of the model, which generates all the markings of the SUT model. Test cases generation from the diagnostic traces that ensures place coverage and marking coverage are computed as in coverage transition.

4.4 Test Suite Generation

Frequently, for a given test purpose criterion, we cannot obtain a single covering test sequence. This is due to the dead-ends in the model. To solve this problem, we allow for the model (and SUT) to be reset to its initial state and to continue the test after the reset to cover the remaining parts. The generated test will then be interpreted as a test suite consisting of a set of test sequences separated by resets (assumed to be implemented correctly in the SUT).

To introduce resets in the model, we shall allow the user to designate some markings as being reset-able i.e. markings that allows to reach the initial marking m_0. Evidently, performing a reset may take some time T_r that must be taken into account when generating time optimal test sequences. Reset-able markings can be encoded into the model by adding reset transitions leading back to the initial marking. Let m_r the reset-able marking, two reset transitions and a new place *q* which must be added as:

The transition *reset*! must be added such as their input places are the encoded places (those of m_r) and its output place is the place *q*. The firing of *reset*! marks the place *q*. $(m_r, -) \xrightarrow{reset!} (q, [T_r, T_r]) \xrightarrow{\tau} (m_0, I_0)$

4.5 Environment Behaviour

Test sequences generated by the techniques presented above may be non-realizable; they may require the SUT environment to operate infinitely fast. Generally, it is only necessary to establish correctness of SUT under the environment assumptions. Therefore assumptions about the environment can be modelled explicitly and will then be taken into consideration during test sequence generation. We demonstrate how different environment assumptions influence the generated test sequences.

Consider an environment where the user takes at least 2 time units between each touch action, such an environment can be obtained by setting the constant T_{react} to 2 in Figure 3(a). The fastest test sequences become **TP1:** *0.touch!.0.dim?.2.touch!.0.bright?* and **TP2:** *0.touch!.0.dim?.2.touch!.0.bright?.2.touch!.0.off?*

Also re-examine the test suite **TC** generated by transition coverage, and compare with the one of execution time 32 generated when T_{react} equals 2.

TC': *0.touch!.0.dim?.4.touch!.0.off?.20.touch!.0.bright?.4.touch!.0.dim?.2.touch!.0.bright?.2. touch!.0.off?*

When the environment is changed to the pausing user (can perform 2 successive quick touches after which he is required to pause for some time: reaction time 2, pausing time 5), the fastest sequence has execution time 33, and follows a completely different strategy.

TC'': *0.touch!.0.dim?.2.touch!.0.bright?.5.touch!.0.dim?.4.touch!.0.off?.20.touch!.0.bright?.2. touch!.0.off?*

6 Conclusion

In this paper, we have demonstrated that the problem of timed test generation is transformed to a problem of model checking. We have shown that time-optimal test suites, computed from either a single test purpose or coverage criteria can be generated using the Tina toolbox. We have also introduced modifications in the transitions firings algorithms taking into account the reactive character of embedded real-time systems. Unlike the technique based on TA [31], the advantages of using TINA are the following: 1) when computing the SSCG for bounded PrTPN, contrary

to the zone graph of TA, no abstraction is required in order to ensure termination; this allows to avoid ad-hoc techniques for enforcing termination of forward analysis; 2) it may help tackling the state explosion problem due to parallel composition of TA.

The DIEOU-PrTPN is quite restrictive, and generalization will benefit many real-time systems.

References

1. Adjir, N., De Saqui-Sanne, P, Rahmouni, M., K., Génération des séquences de test temporisés à partir des réseaux de Petri temporels à chronomètres, NOTERE'07, Maroc.
2. Alur, R., Dill, D., A theory of timed automata, Theoretical Computer Science, 126 (2):183–235, 1994.
3. Berard, B., Cassez, F, Haddad, S., Roux, O. H., et Lime., D., Comparison of the Expressiveness of Timed Automata and Time Petri Nets, In FORMATS'05, Springer LNCS 3829, pages 211–225, 2005.
4. Bérard, B., Cassez F., Haddad S., Lime D. and Roux O. H., When are timed automata weakly timed bisimilar to time Petri nets?, *FSTTCS'5, LNCS 3821,* Hyderabad, India, 2005.
5. Berthomieu, B., M. Diaz, modelling and verification of time dependent systems using time Petri nets, IEEE transactions on software Engineering, 17(3), 1991.
6. Berthomieu B., Ribet P. O., Vernadat F., The tool TINA -- Construction of Abstract State Spaces for Petri Nets and Time Petri Nets, *Inter. JPR, Vol. 42, No 14,* July 2004.
7. Berthomieu B., Peres F., Vernadat F., Bridging the gap between Timed Automata and Bounded Time Petri Nets, *In Proc. of FORMATS 2006. Springer Verlag, LNCS 4202,* 2006.
8. Berthomieu, B., F. Peres, and Vernadat F., Model Checking Bounded Prioritized Time Petri Nets. In ATVA 2007, Springer LNCS 4762, pages 523–532, 2007
9. Berthomieu, B., P.-O. Ribet, and Vernadat F., The tool TINA –construction of abstract state spaces for Petri nets and time Petri nets. IJPR, 42(14) :2741– 2756, 15 July 2004.
10. Berthomieu, B., et Vernadat F., State Space Abstractions for Time Petri Nets, Handbook of Real-Time and Embedded Systems, CRC Press, Boca Raton, FL., U.S.A., 2007.
11. Bouyer P., Dufourd C., Fleury E., and Petit A., Updatable timed automata, *TCS, 321(2–3):291–345,* 2004.
12. Bouyer P., Forward Analysis of Updatable Timed Automata, *FMSD* 24(3), 281-320, 2004.
13. Bouyer P., Chevalier F., On conciseness of extensions of timed automata, *JALC,* 2005.
14. Bouyer P., Serge H., Reynie P. A., Extended Timed Automata and Time Petri Nets, *in ACSD'06, Turku, Finland, 91-100, IEEE Computer Society Press*, juin 2006.
15. Braberman V., Felder M., Marre M., Testing timing behaviour of real-time software, *In Intern.Software Quality Week,* 1997.
16. Brinksma E., Tretmans J., Testing transition systems: An annotated bibliography*, In MOVEP 2000, 2067 of LNCS*, Springer, 2001.
17. Cardell-Oliver R., Conformance test experiments for distributed real-time systems, *In ISSTA'02, ACM Press,* 2002.
18. Cassez F., Roux O. H., Structural translation from time Petri nets to timed automata, *JSS 2006.*
19. Chaki S., Clarke E., M., Ouaknine J., Sharygina N., Sinha N., State/Event-based Software Model Checking, 4th ICIFM, Springer LNCS 2999, 128-147, 2004.
20. Choffrut C. and Goldwurm M., Timed automata with periodic clock constraints, *JALC, 5(4):371–404*, 2000.

21. Cleaveland R., Hennessy M., Testing Equivalence as a Bisimulation Equivalence, Formal Aspects of Computing, 5:1-20, 1993.
22. David A., Hakansson J., Larsen K. G., et Pettersson P., Model checking timed automata with priorities using DBM subtraction, *in FORMATS'06,LNCS 4202, 128–142,* 2006.
23. Demichelis F. and Zielonka W., Controlled timed automata, *In Proc. CONCUR'98, vol. 1466 of LNCS, p. 455–469, Springer,* 1998.
24. de Vries R., Tretmans J., on-the-fly conformance testing using SPIN, *STTT, 2(4): 382-393,* March 2000.
25. Diekert V., Gastin P., Petit A. Removing epsilon-Transitions in Timed Automata, *In 14th an. stacs 1197, p;583-594, LNCS, Vol. 1200, Springer, Lubeck, Germany,* February 1997.
26. En-Nouaary A., Dssouli R., Khendek F., Elqortobi A., Timed test cases generation based on state characterization technique , *In RTSS'98, IEEE*, 1998.
27. Fersman E., Petterson P., and Yi W., Timed automata with asynchronous processes: Schedulability and decidability In *Proc. TACAS'02, vol.2280 of LNCS, P. 67–82,* 2002.
28. Fernandez J.C., Jard C., Jéron T., Viho G., Using on-the-fly verification techniques for the generation of test suites », *In CAV'96, LNCS 1102*, 1996.
29. Khansa W., réseaux de Petri P-temporels : contribution à l'étude des systèmes à événements discrets, thèse de doctorat, université de Savoie, Annecy, France 1997.
30. Henzinger T. A., The theory of hybrid automata, *Proc.LICS'96, 278–29,. IEEE CSP,* 1996.
31. Hessel A., Larsen K., Nielsen B., Pettersson P., Skou A., Time-optimal real-time test case generation using UPPAAL , *In FATES'03,* Montreal, October 2003.
32. Higashino T., Nakata A., Taniguchi K., Cavalli A., Generating test cases for a timed I/O automaton model, *In IFIP Int'l Work, Test Comm. System* Kluwer, 1999.
33. Jéron T., Morel P., Test generation derived from model-cheking, *In Halbwachs and D. peled Editors, CAV'99, Trento, Italy, 1633 of LNCS, 108-122. Springer-Verleg*, july 1999.
34. Jéron T., Rusu V., Zinovieva E., « STG: A symbolic test generation tool, *In TACAS'02, 2280 of LNCS, Springer,* 2002.
35. Khoumsi A., Jéron T., Marchand H., Test cases generation for nondeterministic real-time systems, *In FATES'03, Montreal*, October 2003.
36. Krichen M., Tripakis S., An Expressive and Implementable Formal Framework for Testing Real-Time Systems, *In 17th IFIP Intl. TestCom'05*, 2005
37. Lime D., Roux O., H., State class timed automaton of a time Petri net, in PNPM'03, 124-133, Urbana, USA, 2003, IEEE computer society.
38. Lin J. C., Ho I., « Generating Real-Time Software Test Cases by Time Petri Nets, *IJCA (EI journal), ACTA Press, U.S.A. Vol. 22, No.3,151-158,* Sept. 2000.
39. Merlin P. M., Farber J., Recoverability of communication protocols: Implications of a theoretical study, *IEEE Trans. Com., 24(9):1036-1043*, September 1976.
40. Mikucionis M., K. G. Larsen, Brian Nielsen, T-UPPAAL: Online Model-based Testing of Real-time Systems, *19th IEEE Internat. Conf. ASE, 396-397. Linz, Austria*, 2004.
41. Nielsen B., Skou. A., Automated test generation from timed automata, *In TACAS'01, LNCS 2031, Springer,* 2001.
42. Peleska J., Formal methods for test automation - hard real-time testing of controllers for the airbus aircraft family, *In IDPT'02*, 2002.
43. Ramchadani C., Analysis of asynchronous concurrent systems by timed Petri nets, Cambridge, Mass, MIT, dept Electrical Engineering, Phd thesis, 1992.
44. Shang-Wei L., Pao-Ann H., Chun-Hsian H., et Yean-Ru C., Model checking prioritized timed automata, In ATVA'2005, Springer LNCS 3707, 370–384, 2005.
45. Springintveld J., Vaandrager F., D'Argenio P., Testing timed automata, *TCS, 254*, 2001.
46. Tretmans J., Testing concurrent systems: A formal approach, *In J.C.M Beaten and S. Mauw editors, CONCUR'99 CCT, vol. 1664 of LNCS, 46–65. Springer-Verlag*, 1999.