Test of Preemptive Real-time Systems

Noureddine Adjir [1, 2, 3], Pierre de Saqui-Sannes [1, 2], Kamel Mustapha Rahmouni [4]

[1] ENSICA, Université de Toulouse, 1 place Emile Blouin- 31056 Toulouse Cedex 5, France
{nadjir, pdss}@ensica.fr
[2] LAAS-CNRS, Université de Toulouse, 7 avenue du Colonel Roche- 31077 Toulouse Cedex 4, France
[3] LMMC, Centre Universitaire Moulay Tahar, BP. 138, En-Nasr 20002 Saida, Algérie
[4] Département d'Informatique, Université d'Oran, 31000, Algérie
kamel_rahmouni@yahoo.fr

Abstract

Time Petri nets with stopwatches not only model system/environment interactions and time constraints. They further enable modeling of suspend/resume operations in real-time systems. Assuming the modelled systems are non deterministic and partially observable, the paper proposes a test generation approach which implements an online testing policy and outputs test results that are valid for the (part of the) selected environment. A relativized conformance relation named rswtioco is defined and a test generation algorithm is presented. The proposed approach is illustrated on an example.

1. Introduction

In black box testing, also called model-based testing, test cases are generated from the specification of the system and executed against the system under test (SUT). There are several works of test case generation from specifications of real-time systems. Real-time systems are not only characterized by their capacity to interact with their surrounding environment and to provide the latter the expected outputs at the right time. They may be interrupted at any time while keeping the capacity to restart later on without losing their state information. Therefore a real-time specification model should include a suspend/resume capability. A survey of the literature indicates that reactivity and timeliness have extensively been discussed by those papers which address timed test sequence generation. So, much works on model based testing have considered as formal modelling techniques Alur and Dill's timed automata [1] or time Petri nets [32]. However, all this models cannot enable to model the suspension and resumption of a task or any kind of executable portion of code in real-time systems (think, e.g., of

interrupting a washing machine in order to remove a pencil from a shirt, and closing the machine immediately after).

This paper addresses timed test sequence generation for a timed formal model which takes suspend/resume operations into account. We indeed consider Input/Output Prioritized Time Petri Nets with Stopwatches (I/OPrSwTPN), an extension of Merlin's Time Petri Nets [32] with a suspend/resume capability and static priorities. Such priorities are pervasive in many applications of real-time systems. The proposal implements an online testing approach and defines a relativized conformance relation named *rswtioco* (a stopwatch extension of the *rtioco* relation defined in [33]). Unlike other approaches based on offline testing, we do accept unrestricted non-deterministic and partially observable specifications.

The paper is organized as follows. Section 2 surveys related work. Section 3 presents the I/OPrSwTPN model (syntax and semantics). The *rswtioco* relation is introduced in Section 4. Test generation and execution are discussed in Section 5. Finally, Section 6 concludes the paper.

2. Related Work

2.1. Modeling technique

Much work on timed test sequence generation has considered Alur and Dill's timed automata (TA) [1] as formal modeling techniques (see, e.g. [12], [13], [14], [17], [21], [26], [28], [29], [31], [34], [35], [33] and [36]). This usually requires (symbolic) analysis of the TA model e.g. [33] and [29]. Several extensions of TA have been proposed in the literature in order to facilitate and to improve real time system modeling (e.g. [2], [8], [15], [16], [18], [22] and [24]). We have noticed that:

- Part of these extended TA cannot be analysed using existing tools, particularly using the forward analysis technique implemented by UPPAAL [9]. Therefore, several authors (e.g. [3], [6] and [11]) proposed to transform TA into Time Petri nets (Merlin's model [32]) and to reuse verification algorithms available for TPNs.
- The extension dedicated to model suspension and resumption of actions, like for example stopwatch automata [], are not considered at all in timed testing.

Therefore, we decided to select TPN as starting point for test generation. Unlike papers that limit discussion to Merlin's TPN [30], this paper addresses Input/Output Prioritized TPN with Stopwatches. That model enables modelling of suspend/resume operations and the interactions of the reactive real-time systems.

2.2. Online vs. offline testing

The test generation algorithm proposed in Section 5 implements an online (on the fly) policy. Given that real-time systems are intrinsically non deterministic and because of dense time, a timed test case cannot be represented by a finite tree in offline testing; indeed, test cases and their verdicts are calculated a priori and before execution. Several authors brought solutions that consist in determinizing explicitly the specification (see, e.g., [17], [27]); although [1] demonstrated that (1) TA cannot be determinized in general, and (2) that it is sometimes impossible to withdraw internal actions [20]. The result is that [14], [21], [25], [29] and [36] only address a subclass of TA. A solution to address a model with full expressiveness is to use online test. The latter indeed enables working with non deterministic specifications. Non deterministic specifications can be used if the cause of some decision in unknown or the details that determine the decision are abstracted away. Online testing (1) combines test generation and execution and the specification is determinized implicitly on the fly, (2) dramatically lowers the state explosion risk, since only a subset of the states needs to be stored at any point of time

and (3) it may run for several hours or days, and consequently it may exhibit complex and long test sequences.

2.3. Relativized Conformance relation

Often, the SUT operates in specific environments, and it is only necessary to establish correctness under the modelled environment assumptions. Therefore, and as in [33] and [], we make a distinction between the specified system that is called *controller* and the environment of the system that is called *environment*. The assumptions about the environment are modeled explicitly and will be taken into account during test sequence generation. So, modeling the environment explicitly and separately from the system makes it possible to synthesize only those scenarios which are relevant and realistic for the given type of environment. This in turn reduces the number of required tests and improves the quality of the test suite (see [33] for other advantages). Otherwise, it is possible to create a fully open environment for the controller. This is achieved if the environment can send (and receive) any stimuli at any time i.e. a completely unconstrained one that allows all possible interaction sequences (such environment can at any time synchronise with the external actions of the system). We assume than that the test specification is given as a closed system partitioned into one I/OPrSwTPN modelling the behaviour of the SUT (the controller) and one I/OPrTPN modelling the behaviour of its environment. The upper part of figure 1. shows the model partitioned as described above and the lower part shows the system under test (SUT) and the tester. Therefore, conformance between an implementation and its specification is heavily dependent on the environment. Test verdicts obtained for a specific environment remain valid for more restrictive environments. Overall, the conformance addressed by the paper is said to “relativized” since results are obtained for the considered environment.

Following [33], the paper considers a relativized conformance relation (rswtioco) which extends the tioco relation proposed by [31], itself relying on Tretman's ioco relation [37]. The relation's name includes "sw" by reference to Stopwatch TPN.

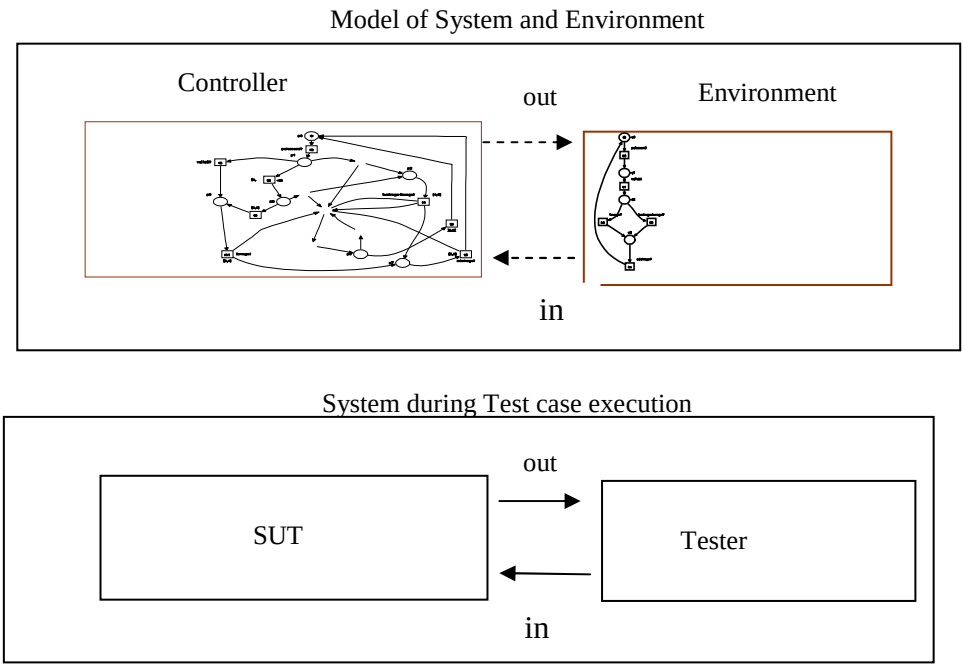


Fig. 1. Model-based testing

3. Input/Output prioritized time Petri nets with stopwatches

During a test process, it is useful to know whether the execution of an action is to be made at the initiative of the system environment (case of input or reception), or whether the system itself activates the execution (case of output or emission). To make the difference between emission and reception of actions, the set of all actions A is partitioned in two disjoint sets of input actions A_{in} and output actions A_{out}. An input (output) is post fixed by ? (!). In addition, we assume the existence of a specific action named *internal* or *unobservable* action and denoted by τ $(\tau \notin A)$. It models the internal events of a system witch are not observed by the tester. They may result from an abstraction of low level details made to facilitate the modelling or to allow a certain freedom to the implementor or more to events which we do not want that the tester to

observe them to facilitate its task. A_τ abbreviates $A_{in} \cup A_{out} \cup \{\tau\}$. $R_{\geq 0}$ and $Q_{\geq 0}$ are the sets of nonnegative real and rational numbers, respectively.

3.1. Timed Input Output Transitions

Timed Input/Output Transition systems (TIOTS) describe systems which combine discrete and continuous transitions. They will be used to describe the semantics of I/OPrSwTPN.

Definition 1. A TIOTS over a finite set of actions, which distinguishes between inputs and outputs, is a quintuplet $\mathcal{A} = (Q, q_0, A_{in}, A_{out}, \rightarrow)$ where Q is a possibly infinite set of states, $q_0 \in Q$ is the initial state and the transition relation $\rightarrow \subseteq Q \times (A_\tau \cup R_{\geq 0}) \times Q$ is decomposed into discrete transitions $\xrightarrow{a}$ (with $a \in A_\tau$) and continuous (delay) transitions $\xrightarrow{d}$ (with $d \in R_{\geq 0}$). The continuous relation satisfying the following properties:

- Nul-Delay: if $q \xrightarrow{0} q$ then $q = q'$,
- Additivity: if $q \xrightarrow{d} q'$ and $q' \xrightarrow{d''} q''$ with $\mathrm{d}, \mathrm{d}' \in R_{\geq 0}$ then $q \xrightarrow{d'+d''} q''$
- Continuity: if $q \xrightarrow{d'+d''} q'$ then $(\exists q'')\left(q \xrightarrow{d'} q'' \wedge q'' \xrightarrow{d''} q'\right)$,
- Temporal determinism: if $q \xrightarrow{d} q' \wedge q \xrightarrow{d} q''$ with $d \in R_{\geq 0}$ then $q' = q''$.

Let $a, a_0, ..., a_k \in A$, $\alpha_0, ..., \alpha_n \in A_\tau$, $\alpha \in A_\tau \cup R_{\geq 0}$ and $d_0, ..., d_{n+1} \in R_{\geq O}$. An execution ρ of a TIOTS $\mathcal{A}$ is a finite sequence of continuous and discrete transitions. It can be written as an alternation of continuous transitions (possibly of duration 0) and discrete transitions: $\rho = q_0 \xrightarrow{d_0} q'_0 \xrightarrow{\alpha_0} q_1 \xrightarrow{d_1} q'_1 \xrightarrow{\alpha_1} \dots q_n \xrightarrow{d_n} q'_n$.

The transition relation $\Rightarrow$ is the relation $\rightarrow$ where internal actions were abstracted ($\Rightarrow \in (A \cup R_{\geq 0})^*$). We have: $q \overset{a}{\Rightarrow} q'$ iff $q \xrightarrow{\tau} * \xrightarrow{a} \xrightarrow{\tau} * q'$, and $q \overset{d}{\Rightarrow} q'$ iff $q \xrightarrow{\tau} * \xrightarrow{d_0} \xrightarrow{\tau} * \xrightarrow{d_1} \xrightarrow{\tau} * \ldots \xrightarrow{\tau} * \xrightarrow{d_n} \xrightarrow{\tau} * q'$ where $d = d_0 + d_1 + \ldots + d_n$. The relation $\Rightarrow$ is extended to sequences of delays and actions. We write: $q \xrightarrow{\alpha}$ iff $q \xrightarrow{\alpha} q'$ and $\xrightarrow{\alpha} q$ iff $q' \xrightarrow{\alpha} q$ for some q'.

Definition 2. An observable timed trace of an execution ρ is the timed word $\sigma \in (A \cup R_{\geq 0})^*$ which is of the form $\sigma = Trace(\rho) = d_0 a_0 \ldots a_k d_{k+1}$

We assume that the TIOTS $\mathcal{A}$ is *strongly input enabled* and *non-blocking*. It is strongly input enabled iff $q \xrightarrow{i}$ for all states q and all the input actions i and non-blocking iff for any state q and any $d \in R_{\geq 0}$ there is a timed output trace $\sigma = d_1 o_1 \ldots o_n d_{n+1}$ such that $s \overset{\sigma}{\Rightarrow}$ and $\sum_i d_i \geq d$. That $\mathcal{A}$ will not block time in any input enabled environment.

We define the timed observable traces of a state q as: $\mathbf{TTr}(q) = \left\{ \sigma \in (A \cup R_{\geq 0})^* \mid q \overset{\sigma}{\Rightarrow} \right\}$

For a state q, and subset $Q' \subseteq Q$ and a timed trace σ, q **after** σ is the set of states which can be reached after σ: q **after** $\sigma = \left\{ q' \mid q \overset{\sigma}{\Rightarrow} q' \right\}$, Q' **after** $\sigma = \bigcup_{q \in Q'} q$ **after** σ

3.2. Input/Output Prioritized Time Petri nets with Stopwatches

Time Petri Nets with Stopwatches (SwTPN) [7], extend Merlin's Time Petri Nets [32] by stopwatch arcs that control the progress of transitions to express suspension and resumption of actions. TPN's are obtained from PN's by associating a temporal interval [Tmin, Tmax] with each transition, specifying firing delays ranges for the transitions.

Tmin and Tmax respectively denote the earliest and latest firing times of the transition (after the latter was enabled). Prioritized Time Petri Nets with Stopwatches (PrSwTPN) extend SwTPN 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 SwTPN, and in particular Prioritized Time Petri nets can be considered equivalent to timed automata, in terms of weak bisimulation []. Since we address the test of reactive systems, we also add an alphabet of actions *A* and a labelling function for transitions. *A* is partitioned in two separate subsets: *inputs actions* A_{in} and *outputs actions* A_{out}. Inputs are the stimuli received by the system from the outside environment. Outputs are the actions sent by this system to the environment. Let $\mathbf{I}^+$ be the set of nonempty real intervals with nonnegative rational endpoints. For $i \in \mathbf{I}^+, \downarrow i$ 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 $\{x - \theta \mid x \in i \wedge x \geq \theta\}$.

Definition 3. An Input/Output Prioritized Time Petri Net with Stopwatches (or I/OPrSwTPN) is a *tuple* $N = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \boldsymbol{Sw}, Pr, m_0, Is, A_\tau, L \rangle$, where :

- $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0 \rangle$ is a *Petri Net*. *P* is the set of *places* and *T* is the set of *transitions*, with $P \cap T = \phi$. $m_0 : P \to \mathbf{N}^+$ is the *initial marking*. $\mathbf{Pre}, \mathbf{Post} : T \to P \to \mathbf{N}$ are the *precondition* and *post-condition functions*.

- $I_S : T \to \mathbf{I}^+$ is the *Static Interval Function* which associates a temporal interval $I_S(t) \in \mathbf{I}^+$ 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.

- $Pr \subseteq T \times T$ is the *priority relation*, assumed irreflexive, asymmetric and transitive. $(t_1, t_2) \in Pr$ is written $t_1 \succ t_2$ or $t_2 \prec t_1$ (t_1 has priority over t_2).

- A is a finite set of *actions*, or *labels*, not containing the internal action τ .

- $L : T \to A_\tau$ is the *labelling* function.

$\boldsymbol{Sw} : T \to P \to \mathbf{N}$ is the *stopwatch incidence function*. $\boldsymbol{Sw}$ associates an integer with each $(p, t) \in P \times T$, values greater than 0 are represented by special arcs, called *stopwatch arcs*, possibly weighted, and characterized by square shaped arrows. Note that these arcs do not convey tokens. Figure 2 shows an I/OPrSwTPN. The arc from place p_0 to transition t_2 is a stopwatch arc of weight 1. The firing of t_0 will freeze the timing evolution of t_2. t_2 will be fireable when its total enabling time reaches 2 time units. If we replace the stopwatch arc by a normal *pre* arc, t_2 will never be fired because of the continuous enabling condition (for more details see [7]).

A *marking* is a function $m : P \to \mathbf{N}^+$. As usual, a *transition t* is *enabled* at *marking* m iff $m \geq \mathbf{Pre}(t)$. In addition, a transition t enabled at m is "*active*" iff $m \geq \boldsymbol{Sw}(t)$, otherwise it is said "*suspnded*". The sets of enabled, active and suspended transitions at m are respectively denoted by:

- $En(m) = \{ t \mid \mathbf{Pre}(t) \leq m \}$,

- $Ac(m) = \{ t \mid t \in En(m) \wedge m \geq Sw(t) \}$ and

- $Su(m) = \{ t \mid t \in En(m) \wedge m < Sw(t) \}$.

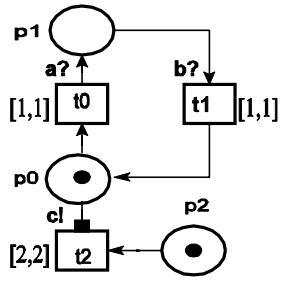


Fig. 2. I/OPrSwTPN example.

The predicate specifying when t' is newly enabled by the firing of t from marking m is defined by: $\uparrow enabled\ (t', m, t) = t' \in En\ (m - \mathbf{Pre}\ (t) + \mathbf{Post}\ (t)) \wedge$

$(t' \notin En\ (m - \mathbf{Pre}\ (t)) \vee t = t')$

Definition 4. A state of an I/OPrSwTPN is a pair (m, I) in which m is a marking and $I : T \to \mathbf{I}^+$, a partial function called the *interval function*, associates a temporal interval in $\mathbf{I}^+$ with every transition $t \in En(m)$.

Definition 5. The semantics of an I/OPrSwTPN $\langle P, T, \mathbf{Pre}, \mathbf{Post}, \boldsymbol{Sw}, Pr, m_0, Is, A_\tau, L \rangle$ is the TIOTS $(Q, q_0, A_{in}, A_{out}, \to)$ where Q is the set of states (m, I) of the I/OPrSwTPN, $q_0 = (m_0, I_0)$ is the initial state, where $I_0 = I_S[En(m_0)]$ is the static interval function I_S restricted to the transitions $En(m_0)$ and $\to \subseteq Q \times T \cup R_{\geq 0} \times Q$ is the state transition relation. It corresponds to two kinds of transitions: discrete transitions are the result of firings transitions of the net and continuous (or delay) transitions are the result of elapsing of time. These transitions are defined as follows, respectively:

– Discrete transitions: $(m, I) \xrightarrow{L(t)} (m', I')$ **iff** $t \in T, L(t) \in A_\tau$ **and**

1. $t \in En(m) \wedge t \in Ac(m)$

2. $0 \in I(t)$

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

4. $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$

5. $(\forall k \in T)(k \in En(m') \Rightarrow I'(k) = \textbf{if } \uparrow enabled(k, m, t) \textbf{ then } I_S(k) \textbf{ else } I(k))$

– Continuous transitions: $(m, I) \xrightarrow{d} (m, I')$ **iff** $d \in R_{\geq 0}$ **and**

6. $(\forall k \in T)(k \in En(m) \wedge k \in Ac(m) \Rightarrow d \leq \uparrow I(k))$

7. $(\forall k \in T)\left(k \in En(m) \Rightarrow I'(k) = \textbf{if } k \in Ac(m) \text{ then } I(k) \dot{-} d \textit{ else } I(k)\right)$

The transition t may fire from (m, I) if (1) it is enabled and active at m, (2) fireable instantly, and (3) no transition with higher priority satisfies these conditions. These

conditions ensure that only active transitions may fire. (4) is the standard marking transformation. From (5), in the target state, the transitions not in conflict with *t* (transitions that remained enabled while *t* fired, *t* excluded) retain their firing intervals, whereas those newly enabled are assigned their static intervals. Firing a transition takes no time. By (7), all firing domains of active transitions are shifted synchronously towards the origin as time elapses, and truncated to nonnegative values. The elapsing of time has sense only for active transitions and changes of dates are thus made only for these transitions. Frozen (suspended) transitions have their temporal interval unchanged. (6) prevents time to elapse as soon as the latest firing time of some active transition is reached.

Clocks take their values in the set of nonnegative real numbers (dense time), and thus a state may admit an infinity of successors states, which implies that the state space of a I/OPrSwTPN may be infinite. Finitely representing state spaces involves grouping some sets of states. Several grouping can be defined, depending on the properties of the state space one would like to preserve. We use the grouping method introduced in [4] which groups some particular sets of states into state classes and preserve marking and traces. A state class, or a symbolic state, is a pair (*m*, *D*) where *m* is a marking of the net and *D* is a firing domain of the possible firings of the enabled transitions at *m*. The domain *D* is described by a system of linear inequalities $W\underline{\phi} \leq \underline{\omega}$. Variables $\underline{\phi}$ are bijectively associated with the transitions enabled at *m*. The symbolic transition relation between state classes is decomposed to:

– $(m, D) \xrightarrow{L(t)} (m', D')$ **iff** $t \in T \wedge L(t) \in A_\tau$ **and**

1. $t \in En(m) \wedge t \in Ac(m)$

2. $0 \in \{\underline{\phi}_t\}$ ($\{\underline{\phi}_t\}$ is the set of solutions of the variable $\underline{\phi}_t$ in the system *D*).

3. $(\forall k \in T)\left(k \in En(m) \wedge k \in Ac(m) \wedge (k \succ t) \Rightarrow 0 \notin \{\underline{\phi}_k\}\right)$

4. $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$

5. $(\forall k \in T)\left(k \in En(m') \Rightarrow \underline{\phi}'_k = \textbf{ if } \uparrow enabled(k,m,t)\right) \textbf{ then } \underline{\phi}'_k \in I_s(k) \textbf{ else } \underline{\phi}_k\Big)$

6. the variables $\underline{\phi}$ are eliminated

– $(m,D) \xrightarrow{d} (m,D')$ i**ff** $d \in R_{\geq 0}$ **and**

7. $(\forall k \in T)\left(k \in En(m) \wedge k \in Ac(m) \Rightarrow \left(d \leq \mathbf{max}\{\underline{\phi}_k\}\right)\right)$ ($\mathbf{max}\{\underline{\phi}_k\}$ is the maximal value).

8. $(\forall k \in T) \Rightarrow \left(k \in En(m) \Rightarrow \underline{\phi}'_k = \textbf{ if } k \in Ac(m) \textbf{ then } \underline{\phi}_k - d \textbf{ else } \underline{\phi}_k\right)$

Informally, the system leaves the initial state $c_\varepsilon = \left(m_0, \{\downarrow I_s[En(m_0)] \leq \underline{\phi}_t \leq \uparrow I_s[En(m_0)]\}\right)$ then make alternately two types of transitions: the transitions of active actions if the current value allows it and the transitions of time which decrease the intervals of the active transitions of the same duration by respecting the date of firings as soon as possible of each of the transitions. The time in the suspended transitions is frozen. So, when a frozen transition becomes active again, due to a change in marking, it resumes with the temporal interval captured in the state rather than its static interval.

In order to test preemptive real-time systems, we distinguish between two types of outputs. First, outputs in the common sense of the word; we call them *active outputs*. Second, special outputs that we call "*indicators*" or *suspended outputs*. The latter are issued by the systems to give indications on suspended actions. For correct behaviour of a preemptive real-time system, a response which corresponds to an active output, resumed or not, and/or suspended output(s)) should not only provide correct values, but the values should also be provided at the right time points. So, delay is also considered as an output.

The set of observable active outputs or delays that can occur in $q \in \mathrm{Q}' \subseteq \mathrm{Q}$ is defined as: $\mathbf{Out}_{aord}(q) = \{a \in A_{out} \cup R_{\geq 0} \mid q \overset{a}{\Rightarrow}\}$, $\mathbf{Out}_{aord}(\mathrm{Q}') = \bigcup_{q \in \mathrm{Q}'} \mathbf{Out}_{aord}(q)$

The set of observables suspended outputs that can occur in $q \in \mathrm{Q}' \subseteq \mathrm{Q}$ is defined by (the function *su* is extended to states):

$$\mathbf{Out}_{su}(q) = \{a \in su(q) \mid \overset{\alpha}{\Rightarrow} q \wedge \alpha \in A_{out} \cup R_{\geq 0}\}, \quad \mathbf{Out}_{su}(\mathrm{Q}') = \bigcup_{q \in \mathrm{Q}'} \mathbf{Out}_{su}(q)$$

Definition 6. Let $\mathcal{Q} = (Q, q_0, A_{in}, A_{out}, \rightarrow)$ be the input enabled and non-blocking TIOTS describing the semantics of the specification and $\mathcal{E} = (E, e_0, A_{out}, A_{in}, \rightarrow)$ is an input enabled and non-blocking TIOTS associated to the environment model of $\mathcal{Q}$. The set of input (output) actions of $\mathcal{E}$ is identical to the output (input) actions of $\mathcal{Q}$ and the environment model not contains suspended actions. The parallel composition of $\mathcal{Q}$ and $\mathcal{E}$ forms a closed system $\mathcal{Q} \| \mathcal{E}$ in with observable behaviour is defined by the TIOTS $(Q \times E, (q_0, e_0), A_{in}, A_{out}, \rightarrow)$ where $\rightarrow$ is defined as:

$$\frac{q \xrightarrow{a} q' \quad e \xrightarrow{a} e'}{(q,e) \xrightarrow{a} (q',e')} \quad \frac{q \xrightarrow{\tau} q'}{(q,e) \xrightarrow{\tau} (q',e')} \quad \frac{e \xrightarrow{\tau} e'}{(q,e) \xrightarrow{\tau} (q',e')} \quad \frac{q \xrightarrow{d} q' \quad e \xrightarrow{d} e'}{(q,e) \xrightarrow{d} (q',e')}$$

4. The rswtioco conformance relation

The motivation behind an introduction of the Relativized Stopwatch Timed Input/Output Conformance relation, or *rswtioco* for short, is to test real-time systems and to take into account their suspend/resume operations. *rswtioco* extends *rtioco* [33], the latter being itself an extension of *ioco* and *tioco* relations by taking time and environment assumptions explicitly into account [37], [31] and [19]. Unlike *ioco* and *tioco*, *rtioco* distinguishes between the system's constraints and the environment's ones. The latter are explicitly and separately modelled from the former. The question "does the implementation conform to its specification?" is answered not for any type of

possible environment but for the considered environment i.e. the environment under which the SUT will operate. A “yes” answer to the previous question which has been obtained for one environment still applies to more restrictive environments. A relativzed conformance relation can be helpful to give restrictions of the environment to avoid generating and executing uninteresting test cases. These restrictions can also be seen as guiding to especially wanted test cases. So, in order to test the suspension/resumption of an action *a* we have to take into account the input to supply to the SUT, and also when to supply it, that enable to suspend/resume the action *a*. This can be done by the choice of (1) the environment model, (2) the choice of the input to supply by the function *chooseAction* and (3) its timing by the function *chooseDelay* (see Algorithm 1).

The rswtioco relation does not allow either of “standard” outputs and ‘’indicators’’ to be emitted in advance or on late, by the system. Also, this relation allows having more information about the non-conformance of a system. So, when the system emits an indicator or an output that was not expected at that time, then we can know if that indicator (resp. output) must be an active output (resp. an indicator) or nothing (see algorithm 1). The proposed *rswtioco* relation makes it possible to answer another question: “does some action *a* resume at the expected date? i.e. *rswtioco* does not allow a suspended action to be resumed in advance or on late. Under assumptions of input enabledness, the *rswtioco* relation coincides with relativized timed trace inclusion. Timed Traces of the SUT operating under an environment must be included in those of the specification under the cover of the same environment.

Definition 7. Given an environment *e*, the conformance relation $rswtioc_e$ between system states $q, t \in Q$ is defined by:

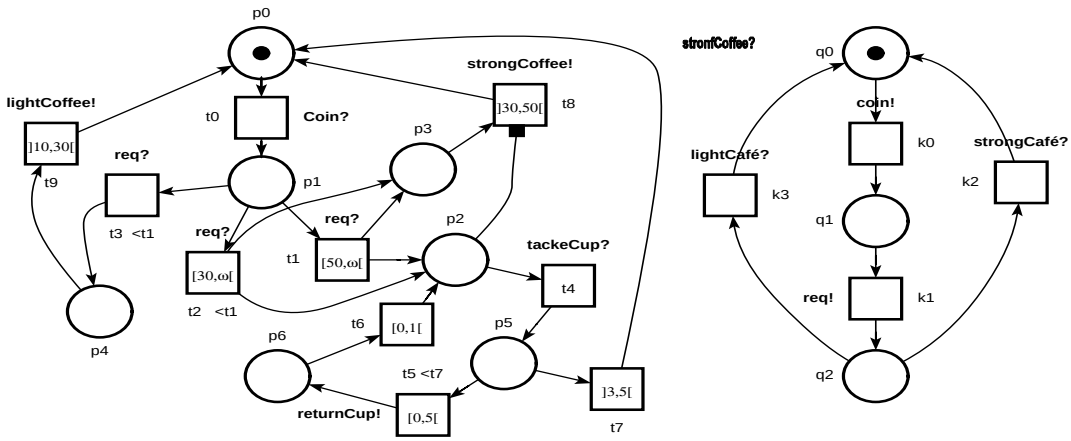
$$q \; rswtioco_e \; t \quad \text{ssi} \quad \forall \sigma \in \mathrm{TTr}(e): \mathbf{Out}_{aord}((q,e)\,\mathbf{after}\,\sigma) \subseteq \mathbf{Out}_{aord}((t,e)\,\mathbf{after}\,\sigma) \wedge$$

$$\mathbf{Out}_{su}((q,e)\mathbf{after}(\mathbf{Out}_{aord}((q,e)\,\mathbf{after}\,\sigma))) \subseteq \mathbf{Out}_{su}((t,e)\,\mathbf{after}(\mathbf{Out}_{aord}((t,e)\mathbf{after}\,\sigma)))$$

Whenever q $rswtioco_e$ t we say that q is a correct implementation of the specification t under the environment constraints expressed by e.

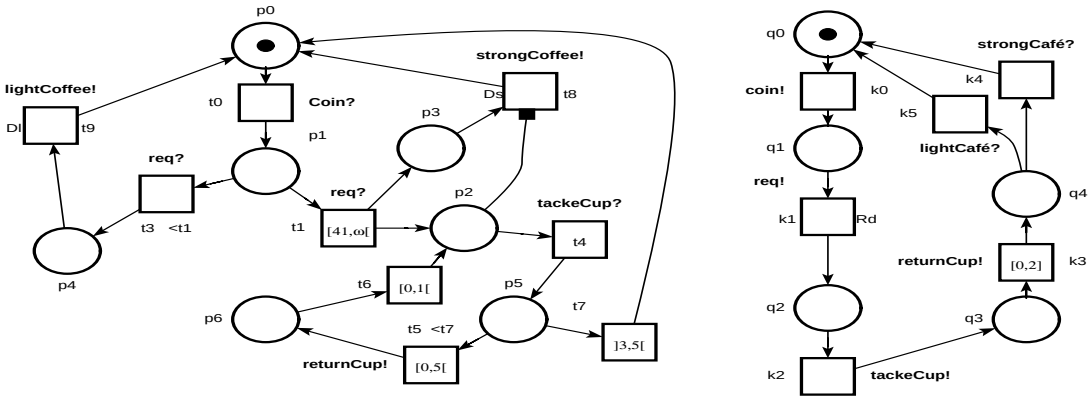
There is a most (least) discriminating input enabled and non-blocking environment U (O) given by $TTr(U) = (A \cup R_{\geq 0})^*$ $\left(TTr(O) = (A_{out} \cup R_{\geq 0})^*\right)$. The corresponding conformance relation $rswtioco_U$ ($rswtioco_O$) specializes to simple trace inclusion (timed output trace inclusion) between system states. In Figure 5(b) and Figure 5(c) the most-discriminating and the least discriminating environments are given when $A_{in} = \{\text{req}, \text{coin}, \text{tCup}, \text{rCup}\}$ and $A_{out} = \{\text{wWoffe}, \text{sCoffee}\}$.

Example. This example is taken from [33] and enriched with suspension/resumption and internal actions. Figure 3(a) shows an I/OPrSwTPN specifying the requirements to a coffee machine. It has a facility that allows the user, after paying, to indicate his/her eagerness to get coffee by pushing a request button on the machine forcing it to output coffee. However, allowing insufficient brewing time results in a light coffee. Waiting less than 30 time units definitely results in a light coffee, and waiting more than 50 definitely results in a strong coffee. Between 30 and 50 time units the choice is non-deterministic, meaning that the SUT/implementor may decide what to produce. After the request, it takes the machine an additional (non deterministic) 10 to 30 (30 to 50) time units to produce light coffee (strong coffee). The user requesting for strong coffee can take his/her coffee at any time during its preparation and can again put back the cup to resume what remains in the machine, on the condition to not exceed 3 time units. The machine makes internal actions to be reset or to resume the preparation of strong coffee. This service is not allowed for the user requesting light coffee. The figure 3(b) models potential (nice) users of the machine that pay before requesting coffee and take his coffee after its preparation.



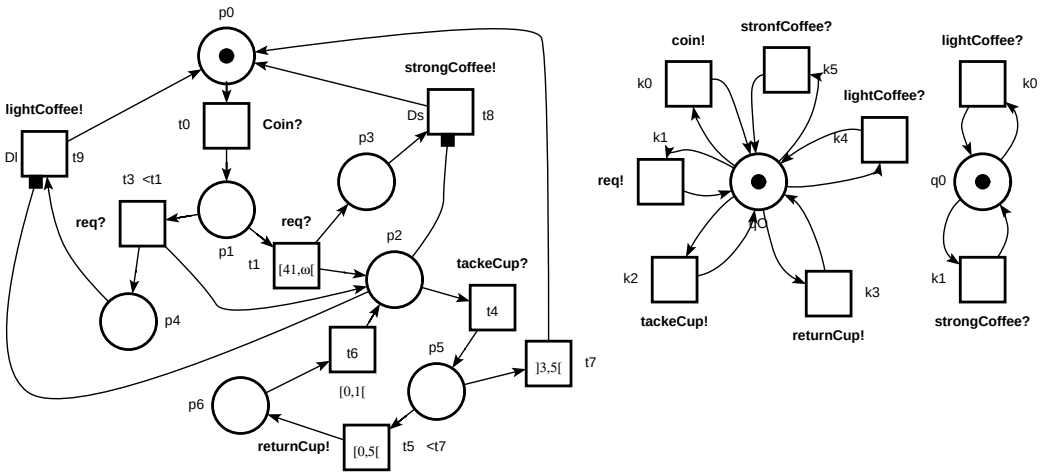
a) Specification $\mathscr{C}_C$ of a machine coffee (b) Example environment $\mathscr{E}_C$

Fig. 3. Specification of machine coffee and an environment models.



(a) IUT: $\mathscr{I}_1(D_s,)$. (b) Environment $\mathscr{E}_1([Rd,\infty[)$

Fig. 4. Implementation of coffee machine



(a) IUT: $\mathscr{I}_2([Ds, Ds], [Dl, Dl])$. (b) Environment $\mathscr{E}_U$ (c) Environment $\mathscr{E}_O$

Fig. 5. An other implementation of coffee machine (Dl and Ds are intervals)

To illustrate our approach we suppose that the SUT can be modelled as an I/OPrswTPN. The (deterministic) implementation $\mathcal{I}_1$([Ds, Ds], [Dl, Dl]) in Figure 4(a) produces light coffee (strong coffee) after than less 40 time units (more than 41 time units) and an additional brewing time of Dl (Ds) time units. Notice that $\mathcal{I}_1$([Ds, Ds], [Dl, Dl]) does not allow the user requesting light coffee to take his cup before this time. Observe that any trace of the implementation $\mathcal{I}_1$([40, 40], [20, 20]) (in any environment) can be matched by the specification; hence $\mathcal{I}_1$([40, 40], [20, 20]) $rswtioco_{\mathcal{E}U}$ $\mathcal{Q}_C$. Thus also $\mathcal{I}_1$([40, 40], [20, 20]) $rswtioco_{\mathcal{E}1}$ $\mathcal{Q}_C$. In contrast, $\mathcal{I}_1$([70, 70], [5, 5]) $\cancel{rswtioco}_{\mathcal{E}U}$ $\mathcal{Q}_C$ for tow reasons: 1) it has the timed trace coin . 30 . req. 5 lightCoffee that $\mathcal{Q}_C$ does not, i.e. it may produce light coffee too soon (no time to insert a cup); 2) it has a trace coin . 50 . req . 70 not in $\mathcal{Q}_C$ meaning that it produces strong coffee too slowly. The implementation $\mathcal{I}_2$([Ds, Ds], [Dl, Dl]) in Figure 5(a) is different from $\mathcal{I}_1$([Ds, Ds], [Dl, Dl]), it allows all users requesting coffee to take it during its preparation (including those requesting light coffee). We have $\mathcal{I}_2$([40, 40], [20, 20]) $\cancel{rswtioco}_{\mathcal{E}U}$ $\mathcal{Q}_C$ and $\mathcal{I}_2$ ([40, 40], [20, 20]) $\cancel{rswtioco}_{\mathcal{E}1}$ $\mathcal{Q}_C$ because it has the timed trace coin . 30 . req. 10 (tackeCup, lightCoffee). 2. (returnCup, lightCoffee). 5. lightCoffee that $\mathcal{Q}_C$ does not. (tackeCup, lightCoffee) means that tackeCup is an active action and lightCoffee is a suspended action. In contrast, $\mathcal{I}_2$([40, 40], [20, 20]) $rswtioco_{\mathcal{E}C}$ $\mathcal{Q}_C$ and $\mathcal{I}_2$([40, 40], [20, 20]) $rswtioco_{\mathcal{E}1}$ $\mathcal{Q}_C$ if $Rd = [60, \infty[$ because $\mathcal{E}_C$ never takes up his cup while the machine preparing coffee and $\mathcal{E}_1$ (60) never requests light coffee.

5. Generations and execution of test cases

The inputs to algorithm 1 are two TIOTS's $\mathcal{Q} \| \mathcal{E}$ describing the semantics of two I/OPrSwTPN's, respectively modelling the SUT and an environment. The algorithm maintains after every execution of a test event (a sent of an input or an observed output

or a delay), the current reachable state set $C \subset Q \times E$ by using the symbolic technique implemented in TINA [5] adapted to the needs of test. The tester is thus a state estimator; it occupies a set of symbolic states and modifies it after every test event. Knowing the set *C*, we can choose the appropriate test primitive and validate the SUT outputs. *C* initially contains the symbolic state c_ε. The tester can perform three basic actions: either send an input (an enabled environment output) to the SUT, or wait for an output after a delay or still reset the SUT and restart. If an output or a delay is observed, the tester verifies if this is conforming to the specification. Any illegal occurrence or absence of a standard output is detected if the modification of the set *C* becomes empty, which happens when the observed trace is not in the specification. The illegal occurrence of a suspended action is detected if it does not belong to *ImpSuspend(C).* The function *After* calculates the accessible symbolic states after the execution of a test event from the current states *C*. It returns an empty set if this event was not authorized by the specification. The functions used in Algorithm 1 are defined as:

- *ChooseAction* selects randomly an input in the environment model applicable to the SUT from the set *C*.

- $Choosedelay(C) = \left\{ d \in R_{\geq 0} \middle| \exists (q,e) \in C.e \overset{d}{\Longrightarrow} \right\}$. Delays can not be randomly chosen from all the set of real numbers if the environment must offer an input to the SUT before certain date.

- $EnvOutput(C) = \left\{ a \in A_{in} \middle| \exists (q,e) \in C.e \xrightarrow{a} \right\}$, *EnvOutput* is empty if the environment has no output to offer.

- $ImpOutput(C) = \left\{ a \in A_{out} \middle| \exists (q,e) \in C.q \xrightarrow{a} \right\}$.

- $ImpSuspend(C) = \left\{ a \in A \middle| \exists (q,e) \in C \wedge a \in su(q) \right\}$.

- $active(o)$ (resp. $suspend(o)$) calculates the active output (resp. the suspended actions). The output *o* is a pair (active output, suspended actions). Each of them can be empty. The *suspend* function is extended to delays.

Algorithm 2 computes the function $\text{Closure}_{\delta\tau}(C,d)$ that collects the reachable symbolic state set within a delay of d. The predicate $\text{Contains}(C,(m,D))$ tests whether a symbolic state (m,D) is covered by a symbolic state in *C*. Sol(D) is the set of solutions of the temporal variables associated with the enabled transitions. The function $\text{Closure}_{\tau}(C)=\text{Closure}_{\delta\tau}(C,0)$ that collects the reachable symbolic state set after all possible internal transition in zero delay can be computed similarly. Given this function, the actual algorithms for computing *After*(*C*, *a*) and *After*(*C*,δ) become trivial: $After(C,a)=\text{Closure}_{\tau}\left(\left\{(m',D')\,\middle|\,(m,D)\in \text{Closure}_{\tau}(C),(m,D)\xrightarrow{a}(m',D')\right\}\right)$, and $After(C,\delta)=\left\{(m,D)\,\middle|\,(m,D)\in \text{Closure}_{\delta\tau}(C,\delta)\right\}$.

Algorithm 1 Generation and execution of test.

GenExeTest(S,E,SUT,N), $C:=\{c_{\varepsilon}=(m_0,D_0)\}$

while $C\neq\phi\wedge iterations\leq N$ **do** *RondomlyChoose*(*Action*, *Delay*, *Restart*)

Action: // offer an input to the SUT

If *EnvOutput*(*C*) ≠ ∅ **then**
- *a* := *ChooseAction*(*EnvOutput*(*C*))
- sent *a* to the SUT
- *C* := *After*(*C*, *a*)

Delay: // wait for an output of the SUT

δ := *ChooseDelay*(*C*) // Wait δ unit of time and test the output *o* (*o* contains eventually
// suspended actions) sent by the SUT.

if *o* occurs at δ' ≤ δ **then**
- *C* := *After*(*C*, δ')
- **if** *active*(*o*)∉ *ImpOutout*(*C*)
  - **then**
    - **return** *fail*
    - **if** *active*(*o*)∈ *ImpSuspend*(*C*)
      - **then** "*active*(*o*) must be a suspended action"
  - **else** *C*:= *After*(*C*, *active*(*o*))
- **if** *suspend(o)* ⊄ *ImpSuspend*(*C*)

```
                    then  return fail
                          S= active(o) - ImpSuspend(C)
                          forall a ∈ S  if a ∈ ImpOutout(C)
                                        then  "a must be active"
            else  C := After(C,δ)   // no output during δ
Restart:      // reset and restart.
  C := {c_ε = (m_0, D_0)}
  Reset SUT
If C=∅ then return fail  else  return  pass
```

Algorithm 2 $\text{Closure}_{\delta\tau}(C,d)$

Pass $:= \phi$, Wait $:= C$

while Wait $\neq \phi$ **do** Wait $:=$ Wait $-\{(m,D)\}$

if $(m,D) \xrightarrow{d'} (m,D')$ where $d' \leq d$ **then** $Pass := Pass \cup \{(m,D')\}$

For each transition $(m,D') \xrightarrow{\tau} (m',D'')$

if $\neg contains(Pass,(m',D''))$ **then** Wait $:=$ Wait $\cup \{(m',D'')\}$

return Pass

Contains$(C,(m,D))$

For each state $(m,D') \in C$ si $\text{Sol}(D) \subseteq (\text{Sol}(D'))$ **then** return true

Return false

6. Conclusions and future work

The paper discusses testing of real-time systems modelled using Stopwatch Time Petri Nets. The latter have been selected for their capacity to model suspend/resume operations in real-time systems (whereas surveyed papers on timed testing only address system/environment interactions and timeliness). Using an online testing approach makes is possible to handle non determinism and partly observable systems.

The paper introduces *rswtioco*, a new conformance relation which differs from *tioco* and *rtioco*. It differs from *tioco* because it addresses the constraints captured by the system separately from the ones inherent to the environment. Also, *rswtioco* differs from both *tioco* and *rtioco* because the latter were defined for timed automata, a modelling technique which does not enable description of suspend/resume operations

i.e. operations where the system's context has to be stored and restored later on. *tioco* and *rtioco* do not allow one to distinguish between suspended actions and not enabled ones.

The algorithm proposed in the paper will be soon implemented in TINA [5]. So far, our investigations have been limited to conformance testing. We plan to address other types of testing in the near future (in particular, robustness testing).

References

1. Alur R., Dill D., « A theory of timed automata Theoretical », Computer Science, 126:183–235, 1994.
2. Bérard B. and Dufourd C., « Timed automata and additive clock constraints », Information Processing Letters (IPL), 75(1–2):1–7, 2000.
3. Bérard B., Cassez F., Haddad S., Lime D. and Roux O. H., « When are timed automata weakly timed bisimilar to time Petri nets ? », *In 25th FSTTCS 2005, vol. 3821 of LNCS,* Hyderabad, India, December 2005, Springer.
4. Berthomieu B., M. Diaz, modelling and verification of time dependent systems using time Petri nets, IEEE transactions on software Engineering, 17(3), 1991.
5. Berthomieu B., Ribet P. O., Vernadat F., « The tool TINA -- Construction of Abstract State Spaces for Petri Nets and Time Petri Nets », *Inter. Journal of Production Research, Vol. 42, No 14,* July 2004.
6. 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.
7. Berthomieu B., Lime D., Roux O. H., Vernadat F., « Reachability Problems and Abstract State Space for Timed Petri Nets with Stopwatches », August 2006, To appear 2007.
8. Bouyer P., Dufourd C., Fleury E., and Petit A., « Updatable timed automata », *Theoretical Computer Science, 321(2–3):291–345,* 2004.
9. Bouyer P., « Forward Analysis of Updatable Timed Automata », *Formal Methods in System Design* 24(3), pages 281-320, 2004.
10. Bouyer P., Chevalier F., « On conciseness of extensions of timed automata », *Journal of Automata, Languages and Combinatorics,* 2005.
11. Bouyer P., Serge H., Reynie P. A., « Extended Timed Automata and Time Petri Nets », *in ACSD'06, Turku, Finland, pages 91-100, IEEE Computer SocietyPress,* juin 2006.
12. Braberman V., Felder M., Marre M., « Testing timing behavior of real-time software », *In Intern. Software Quality Week,* 1997.
13. Brinksma E., Tretmans J., « Testing transition systems: An annotated bibliography », *In MOVEP 2000, volume 2067 of LNCS,* Springer, 2001.
14. Cardell-Oliver R., « Conformance test experiments for distributed real-time systems », *In ISSTA'02, ACM Press,* 2002.
15. Cassez F. and Larsen K. G., « The impressive power of stopwatches », *In Proc. 11th Int.CONCUR'0), vol. 1877 of LNCS, p. 138–152, Springer,* 2000.
16. Choffrut C. and Goldwurm M., « Timed automata with periodic clock constraints », *JALC, 5(4):371–404,* 2000.
17. Cleaveland R., Hennessy M., « Testing Equivalence as a Bisimulation Aquivalence », Farmal Aspects of Computing, 5:1-20, 1993.
18. Demichelis F. and Zielonka W., « Controlled timed automata », *In Proc. 9th Int.CONCUR'98, vol. 1466 of LNCS, p. 455–469, Springer,* 1998.
19. de Vries R., Tretmans J., « on-the-fly conformance testing using SPIN», *Software Tools for Technology Transfer, 2(4) : 382-393,* March 2000.

20. 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.

21. En-Nouaary A., Dssouli R., Khendek F., Elqortobi A., « Timed test cases generation based on state characterization technique », *In RTSS'98, IEEE,* 1998.

22; Fersman E., Petterson P., and Yi W., « Timed automata with asynchronous processes: Schedulability and decidability » In *Proc. 8th Int.TACAS'02, vol.2280 of LNCS, P. 67–82, Springer,* 2002.

23. 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.

24. Henzinger T. A., « The theory of hybrid automata », *In Proc. 11th An.LICS'96, p. 278–292. IEEE Computer Society Press,* 1996.

25. 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.

26. 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.

27. Jéron T., Morel P., « Test generation derived from model-cheking *», In Halbwachs and D. peled Editors, CAV'99, Trento, Italy, Volume 1633 of LNCS, pages 108-122. Springer-Verleg,* july 1999.

28. Jéron T., Rusu V., Zinovieva E., « STG: A symbolic test generation tool », *In TACAS'02, volume 2280 of LNCS, Springer,* 2002.

29. Khoumsi A., Jéron T., Marchand H., « Test cases generation for nondeterministic real-time systems », *In FATES'03, Montreal,* October 2003.

30. 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, pp.151-158,* Sept. 2000.

31. Krichen M., Tripakis S., « An Expressive and Implementable Formal Framework for Testing Real-Time Systems », *In 17th IFIP Intl. TestCom'05,* 2005.

32. Merlin P. M., Farber J., « Recoverability of communication protocols: Implications of a theoretical study », *IEEE Trans. Com., 24(9):1036-1043,* September 1976.

33. 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,* September 24, 2004.

34. Nielsen B., Skou. A., « Automated test generation from timed automata », *In TACAS'01, LNCS 2031, Springer,* 2001.

35. Peleska J., « Formal methods for test automation - hard real-time testing of controllers for the airbus aircraft family », *In IDPT'02,* 2002.

36. Springintveld J., Vaandrager F., D'Argenio P., « Testing timed automata », *Theoretical Computer Science, 254,* 2001.

37. Tretmans J., «Testing concurrent systems: A formal approach », *In J.C.M Baeten and S. Mauw, editors, CONCUR'99 – 10th Int. CCT, vol. 1664 of LNCS, p. 46–65. Springer-Verlag,* 1999.