Conformance Testing of Preemptive Real-Time Systems

Noureddine Adjir, Faculty of Exact and Applied Sciences, University of Oran, Oran, Algeria
Pierre de Saqui-Sannes, ISAE, University of Toulouse, Toulouse, France
Kamel Mustapha Rahmouni, Computer Science Department, University of Oran, Oran, Algeria

ABSTRACT
The paper presents an approach for model-based black-box conformance testing of preemptive real-time systems using Labeled Prioritized Time Petri Nets with Stopwatches (LPrSwTPN). These models not only specify system/environment interactions and time constraints. They further enable modelling of suspend/resume operations in real-time systems. The test specification used to generate test primitives, to check the correctness of system responses and to draw test verdicts is an LPrSwTPN made up of two concurrent sub-nets that respectively specify the system under test and its environment. The algorithms used in the TINA model analyzer have been extended to support concurrent composed subnets. Relativized stopwatch timed input/output conformance serves as the notion of implementation correctness, essentially timed trace inclusion taking environment assumptions into account. Assuming the modelled systems are non deterministic and partially observable, the paper proposes a test generation and execution algorithm which is based on symbolic techniques and implements an online testing policy and outputs test results for the (part of the) selected environment.

Keywords: Conformance Testing, Online Testing, Preemptive Real-Time Systems, Stopwatches, Time Petri Nets

1. INTRODUCTION
The embedded real-time industry is changing fast – systems have become larger, more complex, and preemptive. For real-time systems, the timely reaction is just as important as the kind of reaction. Thus the system must not only produce correct result, but must do so at the correct time; neither too early nor too late. Fly-by-wire systems in modern airplanes are an example for such embedded systems. If a pilot hits the brakes, the breaking system should engage almost immediately to ensure secure travelling. Furthermore, real-time systems may be interruptible. They may be interrupted at any time while keeping the capacity to restart later on without losing their state information (think, e.g., of interrupting a washing machine in order to remove a pencil from a shirt, and closing the machine immediately after). Such systems need to be tested in order to check their reliability before use. In testing real-time systems, the tester must consider when to stimulate the system, when to interrupt or resume operations, when to expect responses to be issued and how to assign verdicts to any timed sequence it may
observe and partly control. Further, the test cases must be executed in real-time.

Without automation and modeling tools, testing remains ad hoc, time-consuming and error prone. With the use of models in software/hardware design and development, timed model-based testing has received increasing attention from industry practitioners. Therefore, Timed Model-Based Testing uses timed models describing the desired behaviour of a system to automate the testing process. Using models to generate test cases and assign verdicts is cheaper and more effective than a completely manual approach. Conformance testing is a way of black-box testing in which common testing tasks such as test case generation and test result evaluation are based on a model of the system. Thus, no knowledge about the internal workings of the program to be tested is used and the tests are limited to the functional and timing properties. A survey of the literature indicates that those papers which address timed test sequence generation have extensively discussed reactivity and timeliness. So, much work on model based testing have considered as formal modelling techniques timed automata (TA) (Alur, 1994) or time Petri nets (Merlin, 1976). 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. Therefore a real-time specification model should include a suspend/resume capability. The paper addresses model-based black-box conformance testing of preemptive real-time systems. It checks a System Under Test (SUT) against its specification. This is typically achieved in a controlled environment where the SUT is executed and stimulated with inputs and delays according to a test specification, and the responses of the SUT are checked to conform to its specification. Precisely, the paper presents a technique for conformance testing of preemptive real-time systems based on Labelled Prioritized Time Petri Nets with Stopwatches models (LPrSwTPN). The test specification is given as an LPrSwTPN made up of two composed subnets that respectively model the expected behaviour of the SUT and the latter’s environment. The proposal implements an online testing approach and proposes a relativized conformance relation named rswtioco (Relativized Stopwatch Timed Input Output Conformance), between model and SUT which coincides with timed trace inclusion taking assumptions about the environment behavior explicitly into account. It is an extension of the rtioco relation (Hessel, 2008). In addition to allowing explicit and independent modelling of the environment, it also has some nice theoretical properties that allow testing effort to be reused when the environment or system requirements change. Unlike other approaches based on offline testing, we do accept unrestricted non-deterministic and partially observable specifications.

The rest of paper is organized as follows. Section 2 surveys related work. Section 3 shows what is new in the test of preemptive real-time systems. In section 4, we illustrate and compare the two different approaches to timed testing: offline and online testing. Section 5 describes the test specification. Section 6 introduces LPrSwTPN, their formal semantics in terms of timed labeled transition systems, and their use to model and specify the behavior of real-time systems. The rswtioco relation is presented in section 7. In section 8, we present an online testing algorithm of real-time systems from LPrSwTPN specifications allowing full non-determinism. This algorithm combines test cases generation and their execution. Finally, section 9 concludes the paper.

2. RELATED WORK AND MOTIVATION

Little work has been done on model-based testing from TPNs (e.g. (Adjir, 2009; Lin 2000), the subject being essentially addressed for TA. Several extensions of timed automata have been proposed in the literature in order to facilitate and to improve the modelling and testing of real time systems (e.g. (Bérard, 2000; Bouyer, 2004; Choffrut, 2000). Unfortunately, we notice that: (1) part of these extensions cannot be analyzed
using existing tools as UPPAAL (Bouyer, 2004). Therefore, several authors only addressed subclasses of TA or proposed to transform TA in other models (e.g. TPN (Bérard, 2005; Bouyer, 2006) to reuse their efficient verification algorithms). (2) the extension dedicated to model suspension/resumption of actions, e.g. stopwatch automata (Cassez, 2000) and interrupt timed automata (Bérard, 2012), have not yet efficient analysis methods and are not considered at all in timed testing.

Therefore, we decided to select LPrSwTPN as starting point for timed test generation. That model enables modelling of suspend/resume operations and the interactions of the reactive real-time systems. PN are characterized by their expressive power of parallelism and concurrency, and the conciseness of the models. TPN (Merlin, 1976) are one among the important formal models widely used to specify and verify real-time systems. In addition, the efficient analysis methods proposed by (Berthomieu, 2004) have contributed to their wide use. Chronometers (SwTPN) (Berthomieu, 2007) allow modelling the suspension of actions and their resumption later without losing their information. They may be interrupted at any time while keeping the capacity to restart later on. Adding priorities to TPN (PrTPN) increases their expressiveness. So it is shown in (Berthomieu, 2006) that the expressiveness of PrTPN is very close to that TA, in terms of weak timed bisimilarity. Since we address the testing of reactive systems, we associate a label with each transition (LPrSwTPN). It may be an input or an output or an internal action.

3. THE OUTPUTS OF PREEMPTIVE REAL-TIME SYSTEMS

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

4. ONLINE VS. OFFLINE TESTING

There are two different approaches to timed testing: offline and online testing. In offline test generation test cases and their verdicts are pre-computed completely from the specification before they are executed on the SUT. The advantages of offline test generation are that test cases are easier, cheaper, and faster to execute because all time constraints in the specification have been resolved at test generation time, and in addition, that the test suite can be generated with some a-priori guarantees, e.g., that the specification is structurally covered, or that a given set of test-objectives are satisfied. There are two main disadvantages of offline testing. One is that the specification must be analyzed in its entirety, which often results in a state explosion which limits the size of the specification that can be handled. Another problem for real-time systems is non-deterministic implementations and specifications. In this case, the ordering and timing output cannot be predicted, and the test case must be adaptive. Typically, the timed test case takes the form of a test-tree that branches for all possible outcomes. The test case may need to branch for all time instances where an output could arrive and it cannot be represented by a finite tree. Offline test generators therefore often limit the expressiveness and amount of non-determinism of the specification language. This has been a particular problem for offline test generation from timed models specifications, because the technique of determinizing the specification cannot be directly applied. For example, for the methods based on TA, several authors brought solutions that consist in determinizing explicitly the specification;
although (1) TA cannot be determinized in general (Alur, 1994), and (2) that it is sometimes impossible to withdraw internal actions (Diekert, 1997). The result is that some works only address a subclass of TA. Given a restricted class of deterministic and output urgent TPN, we have showed in (Adjir, 2009) how it is possible to synthesize test cases that are guaranteed to take the least possible time to execute. We also have defined a language for defining test purposes and coverage criteria.

A solution to address a model with full expressiveness is to use online testing. The latter indeed enables working with non-deterministic specifications. In online (on-the-fly) testing, which combines test generation and execution, the test generator interactively interprets the model, and stimulates and observes the SUT. Only a single test input is generated from the model at a time that is then immediately executed on the SUT. Then the produced outputs (active and suspended if any) by the SUT as well as their occurrence times are checked against the specification, a new input is produced and so forth until it is decided to end the test, or an error is detected. Typically, the inputs and delays are chosen randomly. There are several advantages of online testing (Hessel, 2008). Testing may potentially continue for a long time (hours or even days), and consequently long, intricate test cases that stress the SUT may be executed. The state-space-explosion problem experienced by many offline test generation tools is reduced since only a subset of the state-space needs to be stored at any point in time. Further, online test generators often allow more expressive specification languages, especially non-determinism in real-time models: Since they are generated event-by-event they are automatically adaptive to the non-determinism of the specification i.e. the specification is determinized implicitly on the fly. A disadvantage is that the test runs are typically long, complex and consequently the cause of a test failure may be difficult to diagnose. Although some guidance is possible, test generation is typically randomized which means that satisfaction of coverage criteria cannot be a priory guaranteed, but must instead be evaluated post mortem. In Section 8 we present an online testing algorithm allowing full non-determinism specifications.

5. TEST SPECIFICATION

An embedded system interacts closely with its environment. A major development task is to ensure that the system works correctly in this environment. So, testing involves a system surrounded by an environment. An uncontrolled and possibly imaginary environment would indeed allow all possible interaction sequences. But, due to lack of resources it is not feasible to validate the system for all possible (imaginary) environments. Also it is not necessary if the environments are known to a large extent. Practically, each system operates in specific environments called its real operating environment, and it is only necessary to establish its correctness under the modelled environment assumptions. However, the requirements and the assumptions of the environment should be clear and explicit. Therefore, we make a distinction between the specified system and its environment. Modeling the environment explicitly and separately from the system and taking this into account during test generation has several advantages: (1) we can synthesize only relevant and realistic scenarios for the given type of environment, which in turn reduces the number of required tests and improves the quality of the test suite; (2) the engineer can guide the test generator to specific situations of interest; (3) a separate environment model avoids explicit changes to the system model if testing must be done under different assumptions or use patterns. Otherwise, it is possible to create a fully open environment for the SUT i.e. a completely unconstrained one that allows all possible interaction sequences. Consequently, the 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.

We denote the system being developed SUT and its real operating environment RealENV (See Figure 1). The SUT and its environment communicate by exchanging input and output signals (seen from the SUT). When the SUT is being tested, the tester plays the role of the environment. Using a model-based development approach testing, the environment assumptions and system requirements are captured through abstract behavioral models, communicating on abstract signals. We assume that the test specification, noted $M = M_{SUT} || M_{En}$, is given as an LPrSwTPN made up of two concurrent subnets. $M_{SUT}$ models the expected behavior of the SUT while $M_{En}$ models the behavior of the environment (See Figure 2). We need to distinguish inputs and outputs between the SUT and the environment, which are the only observable events when we consider the SUT as a black box. The set of all observable actions is then partitioned in input and output actions noted respectively $A_{in}$ and $A_{out}$. An observable action can be interrupted at any time and resumed after. We must output to the outside the suspension information by an indicator output. The outputs are not controllable by the system and should be tested also with their delivery dates. An input (a standard output) is post fixed by $?$ (!) and an indicator output has the same label as the observable action. We assume that $A$ is equipped with a mapping $\overline{a} : A \rightarrow A$ such that for all actions $\overline{a} = a$ and $\overline{a}$ is the complementary action of $a$ such that $\overline{a}! = a? \land a?! = a!$.

The system may perform internal events. 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. Internal actions are not observed by the environment and thus to the tester. They are denoted $\tau (\tau \notin A)$.

6. ENVIRONMENT AND SYSTEM MODELING

6.1 Labelled Prioritized Time Petri Nets with Stopwatches

Time Petri Nets (TPN) (Merlin, 1976) extend Petri Nets with temporal intervals on transitions to model time constraints. Prioritized Time Petri Nets (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 fireable at the same instant. Such priorities increase the expressive power of TPN. TPN with Stopwatches (SwTPN) extend (Pr)TPN by stopwatch arcs that control the

Figure 1. The SUT and its real environment RealENV

Figure 2. The test specification $M$ composed of The SUT model $M_{SUT}$ and its environment model $M_{En}$
progress of transitions to express suspension and resumption of actions (Berthomieu, 2007). A label that may be empty is associated to each transition (LPrSwTPN) to denote an observable action or an internal operation of the system.

6.1.1 Notations for LPrSwTPN

The sets \( \mathbb{N}, \mathbb{Q}, \mathbb{Q}_{\geq 0}, \mathbb{R}, \mathbb{R}_{\geq 0} \) are respectively the sets of natural, rational, non-negative rational, real and non-negative real numbers. We consider the set \( \mathbb{I}^+ \) of non-empty real intervals \([a, b] \) with bounds \( a, b \in \mathbb{R}_{\geq 0} \). We consider both open and closed bounds, and also allow a right open infinite bound as in \( \left(1, \infty\right] \).

For any \( \theta \in \mathbb{R}_{\geq 0} \), \( \downarrow \theta \) represents the interval \( \{x \in \mathbb{R} : x \leq \theta\} \), and \( \uparrow \theta \) represents the interval \( \{x \in \mathbb{R} : x \geq \theta\} \).

For any \( \theta \in \mathbb{R}_{\geq 0} \), \( \downarrow \theta \) represents the interval \( \{x \in \mathbb{R} : x \leq \theta\} \), and \( \uparrow \theta \) represents the interval \( \{x \in \mathbb{R} : x \geq \theta\} \).

3. \( \prec \subseteq T \times T \) is the priority relation, assumed irreflexive, asymmetric and transitive, between transitions. \( t_1 \prec t_2 \) means \( t_2 \) has priority over \( t_1 \).

4. \( \Lambda : T \rightarrow A_\tau \) is the labelling function that associates to each transition an action. The internal \( \tau \)-action is indicated by an absent action-label.

5. \( \text{Sw} : T \rightarrow \mathbb{N} \) is the stopwatch incidence function. \( \text{Sw} \) 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 3 shows an LPrSwTPN. 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 5 time units. If we replace the stopwatch arc by a normal arc, \( t_2 \) will never be fired.

The transitions of the net \( M = M_{\text{SUT}} \parallel M_{\text{En}} \) are partitioned into purely transitions of the SUT model \( M_{\text{SUT}} \) (invisible for the environment \( M_{\text{En}} \), normally labelled with \( \tau \) and indicated by an absent label action) and synchronizing transitions between the \( M_{\text{SUT}} \) and the \( M_{\text{En}} \) models (observable for both parties). We note \( T_{\text{SUT}} \) the set of the SUT model transitions and \( T_{\text{En}} \) the set...
of the environment model transitions. The set of transitions labeled with internal actions is
\[ T = \{ t \in T_{\text{SWJ}} / \Lambda (t) = \tau \}. \]
They are fired individually. A couple \((t, t') \in (T_{\text{SWJ}} \times T_{\text{En}}) \) is a synchronizing transition if they are labeled with complementary actions \( a, \overline{a} \) respectively (e.g. \( \Lambda (t) = a \) (resp. \( a' \)) and \( \Lambda (t') = \overline{a} \) (resp. \( a \))). We assume that the first component is an action of the SUT model \( M_{\text{SUT}} \) while the second is of the environment model \( M_{\text{En}} \). The synchronizing transitions are fired by complementary actions couples (e.g. \( a \) and \( a' \)). The set of the environment model transitions that complement a synchronizing transition \( t \in T_{\text{SWJ}} \) is
\[ CT_{\text{SWJ}} (t) = \{ t' \in T_{\text{En}} / \Lambda (t') = a \text{ and } \Lambda (t) = \overline{a} \}. \]

To illustrate the concepts, we use the coffee machine model depicted by Figure 4. It shows an LPrSwTPN specifying the requirements to a coffee machine. The SUT model accepts a coin and a request for coffee (inputs). Depending on when the request for coffee is issued, weak (light) or strong coffee (outputs) is produced. 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. Note the non-determinism when the request is issued in the time interval \([30, 50]\). The choice is non-deterministic, meaning that the SUT/implementor may decide what to produce. After the request, the machine takes 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 5 time units. This service is not allowed for the user requesting light coffee. The machine makes internal actions to be reset or to choose between preparing light or strong coffee in the non-deterministic case.

The LPrSwTPN shown in Figure 4 can be composed in parallel with the environment models \( M_{\text{Eni}} \) shown in Figure 5, 6 and 7 respectively. We obtain three composed models
\[ M_i = M_{\text{SUT}} \mid M_{\text{Eni}} \quad (i = 1, 2, 3) \] over
\[ A_{\text{in}} = \{ \text{coin, req, takeCup, returnCup} \} \] and
\[ A_{\text{out}} = \{ \text{strongCoffee, lightCoffee} \}. \] Figure 5 models potential users of the machine that pay before requesting coffee and take their coffee after its preparation. In Figure 7, 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.
Figure 4. $M_{\text{SUT}}$: a specification LPrSwTPN of the SUT coffee machine

Figure 5. An environment model $M_{\text{En1}}$

Figure 6. n other environment $M_{\text{En2}}$
6.1.3 Semantic of LPrSwTPN

6.1.3.1 Timed Transition Systems
Timed Transition Systems (TTS) describe systems that combine discrete and continuous evolutions. Here, they are used to define the semantics of the parallel composition of LPrSwTPN.

A TTS over a finite set of actions $A_S$ is a transition system $E = (E, e_0, A_S, →)$ where $E$ is the, possibly infinite, set of states of the system, $e_0$ is the initial state, $A_S$ is the set of actions composed of internal action $τ$ and couples of synchronizing actions $ AS τ$. The transition relation $→ ⊆ E \times × (\tau) \times E$ consists of (1) discrete transitions $e → e'$ or $e → e'$ representing instantaneous actions $((a, \vec{a}) \in A_s$ and $\lambda \in A'$) and (2) continuous (delay) transitions $e → e'$ representing the passage of $d$ units of time. Moreover, we require the following standard properties for TTS: (1) Time-determinism: if $e → e'$ and $e → e''$ then $e' = e''$, (2) 0-delay: $e → e$, (3) Additivity: if $e → d → e'$ and $e' → e''$, $(d, d' ∈ \mathbb{R}_{≥0})$ then $e → d + d' → e''$, (4) Continuity: if $e → d → e'$ then $e → d → e''$ such that $e → d → e''$.  

6.1.3.2 Notations for TTS
Let $α, α_{0...n} ∈ A_S$, $β, β_{0...n} ∈ \{τ\} \cup \mathbb{R}_{≥0}$ $ω, ω_{0...n} ∈ (A_s × A') \cup \{τ\}$ $λ, λ_{0...n} ∈ A', d, d_{0...n} ∈ \mathbb{R}_{≥0}$. We write $e → α → λ → e'$ iff $e → α → λ → e'$ and $e → β → e'$ for some $e'$. The transition relation $→$ is the relation where internal actions were abstracted ($→ ∈ \big((A_s × A') \cup \{τ\}\big)$). We write $e → α → λ → e'$ iff $e → τ → * → α → λ → τ → * → e'$ and $e → d → e'$ iff $e → τ → * → d_0 → τ → * → d_1 → τ → * → d_n → τ → * → e'$ ($d = d_0 + d_1 + ... + d_n$). We write $e → α → λ →$
iff $e \xrightarrow{d} e'$ and $e \xrightarrow{a} e'$ for some $e'$. We extend $e \xrightarrow{a} e'$ to sequences in the usual manner $e \xrightarrow{\sigma} e'$ iff $e = e_0, e_n = e' \land \ e_{i-1} \xrightarrow{\alpha \lambda} e_i$ or $e_{i+1} \xrightarrow{d} e_i$ where 

$\sigma = d_0 \alpha_0 \lambda_0 \ldots d_n \alpha_n \lambda_n$.

A SUT (resp. an environment) model is strongly input enabled iff $\prod_{e \in E_{in}} e \xrightarrow{a^?}$ (resp. $\prod_{e \in E_{Env}} e \xrightarrow{a^?}$) for all states $e$ and for all input (resp. output) actions $a$? . It is weakly input enabled iff $\prod_{e \in E_{in}} e \xrightarrow{a^?}$ for all states $e$ and for all input actions $a$? . We assume that input actions (seen from the system point of view) are controlled by the environment and outputs are controlled by the system. An input enabled system cannot refuse input actions. However it may decide to ignore the input by executing a synchronizing transition that results in the same state. A SUT model is non-blocking iff for any state $e$ and any $d \in \mathbb{R}_{\geq 0}$, there is a timed trace $\sigma = d_0 \alpha_0 \lambda_0 \ldots \alpha_k \lambda_k d_{k+1}$ where $\alpha_i = (a_i, \bar{a}_i)$ and $a_i \in A_{out}$ (all the first components of the SUT are outputs), such that $e \xrightarrow{\sigma}$ and $\sum_i d_i \geq d$. Thus the SUT will not block time in any input enabled environment. This property ensures that a system won’t force or rush its environment to deliver an input, and vice versa, the environment will never force outputs from the system.

An observable timed trace is the timed word $\sigma \in \left( \left( A_{s} \times A' \right) \cup \mathbb{R}_{\geq 0} \right)^*$ which is of the form $\sigma = d_0 \alpha_0 \lambda_0 \ldots \alpha_k \lambda_k d_{k+1}$ where $\alpha_i$ is a couple of synchronizing actions and $\lambda_i$ is an eventual set, may be empty, of suspended actions which may appear after the firing of $\alpha_i$. We define the observable timed traces of a state $e$ as:

$\text{TTr}(e) = \left\{ \sigma \in \left( \left( A_{s} \times A' \right) \cup \mathbb{R}_{\geq 0} \right)^* \mid e \xrightarrow{\sigma} \right\}$. For a state $e$ (and a subset $E' \subseteq E$) and a timed trace $\sigma$, $\text{After}(e, \sigma)$ is the set of states which can be reached after $\sigma$. $\text{After}(e, \sigma) = \left\{ e' \mid e \xrightarrow{\sigma} e' \right\}$, $\text{After}(E', \sigma) = \bigcup_{e \in E'} \text{After}(e, \sigma)$. The set $\text{Out}(e)$ of observable active and suspended outputs or delays from states $e \in E' \subseteq E$ is defined as:

$\text{Out}(e) = \left\{ (a, \lambda) \in A_{out} \times A' \mid \exists \pi \in A_{in} : e \xrightarrow{(a, \lambda) \pi} \right\} \cup \left\{ d \in \mathbb{R}_{\geq 0} \mid e \xrightarrow{d} \right\}$

$\text{Out}(E') = \bigcup_{e \in E'} \text{Out}(e)$.

6.1.3.3 States of an LPrSwTPN

A state of an LPrSwTPN is a pair $e = (m, I)$, where $m$ is a marking of the net. A marking is a function $m : P \rightarrow \mathbb{N}^+$ with $m(p)$ the number of tokens in place $p$. A transition $t$ is enabled at marking $m$ iff $m \geq \text{Pre}(t)$. We denote by $\text{En}(m)$ the set of transitions enabled at $m$. It is then equal to $\text{En}(m) = \left\{ t \in T \mid m \geq \text{Pre}(t) \right\}$. In addition, an enabled transition $t$ at $m$ is active iff $m \geq \text{Sw}(t)$, otherwise it is said “suspended”. The set of active (resp. suspended) transitions at $m$ is denoted by $\text{Ac}(m) = \left\{ t \mid t \in \text{En}(m) \land m \geq \text{Sw}(t) \right\}$ (resp. $\text{Su}(m) = \left\{ t \mid t \in \text{En}(m) \land m < \text{Sw}(t) \right\}$). $\mathcal{M}(e)$ is the marking of the state $e$. $I$ is a partial function called the interval function. It associates exactly a temporal interval in $I^+$ with every enabled transition $\text{I} : \text{En}(m) \rightarrow I^+$. $\text{I}(t)$ represents the firing interval of the enabled transition $t$. Intuitively, if $t \in \text{Ac}(\mathcal{M}(e))$, $\text{I}(t)$ is shifted towards the origin as time elapses, and truncated to no negative times while a suspended transition $t' \in \text{Su}(\mathcal{M}(e))$ has its temporal interval $\text{I}(t')$ unchanged. Assuming that the amount of time that has elapsed since
is enabled for the last one is \( \theta \) then
\[
I(t) = I_s(t) - \theta \quad \text{if } t \text{ is an active transition. An}
\]
enabled transition \( t \) is fireable if (1) it is active, (2) it is immediately fireable (0 \( \in I(t) \)) and, (3)
no other transition with higher priority is fireable at the same instant, (4) if \( t \) is synchronizing transition then its complementary transition is also fireable. After the firing, some transitions are associated with their intervals \( I_s(t) \) and we say that they are newly enabled. The initial state is \( e_0 = \{m_0, I_0\} \)
where \( I_0 = [I_s(En(m_0)] \cup I_s(En(m_0]) \) (the interval function \( I_0 \) is \( I_s \) restricted to the en-
abled transitions \( En(m_0) \)).

The initial state of the LPrSwTPN \( M_1=\{M_{\text{init}} \} \)
\( \parallel M_{\text{init}} \) is \( e_0 = \{p_0, q_0\} \) (where \( p_0, q_0 \) are both marked with one token),
\( En(m) = \{t_0, k_0\} \), \( Ac(M(e_0)) = \{t_0, k_0\} \) and \( Su(M(e_0)) = \phi \). The transitions \( (t_0, k_0) \)
labeled respectively by (\( \text{coin?}, \text{coin!} \)) can be fired respectively on \( [0, \infty[ \).

The temporal information in states will be seen as firing domains instead of interval functions. The firing domain of a state \( e = (m, I) \)
is then described by an equations linear system with one variable per enabled transition (noted as transitions). The state will be then noted \( e = (m, D) \)
where
\[
D = \{\phi \mid (\forall t \in En(m)) (\phi_t \in I(t))\}.
\]
The state \( e_0 = \{m_0, D_0\} \) of \( M_1 \) is
\[
e_0 = \{m_0, \{\phi \mid 0 \leq \phi_m, 0 \leq \phi_{k_0}\}\}.
\]

6.1.3.4 Newly Enabled Transition
For \( m \in \mathbb{N}^+, l \in T_{\text{sw}} \) and \( t \in T_r \) such that
\( t \in En(m) \) and \( t \in Ac(m) \) we define a
predicate \( ne_e(l, m, t) \) which is true if \( l \) is newly enabled by the firing of \( t \) from \( m \), and
false otherwise. Formally, the predicate is defined by:
\[
ne_e(l, m, t) = \begin{cases} 
  l \in En(m) - \{l\} + Post(t) \land & \text{For} \\
  l \notin En(m) - \{l\} \lor l = t & \\
  m \in \mathbb{N}^+, k \in \mathbb{T}, t \in (T_{\text{sw}} - T_r) & \text{and} \\
  t' \in CT_{\text{sw}}(t) & \text{such that} \\
  t, t' \in En(m) & \text{and} \\
  t, t' \in Ac(m) \text{the predicate } ne_e(k, m, (t, t')) & \text{which is true if } k \text{ is newly enabled by the firing of } t \text{ and } t' \text{ simultaneously from } m, \\
  \text{false otherwise by:} \\
  ne_e(k, m, (t, t')) = \begin{cases} 
 k \in En(m) - \{l\} - \{l\} + Post(t) + Post(t') \land & \\
 k \notin En(m) - \{l\} \lor k = t' & \\
 \end{cases}
\end{cases}
\]
The predicate \( ne_e(k, m, (t, t')) \) (resp. \( ne_e(k, m, l) \)) indicates the necessity to associate to \( k \) its static interval after firing simultaneously the couple \( (t, t') \) (resp. individually the transition \( l \)) at the marking \( m \). Intuitively, it associates to the couple \( (t, t') \) (resp. \( l \)) and to the transitions that could not be fired in parallel with \( (t, t') \) (resp. \( l \)) their static intervals.

6.1.3.5 The Semantics of an LPrSwTPN
The semantics of an LPrSwTPN \( \mathcal{N} = \{P, T, Pre, Post, Sw, \leq, m_0, I, A\} \) is a TTS
\( [\mathcal{N}] = (E, e_0, A_{s'}, \rightarrow) \) where \( E \) is the set of states \( (m, I) \) of \( \mathcal{N} \), \( e_0 \) its initial state and
\( \rightarrow \subseteq E \times (\{A_s \times A'\}) \{r\} \cup \mathbb{R}_{\geq 0} \times E \) consists of two kinds of transitions between states: discrete and continuous transitions.

6.1.3.6 Transitions Firing Algorithms
The continuous transition relation is the result of time elapsing. It is defined by
\[
e = (m, I) \rightarrow d = e' = (m, I') \text{ iff}
\]
1. \( d \in \mathbb{R}_{\geq 0} \)
2. \( (\forall t \in T) \{t \in En(m) \land t \in Ac(m) \Rightarrow d \leq \uparrow I(t)\} \)
transition $t$ from state $e$ and is noted $\prod_{SUT} e \rightarrow e'$. (4) is the standard marking transformation. (5) In the target state, the transitions which are newly enabled are associated with their static intervals.

- The synchronizing transition relation is defined by
  
  $$e = (m, I) \xrightarrow{\tau} e' = (m', I')$$

1. $\exists t \in T_{SUT} - T_{\tau} (\Lambda(t) = a \land t \in En(m) \land t \in Ac(m))$
2. $0 \in I(t)$
3. $\forall k \in T_{SUT} (k \in En(m) \land k \in Ac(m) \land k \leq k' \Rightarrow 0 \notin I(k'))$
4. $\exists t' \in T_{\tau} (\Lambda(t') = a \land t' \in En(m))$
5. $0 \in I(t')$
6. $\forall k' \in T_{\tau} (k' \in En(m') \land t' < k' \Rightarrow 0 \notin I(k'))$
7. $m' = m - \delta(t' + Pre(t) + Post(t) + Post(t'))$
8. $\forall k \in T_{\tau} (k \in En(m') \Rightarrow 1'(k') = 1(k)$

The synchronizing transitions $(t, t') \in T_{SUT} \times T_{\tau}$ labeled respectively $(a, a)$ may fire simultaneously from the state $e$ if they are enabled and active (1 and 4), immediately fireable (2 and 5) and neither a transition of $M_{SUT}$ nor a transition of $M_{\tau}$ with higher priorities compared to $t$ and $t'$ respectively satisfies these conditions (3 and 6). The conjunction $(1) \land (2) \land (3)$ (resp. $(4) \land (5) \land (6)$) is called the fireability predicate of the synchronizing transition $t$ of the SUT model (resp. $t'$ of the environment model) from the state $e$ and it is noted $\prod_{SUT} e \rightarrow \tau$ (resp. $\prod_{\tau} e \rightarrow \pi$). (8) In the target state, the transitions that remained enabled while $t, t'$ fired ($t, t'$ being excluded) retain their intervals,
the others which are newly enabled are associated with their static intervals.

With the properties of TTS, a run $\rho$ of $[N]$ can be defined as a finite sequence of moves

$$e_0 \xrightarrow{d} e_0' \xrightarrow{\omega} e_1 \xrightarrow{d} e_1' \xrightarrow{\omega} e_2 \cdots e_n \xrightarrow{\omega} e_{n+1}$$

where discrete and continuous transitions (possibly of duration 0) alternate. The discrete transitions are either synchronizing transitions followed by the eventual suspended actions ($\omega = \alpha_1, \lambda_1$ where $\alpha_1 = \{a, \bar{a}\} \in A_s$ and $\lambda_1 \in A^*$ are suspended transitions) or pure transitions ($\omega = \tau$), and $d_{i,0 \leq i \leq n}$ are their relative firing times.

From the initial state $e_0 = (m_0, D_0)$ of the composed model $M_1$ Figure 4 and 5, if the coffee machine receives a coin at 0.85$t.u.$ then we have a discrete transition $(t_0, k_0)$ preceded by a temporal transition leading to $e_1 = \{p_0,q_0,\{0 \leq t_0, 0 \leq k_0\}\}$. The firing of $(t_0, k_0)$ labeled with (coin?, coin!) from $e_1$ leads to $e_2 = (m_1, D_2)$ where $m_1 = p_1 q_1$; $D_2 = \{0 \leq t_0 \leq 10, 0 \leq k_0 \leq \infty \}$. Observe that $e_1$ is not fired while $e_2$ is true because the predicate $\{\text{t.k}\}$ is false and there is no complementary transition $\{\text{t.k}\}$ is not enabled). Their firing after 20$t.u.$ (e.g. the brewing of a light coffee takes 20$t.u.$) from the state $e_4$ leads to $e_5$ passing by $e_5 = (m_2, D_5)$

$$e_4 \xrightarrow{20} e_5 \xrightarrow{\text{lightCoffee}, \text{lightCoffee}, \phi} e_0$$

where

$$\begin{align*}
0 &\leq t_4 \leq 10 \\
0 &\leq k_4 \leq \infty \\
0 &\leq k_3 \leq \infty
\end{align*}$$

From $e_4$ only the synchronizing transitions $(t_9, k_3)$ can be fired while $k_2$ can’t be fired ($\prod_{\text{Env}} e_4 \xrightarrow{k_2}$ is not true because the complementary transition $\Lambda(k_2)$ is not enabled). Their firing after 20$t.u.$ (e.g. the brewing of a light coffee takes 20$t.u.$) from the state $e_5$ leads to $e_6$ passing by $e_6 = (m_2, D_6)$

$$e_6 \xrightarrow{20} e_7 \xrightarrow{\text{lightCoffee}, \text{lightCoffee}, \phi} e_0$$

where

$$\begin{align*}
0 &\leq t_6 \leq 10 \\
0 &\leq k_6 \leq \infty \\
0 &\leq k_3 \leq \infty
\end{align*}$$

The transitions $t_4$ and $k_3$ can’t fire ($\prod_{\text{Env}} e_4 \xrightarrow{k_3}$ and $\prod_{\text{Env}} e_7 \xrightarrow{k_3}$) are false (the complementary transition $\Lambda(k_3)$ is not enabled and there is no complementary transition of $t_4$). The firing of $(t_4, k_2)$ after
(e.g. the brewing of a strong coffee takes $35\text{ t.u.}$) from $e_7$ leads to $e_0$ passing by $e_9 = (m_4 = p_2p_3q_0, D_9)$

\[e_7 \xrightarrow{35} e_8 \] \[e_8 \xrightarrow{[\text{strongCoffee!, strongcoffee?}]} e_9 \xrightarrow{r} e_0\]

where $D_8 : \begin{cases} 0 \leq t_4 \leq \infty \\ 0 \leq t_8 \leq 15 \\ 0 \leq k_2 \leq \infty \\ 0 \leq k_3 \leq \infty \end{cases}$

and $D_9 : \begin{cases} 0 \leq t_{10} \leq 0 \\ 0 \leq k_0 \leq \infty \end{cases}$

Consider now the model $M_1$ composed of $M_{\text{SU1}}$ and $M_{\text{En3}}$ (Figure 4 and 7). Starting from the initial state $e_0$, the timed trace $\sigma = (\text{coin?}, \phi) 55 (\text{req?, req!}, \phi)$ is performed by $M_2$. We have the sequence of transitions

\[e_0 \xrightarrow{0.85} e_1 \xrightarrow{[\text{coin!, coin?}]} e_2 \xrightarrow{55} e_3 \xrightarrow{[\text{req!, req?}]} e_4\]

and then $\sigma \in \text{Tr} (e_0)$; After $(e_0', \sigma) = \{e_4'\}$ where $e_4' = (m_2', D_4')$ and $m_2': p_2p_3q_5$ and

\[D_4' : \begin{cases} 0 \leq t_4 \leq \infty \\ 0 \leq t_8 \leq 15 \\ 0 \leq k_6 \leq \infty \\ 0 \leq k_{10} \leq \infty \end{cases} \]

As above, the user can only be served a strong coffee because he has requesting for a coffee at $50\text{ t.u.}$ Suppose that he likes to taste the coffee during its preparation and return the cup to have the remain of his coffee. He can takes the cup and returns it not after than $5\text{ t.u.}$ as specified in Figure 4, suppose that he takes it at $35\text{ t.u.}$ and return it at $4\text{ t.u.}$ We have the following run

\[e_4' \xrightarrow{35} e_5' \xrightarrow{[\text{tackeCup?,tackeCup!}]} e_6' \xrightarrow{[\text{strongCoffee!, strongcoffee?}]} e_7' \xrightarrow{r} e_8' \xrightarrow{10} e_9' \xrightarrow{[\text{strongcoffee?}, \phi]} e_0' \]

and:

2. $\mathcal{A}(\mathcal{M}(e_0')) = \{t_5, t_3, k_9; \text{labeled respectively by returnCup?, returnCup!, strongcoffee?} \}$

3. $\mathcal{S}(\mathcal{M}(e_0')) = \{t_8 \mid (t_8) = \text{strongCoffee!} \}$

4. $\mathcal{O}(e_0') = \{\{\phi, \text{strongCoffee!}\} \cup \{d \in \mathbb{R}_+ \mid d \leq 5\} \}$(a suspended output $\text{strongCoffee!}$ and a delay $d$).

6.1.4 Non-Determinism and Time

For many real-time systems the ordering of time events cannot be known a priori, and hence a deterministic model cannot appropriately capture its behavior. Non-determinism plays a particular role because it is used to express timing and ordering uncertainty. A typical real-time requirement is that the SUT must deliver an output within a given time bound, but as long as the deadline is satisfied, the SUT conforms. In TTS, this is specified as a nondeterministic choice between letting time pass and producing an output. In LPRTWTPN this is described by associating a temporal interval with the transition producing the output. Non-determinism is also used in specification as a means of abstraction. It may be that the implementation internally exhibits non-determinism that cannot be observed or controlled by the tester. A further typical use of non-determinism is to model optional behavior that is permitted, but not required by all implementations.

A non-deterministic model may reach several possible states after having executed an action, and as a consequence it may have different possible next behaviors. This possible set of states represents the uncertainty the tester has about the exact state of a (conforming) SUT, and the tester must be prepared to accept any legal next behavior according to the state set. Non-deterministic timed specifications are algorithmically and computationally more
complex to analyze because they require symbolic manipulation of sets of infinite sets of states. The required reachability algorithms for online testing are similar to those used for model analysis except that only states up to a certain time limit need to be computed. Due to non-determinism it is necessary to represent the state-set as a set \( Q \) of symbolic states.

The specification of Figure 4 is non-deterministic in two ways. First, the coffee machine switches state within interval delays, but it is unknown when. Thus from e.g., state \( e_7 \) the controller of the coffee machine may execute any of the observable traces, \( d.\{\text{strongCoffee!}, \text{strongcoffee?}\}(30 \leq d \leq 50) \).

Note that \( \text{Out}(e_7) = \{\{\text{strongCoffee!}, \phi\}\} \cup \{d \mid d \in [30,50]\} \)

(An active output \( \text{strongCoffee!} \), no suspended output and a delay \( d \)). Second, there are several next states to a request for a coffee if it is issued in the interval \( [30,50] \) e.g. \( \{e_2, 40(\text{req?}, \text{req!})\} \) the machine may brew in non-deterministic way light or strong coffees.

6.1.5 Symbolic State-Set Computation

Because of temporal non-determinism (dense time), a state may admit an infinity of successor states, which implies that the state space of an LPrSwTPN may be infinite. Finitely representing state spaces involves grouping some particular sets of states into symbolic states. For TPN, state space abstractions are available that preserve markings and all traces, and states and traces (Berthomieu, 2008). Unfortunately, the first abstraction termed SC for classical State Class is too coarse to preserve the effects of priorities (Berthomieu, 2007) We investigate in this paper the extensions of the second abstraction termed SSC (for Strong State Classes), also called state zones by some authors, to LPrSwTPN. For the construction of the SSC, clock domains serve this purpose where the principal is as follows. With each reachable state, one may associate a clock function \( \gamma \). The later associates with each enabled transition at the state the time elapsed since it was last enabled. Clock functions may also be seen as vectors \( \gamma \) indexed over the enabled transitions ( \( \gamma_t \) is the time elapsed since \( t \) was last enabled).

The initial state of the LPrSwTPN \( M_{\text{En1}} \) is \( s_0 = \left( p_0, q_0, \gamma, \rightarrow = 0, \gamma, \rightarrow = 0 \right) \).

Informally, the system leaves the initial state \( s_0 = \left( m_0, \gamma, \rightarrow = 0 \right) \) by making alternately two types of transitions: discrete transitions if the current value allows it and time transitions that increase the clock values of the active transitions by the same duration. The time in the suspended transitions is frozen. So, when a frozen transition becomes active again, due to a change in markings, it resumes with the clock domain captured in the state rather than the value 0. The new transition relation \( \sim \) is also decomposed to:

1. The continuous transition defined by
   \[
   s = \left( m, \gamma \right) \rightarrow s' = \left( m, \gamma' \right) \quad \text{iff}
   \]
   a. \( d \in \mathbb{R}_{\geq 0} \)
   b. \( \left( \forall t \in T \right) t \in \text{En}(m) \lor t \in \text{Ac}(m) \Rightarrow d \leq \frac{1}{I_s(t) - \gamma_t} \)
   c. \( \left( \forall t \in T \right) t \in \text{En}(m) \Rightarrow \gamma' = \gamma_t + d \quad \text{else} \quad \gamma' \)

2. The internal independent transition relation defined by
   \[
   s = \left( m, \gamma \right) \rightarrow s' = \left( m', \gamma' \right) \quad \text{iff}
   \]
   a. \( \left( \exists t \in T \right) t \in \text{En}(m) \land t \in \text{Ac}(m) \)
   b. \( \gamma' \in I_s(t) \)
   c. \( \left( \forall k \in T_{\text{SUT}} \right) \left( k \in \text{En}(m) \land k \in \text{Ac}(m) \Rightarrow \gamma'\right. \)
   d. \( m' = m - \text{Pre}(t) + \text{Post}(t) \)
   e. \( \left( \forall k \in T_{\text{SUT}} \right) \left( k \in \text{En}(m') \Rightarrow \gamma'\right. \)
   f. \( \left( \text{if } ne_t(k,m,t) \text{ then } 0 \text{ else } \gamma'\right) \)
3. The synchronizing transition relation defined by $(m, \gamma) \leadsto (m', \gamma')$ iff
a. $(\exists t \in T_{suk} - T)(\Lambda(t) = a \land t \in En(m) \land t \in Ac(m))$
b. $\gamma_t \in I_1(t)$
c. $(\forall k \in T_{suk})(k \in En(m) \land k \in Ac(m))$

3. The synchronizing transition relation defined by $(m, \gamma) \leadsto (m', \gamma')$ iff
a. $(\exists t \in T_{suk} - T)(\Lambda(t) = a \land t \in En(m) \land t \in Ac(m))$
b. $\gamma_t \in I_1(t)$
c. $(\forall k \in T_{suk})(k \in En(m) \land k \in Ac(m))$

d. $(\exists t' \in T_{a})(\Lambda(t') = t' \in En(m) \land t' \in Ac(m))$
e. $\gamma_{t'} \in I_1(t')$
f. $(\forall k' \in T_{a})(k' \in En(m) \land t' \in t' \in k' \Rightarrow \gamma_{k'} \notin I_1(k'))$
g. $m' = m - (\text{Pre}(t) + \text{Pre}(t')) + \text{Post}(t) + \text{Post}(t')$
h. $\text{ne}_{a,\pi}(k, m, (t, t'))$ then 0 else $\gamma_k$

From the initial state in the model $M_1$, if the coffee machine receives an input request from the user at 0.85 t.u., then we have a transition $s_0 \leadsto s_1$

$s_1 = \left\{ p_0q_0, \gamma_0 = 0.85, \gamma_o = 0.85 \right\}$. The firing of the synchronizing transitions $(k_0, k_0)$ from $s_1$ leads to

$s_2 = \left\{ p_0q_1, \gamma_0 = 0, \gamma_1 = 0, \gamma_0 = 0, \gamma_1 = 0 \right\}$

As we have shown above the state space of an LPrSwTPN may be infinite. Therefore, we use the SSC abstraction. A class or a symbolic state is of the form $(m, Q)$: a marking and a clock system $Q = \{ G_{\gamma} \leq g \}$. The set of states denoted by $(m, Q = \{ G_{\gamma} \leq g \})$ is the set $\{ m, \phi(\gamma) \mid \gamma \in \langle Q \rangle \}$, where $\langle Q \rangle$ is the solution set of $Q$ and firing domain $\phi(\gamma)$ is the solution set in $\phi$ of:

$0 \leq \phi$ and $\phi \leq \phi + \gamma \leq \theta$

where $\phi_k \leq \Pi_k(\gamma)$ and $\Pi_k(\gamma)$. The initial symbolic state $(m_0, Q_0)$ is obtained from the state $(m_0, Q)$ where

$Q_0 = \{ G_{\gamma} \leq g \mid m_0, 0 \leq \gamma, 0 \leq \gamma \} \text{ and } m_0, \gamma' \leq \gamma \leq (t \in En(m_0))$

$Q_0 = \{ 0 \leq \gamma, 0 \leq \gamma \leq t \in En(m_0) \}$. The solution set $\langle Q_0 \rangle$ assigns all clocks of enabled transitions $En(m_0)$ to zero.

A symbolic computation step consists of performing a synchronizing or an internal action, noted respectively by $(m, Q) \leadsto (m', Q')$ and $(m, Q) \leadsto (m', Q')$, followed by some delay. It is performed iff

$(m, Q) \leadsto (m', Q')$ or $(m, Q) \leadsto (m', Q')$ and $Q' = \left\{ G_{\gamma} \leq g \mid (m, \gamma) \leadsto (m', \gamma') \right\}$

or $Q' = \left\{ G_{\gamma} \leq g \mid (m, \gamma) \leadsto (m', \gamma') \right\}$.

Symbolic transition relation between classes $\leadsto$ are:

1. The internal symbolic transition $(m, Q) \leadsto (m', Q')$ iff

a. $(\exists t \in T_{suk})(t \in T_\gamma \land t \in En(m) \land t \in Ac(m))$

b. $\downarrow I_k(t) - \gamma \leq 0$
ii. \( \{ I_s(k) \mid \gamma_s \leq 0 \} \ \forall k \in \text{T}_{\text{sw}} \land k \in \text{En}(m) \land k \in \text{Ac}(m) \land t < k \) is consistent

c. \( m' = m - \text{Pre}(t) + \text{Post}(t) \)

d. \( Q'' \) obtained by
   i. The constraints (b) above, and
   ii. \( \forall \in (m') \left( \gamma''_s = \text{if } ne(x,k,m,t) \text{ then } 0 \text{ else } \gamma'_s \right) \)
   iii. The variables \( \gamma'' \) are eliminated

e. \( Q'' \) augmented with
   i. \( d \geq 0 \)
   ii. \( \{ d \leq I_s(k) - \gamma''_s \mid k \in \text{En}(m') \land k \in \text{Ac}(m') \} \) is consistent

f. \( Q' \) obtained by the constraints (5) above, and
   i. \( \forall \in (m') \left( \gamma'_s = \text{if } k \in \text{Ac}(m') \text{ then } \gamma''_s + \right) \)
   ii. The variables \( \gamma'' \) and \( d \) are eliminated

2. The synchronizing symbolic transition
   \( (m,Q) \rightarrow (m',Q') \) iff
   a. \( (\exists t \in \text{T}_{\text{sw}}) \left( t \in T - \text{T}_{\alpha} \land \Lambda(t) = a \land t \in \text{En}(m) \land t \in \text{Ac}(m) \right) \)
   b. \( Q \) augmented with
      i. \( \downarrow I_s(t) - \gamma' \leq 0 \)
      ii. \( \downarrow I_s(k) - \gamma_s > 0 \)
      k \in \text{T}_{\text{sw}} \land k \in \text{En}(m) \land k \in \text{Ac}(m) \land t < k \)

c. \( \exists t' \in \text{T}_{\alpha} \left( \Lambda(t') = \bar{a} \land t' \in \text{En}(m) \land t' \in \text{Ac}(m) \right) \)
   Q augmented with
   i. \( \downarrow I_s(t') - \gamma' \leq 0 \)
   ii. \( \downarrow I_s(k') - \gamma_{k'} > 0 \)
   k' \in \text{T}_{\alpha} \land k' \in \text{En}(m) \land t' < k' \) is consistent

e. \( m' = m - \left( \text{Pre}(t) + \text{Pre}(t') \right) + \text{Post}(t) + \text{Post}(t') \)
   \( \epsilon Q'' \) obtained by (a) The constraints (2) and (4) above, and
   i. \( \forall \in (m') \left( \gamma''_s = \text{if } ne(x,k,m,t') \text{ then } 0 \text{ else } \gamma'_s \right) \)
   ii. The variables \( \gamma'' \) are eliminated

Q'' augmented with
   i. \( d \geq 0 \)
   ii. \( \{ d \leq I_s(k) - \gamma''_s \mid k \in \text{En}(m') \land k \in \text{Ac}(m') \} \) is consistent

h. \( Q' \) obtained by
   i. The constraints (g) above, and
   ii. \( \forall \in (m') \left( \gamma'_s = \text{if } k \in \text{Ac}(m') \text{ then } \gamma''_s + \right) \)
   \( \epsilon \downarrow I_s \left( k \in \text{Su}(m') \right) \)
   iii. The variables \( \gamma'' \) and \( d \) are eliminated

7. THE RSWTIOCO CONFORMANCE RELATION AND TEST HYPOTHESIS

A conformance relation formalizes the set of SUT that behave correctly compared to a reference specification. In this paper, we require Relativized Stopwatch Timed Input/Output Conformance relation (rswtico). This relation supports reactivity, timing, suspension/resumption principal characteristics of real time systems and takes environment assumptions into account. So, it is indexed by the name of the considered environment (rswtico_{En}). Our notion derives from the rtioco relation (Hessel, 2008). The latter is itself an extension of tioco (Krichen, 2009) which in turn is an extension ioco (De vries, 2000 ; Tretmens 1999).

Under assumptions of weak input enabledness rswtico_{En} coincides with relativized timed trace inclusion i.e. Timed Traces of the SUT operating under an environment En must be
included in those of the specification under the cover of the same environment. Like \textit{rtioco}, this relation ensures that the SUT has only the behavior allowed by the specification with the difference that outputs here may be both standard outputs or delays and indicator outputs (suspended actions). In particular, 1) the SUT is not allowed to produce an output at a time when one is not allowed by the specification, 2) it is not allowed to omit producing an output when one is required by the specification (the \textit{SUT} may delay only if the specification also may delay). Unlike \textit{tioco} (Krichen, 2009) \textit{rswtioco} distinguishes between the system’s constraints and the environment’s ones. Due to this separation, testing can be limited to certain parts of the SUT (model). So, the question “does the SUT conform to its specification?” is answered not for any type of possible environment but for the considered one. A “yes” answer to the previous question obtained for one environment still applies to more restrictive environments. A relativized 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 or resumption of an action \textit{a} we have to consider the (part of the) environment that drives the syntactical parts of the SUT that satisfies this objective, the input to supply to the SUT, and also when to supply it, that enable to suspend or resume the action \textit{a} (see Algorithm 1).

The \textit{rswtioco} relation does not only allow outputs to be emitted in advance or on late by the SUT but also 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 \textit{rswtioco} relation makes it possible to answer another question: “does some action \textit{a} resume at the expected date?"

A SUT is not a formal object. However, formally proving its conformity requires modeling its semantics by a formal object. We assumes it can be modeled by an unknown \textit{LPrSwTPN} denoted $M_{\text{HI}}$ (hypothesis implementation model). For the SUT to be testable the \textit{LPrSwTPN} of its specification 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 weak-input enabled and non-blocking \textit{LPrSwTPN}, and that the SUT can be modelled by some unknown weak-input enabled and non-blocking \textit{LPrSWTPN}. We allow for the SUT to be reset to its initial state.

Given an environment $E_n$ expressed by the \textit{LPrTPN} model $M_{E_n}$, a SUT $I$ expressed by an unknown \textit{LPrSwTPN} model $M_{\text{HI}}$ and a specification $S$ of the SUT expressed by the \textit{LPrSwTPN} model $M_{\text{SUT}}$. Let $M$ be the \textit{LPrSwTPN} of $S$ together with its intended environment ($M=M_{\text{SUT}} ||M_{E_n}$) with the initial state $s_0$ and $M'$ be the \textit{LPrSwTPN} of the SUT together with the same environment ($M'=M_{\text{HI}} ||M_{E_n}$) with the initial state $e_0$. Let $M$ and $M'$ be two weak-input enabled and non-blocking \textit{LPrSwTPN}. The \textit{En}-relativized conformance relation \textit{rswtioco}$_{En}$ between systems $I$ and $S$ is defined as:

$$I \textit{rswtioco} _{E_n} S \iff \forall \sigma \in TTr(M) : \text{Out}(\text{After}(e_0, \sigma)) \subseteq \text{Out}(\text{After}(s_0, \sigma))$$

(1)

Whenever $I \textit{rswtioco} _{En} S$ we say that $I$ is a correct implementation of the specification $S$ under the environment constraints expressed by $E$. Given the notion of relativized conformance, it is natural to consider the preorder on environments based on their discriminating power. For environments $E$ and $E'$:

$$E \sqsubseteq E' \iff \textit{rswtioco} _{En} \subseteq \textit{rswtioco} _{En'}$$ (To be read $E'$ is more discriminating than $E$).

It follows from the definition of $\textit{rswtioco} _{En}$ that $E \sqsubseteq E'$ iff the behaviour of $E$ is included in the behaviour of $E'$. There is a most
(least) discriminating input enabled and non-blocking environment $E_{nu}$ ($E_{no}$) (See Figure 10 (a) and 10 (b)) given by

$$TT_{R} (E_{nu}) = ( \cup_{0} R_{nu} ) , TT_{R} (E_{ne}) = ( \cup_{0} R_{ne} ) .$$

The corresponding conformance relation $rs_{w}ti_{co} E_{nu} (rs_{w}ti_{co} E_{no})$ specializes to simple trace inclusion (timed output trace inclusion).

Moreover, because we treat environment constraints explicitly and separately, $rs_{w}ti_{co} E_{nu}$ has some nice theoretical and practical attractive properties that allows the tester to re-use testing effort if either the environment assumption is strengthened, or if the system specification is weakened. Assume that $J rs_{w}ti_{co} E_{nu}$, then without re-testing

if $S \subseteq S'$ then $J rs_{w}ti_{co} E_{nu}$ $S'$

if $E_{nu} \subseteq E_{nu}$ then $J rs_{w}ti_{co} E_{nu}$ $S$

In the following we exemplify how our conformance relation discriminates systems, and illustrate the potential power of environment assumptions and how this can help to increase the relevance of the generated tests for a given environment.

The implementation $I_{l}(I_{l}, I_{s})$ in Figure 9 produces light coffee (strong coffee) after less than 40 t.u. (more than 41 t.u.) and an additional brewing time in $\downarrow \uparrow (I_{l}, I_{s}) (\downarrow I_{s}, \uparrow I_{s})$. Notice
that $\mathcal{F}_1\left([\downarrow U, \uparrow U], [\downarrow I_s, \uparrow I_s]\right)$ does not allow the user requesting light coffee to take his cup before the brewing time $I_l$. Observe that any trace of the SUT $I_1([20, 25],[40, 45])$ in $E_{nu}$ can be matched by the specification; hence $I_1([20, 25],[40, 45]) \ rswtioco_{Enu} S$. Thus also $I_1([20, 25],[40, 45]) \ rswtioco_{Enu} S$ ($i=1,2,3,4$) by (3). In contrast, $I_1([2, 5],[60, 70]) \ rswtioco_{Enu} S$ for two reasons: 1) it has the timed trace: 10. $(\text{coin}?, \text{coin}!)\varnothing.30.(\text{req}?, \text{req}!)\varnothing.3.$ (lightCoffee!,lightcoffee?)\varnothing that $S$ does not, i.e. it may produce light coffee too soon (no time to insert a cup); 2) it has a trace: 15. $(\text{coin}?, \text{coin}!)\varnothing.50.(\text{req}?, \text{req}!)\varnothing.65.$ (strongCoffee!,strongcoffee?)\varnothing not in $S$ meaning that it produces strong coffee too slowly. The SUT can thus perform standard outputs at a time not allowed by the specification. Assume now that the strong coffee error is fixed, and that the machine $I_2([2, 5],[40, 45])$ is used in the restricted environment $E_{n2}$; here despite the remaining light coffee error in $E_{nu}$, $I_2([2, 5],[40, 45]) \ rswtioco_{En2} S$ because $E_{n2}$ never requests light coffee. The SUT outputs a suspended output lightCoffee! at a time not allowed by the specification. If the transitions $k_6$ and $k_{13}$ in the environments $E_{n3}$ and $E_{n4}$ labelled with takeCup! are associated with the interval $[50, 60]$ then we have $I_1([20, 25],[45, 45]) \ rswtioco_{En1} S$ for $i=3,4$. In these cases, the transitions $k_6$ and $k_{13}$ will never be fired and thus the transitions strongcoffee! and lightcoffee! will never be suspended. Assume now that the strong coffee error is fixed, and that the machine $I_2([2, 5],[40, 45])$ is used in the restricted environment $E_{n2}$; here despite the remaining light coffee error in $E_{nu}$, $I_2([2, 5],[40, 45]) \ rswtioco_{En2} S$ and thus also $I_2([2, 5],[40, 45]) \ rswtioco_{En1} S$ because $E_{n2}$ never takes the cup during the preparation of the coffee while $E_{nu}$ never requests light coffee.

8. TEST GENERATION AND EXECUTION

The main idea of the algorithm is the following. Start where $Q$ contains the initial symbolic state $(m_0, Q_0)$. Then continually compute the set of states $Q$ that both the environment composed to the specification can be in after the observed test run so far. This is done until either $Q$ is empty (no legal states) or improper suspended output(s) is (are) issued and a fail verdict is assigned, or it has reached the defined number of iterations $N$ and a passed verdict is assigned. The randomized online testing algorithm is then a state estimator; it occupies a set of symbolic states $Q$ and modifies it after every test event. The set $Q$ is updated each time an
input is offered or an output or a delay is observed. This information allows us to choose the proper test primitive and validate the outputs of the SUT. The algorithm randomly performs one of three basic actions: (1) Sends an input \( a \) (enabled environment output) randomly chosen among legal inputs according to \( \mathcal{E} \) to the SUT, then updates \( \mathcal{E} \) according to \( \text{After} \left( \mathcal{E}, \left(a?,a!\right) \right) \). (2) Waits for an output \( o \) after a delay \( \delta \). Randomly chooses how long it will wait. If an active output \( \text{active}(o) \) and/or suspended output(s) \( \text{suspend}(o) \) or a delay \( \delta' \leq \delta \) are observed, the tester verifies if these are conforming to the specification. If it observes an active output and/or suspended outputs before the chosen time has passed, it updates the state set \( \mathcal{E} \) according to how long it has waited (\( \text{After} \left( \mathcal{E}, \delta' \right) \)). It checks also if the outputs are legal ones according to \( \mathcal{E} \). If they are legal outputs, \( \mathcal{E} \) is updated according to \( \text{After} \left( \mathcal{E}, \left(\text{active}(o),\text{active}(o)\right) \right) \); else a fail verdict is assigned. If no output is observed during sleeping the set \( \mathcal{E} \) is updated according to the passing of time (\( \text{After} \left( \mathcal{E}, \delta \right) \)). (3) Resets the SUT and restarts.

The functions used in Algorithm 1 are defined as:

\[
\text{EnvOutput} \left( \mathcal{E} \right) = \left\{ a! \in A_{in} \left| E \left( t \right) \right. \left( t \in T_{ex} \land \exists e \in \mathcal{E} \right) \rightarrow \left( \Lambda \left( t \right) = a! \land \prod_{e \in \mathcal{E}} e - a! \right) \right\}
\]

\[
\text{Delays} \left( \mathcal{E} \right) = \left\{ d \leq \delta \left| E \left( t \right) \left( t \in T_{ex} \land \exists e \in \mathcal{E} \right) \rightarrow \left( \Lambda \left( t \right) = a! \land e - a! \rightarrow e' \land \prod_{e' \in \mathcal{E}} e' - a! \right) \right\}
\]

\[
\text{ImpOutput} \left( \mathcal{E} \right) = \left\{ a! \in A_{out} \left| E \left( t \right) \left( t \in T_{SUT} \land \exists e \in \mathcal{E} \right) \rightarrow \left( \Lambda \left( t \right) = a! \land \prod_{e \in \mathcal{E}} e - a! \right) \right\}
\]

\[
\text{ImpSuspend} \left( \mathcal{E} \right) = \left\{ a \in A \left| E \left( t \right) \left( t \in T_{SUT} \land \exists e \in \mathcal{E} \right) \rightarrow \left( \Lambda \left( t \right) = a \land t \in S \left( \mathcal{E} \right) \right) \right\}
\]

\( \text{EnvOutput} \left( \mathcal{E} \right) \) is the set of input actions (enabled environment outputs) that are allowed by the environment in the current symbolic state set \( \mathcal{E} \), and is empty if the environment model has no outputs to offer. \( \text{RandomlyChooseAction} \) selects randomly an input from \( \text{EnvOutput} \left( \mathcal{E} \right) \) applicable to the SUT. If the environment must offer an input to the SUT before a certain moment in time, delays cannot be randomly chosen; in this case, \( \text{Delays} \) must
pick a real number from the interval that fulfils those constraint. \( \text{ImpOutput} (\mathcal{E}) \) is the set of active output actions that are allowed according to SUT specification in the current state set. \( \text{ImpSuspend} (\mathcal{E}) \) is the set of actions that are suspended according to SUT specification in the current state set. An output \( o \) is a pair (active output, suspended actions) denoted respectively by the functions \( \text{active} (o) \) and \( \text{suspend} (o) \). Any illegal occurrence or absence of a standard output or a delay from the SUT is detected if the set \( \mathcal{E} \) update by the function \( \text{After} \) leads to an empty set, which happens when the observed trace is not allowed by the specification. The presence (resp. absence) of improper (resp. proper) suspended outputs from

Algorithm 1. \( \text{GenExeTest} (S, \mathcal{E}, \text{SUT}, N) \) // Generation and execution of test.

Initially \( \mathcal{E} := \{ (m_0, Q_0) \} \)

while \( \mathcal{E} \neq \emptyset \wedge \text{iterations} \leq N \) do RandomlyChoose(\text{Action, Delay, Restart})
   case Action: // offer an input to the SUT
      If \( \text{EnvOutput} (\mathcal{E}) \neq \emptyset \) then
         \( a! := \text{RandomlyChooseAction} (\text{EnvOutput} (\mathcal{E})) \)
         send \( a! \) to the SUT
         \( \mathcal{E} := \text{After} (\mathcal{E}, (a!,a!)) \)
   case Delay: // wait for an output of the SUT. Sleep for \( \delta \) and wake up on output \( o \)
      \( (o \text{ contains eventually suspended actions)} \) sent by the SUT.
      \( \delta := \text{RandomlyChooseDelay} (\text{Delays} (\mathcal{E})) \)
      if \( o \text{ occurs at } \delta' \leq \delta \) then do
         \( \mathcal{E} := \text{After} (\mathcal{E}, \delta') \)
         \( a! := \text{active} (o) \)
         if \( \text{suspend} (o) \not\subseteq \text{ImpSuspend} (\mathcal{E}) \) then
            do return fail
            For each
               \( b! \in \text{ImpOutput} (\mathcal{E}) \cap \text{Suspend} \ (o) \)
            then "\( b! \) must be a an active output"
            done
         if \( a! \not\in \text{ImpOutput} (\mathcal{E}) \) then do return fail
         if \( a! \in \text{ImpSuspend} (\mathcal{E}) \) then "a!
         must be a suspended action"
         done
         else \( \mathcal{E} := \text{After} (\mathcal{E}, \delta) \)
      done
   case Restart: \( \mathcal{E} := \{ (m_0, Q_0) \} \) and Reset SUT // reset and restart
      If \( \mathcal{E} = \emptyset \) then return fail else return pass
Algorithm 2. After $(\mathcal{E}, (a, \overline{a}))$
Passed := $\emptyset$, Waiting := $\text{Closure}_{r}(\mathcal{E})$

For each symbolic state $(m, Q) \in \text{Waiting}$

For each symbolic transition $(m, Q) \xrightarrow{(a, \overline{a})_{\mathcal{S}, \delta}} (m', Q')$ if

$\neg \text{contains}(\text{Passed}, (m', Q'))$

Then $\text{Passed} := \text{Passed} \cup \{(m', Q')\}$

return $\text{Closure}_{r}(\text{Passed})$

Algorithm 3. After $(\mathcal{E}, \delta)$

After $(\mathcal{E}, \delta) = \{(m, Q') | (m, Q) \in \text{Closure}_{\mathcal{E}}(\mathcal{E}, \delta)\}$

SUT is detected if the outputs $\text{suspend}(o)$ are not (resp. are) in $\text{ImpSuspend}(\mathcal{E})$.

The function $\text{After}(\mathcal{E}, \alpha)$ (resp. $\text{After}(\mathcal{E}, \delta)$) computes the set of states $\mathcal{E}'$ reachable after the action $\alpha \in A_s$ (resp. time delay $\delta$). $\text{After}$ computes a closure of states reachable after performing all potential internal transitions and one observable synchronizing action or delay. It returns an empty set if the action or delay was not allowed by the specification. Different strategies can be applied to guide the test generation to interesting or uncovered states by changing the model of environment, “choose” functions and adopting them to a particular test purposes.

Algorithm 4 computes the function $\text{Closure}_{\mathcal{E}}(\mathcal{E}, d)$ that collects the reachable symbolic state set within a delay of $d$. The

Algorithm 4. $\text{Closure}_{\mathcal{E}}(\mathcal{E}, d)$
Passed := $\emptyset$, Waiting := $\mathcal{E}$

while Waiting $\neq \emptyset$ do

Waiting := Waiting $\setminus \{(m, Q)\}$ // pick a symbolic state

if $(m, Q) \xrightarrow{d'} (m, Q')$ where $d' \leq d$ then

Passed := Passed $\cup \{(m, Q')\}$

For each symbolic transition $(m, Q') \xrightarrow{\tau} (m', Q'')$ if

$\neg \text{contains}(\text{Passed}, (m', Q''))$

then

Waiting := Waiting $\cup \{(m', Q'')\}$

return Passed
Predicate \( \text{Contains}(\mathcal{E}, (m, Q)) \) tests whether a symbolic state \( (m, Q) \) is covered by a symbolic state in \( \mathcal{E} \). \( \langle Q \rangle \) is the set of solutions of the temporal variables associated with the enabled transitions. Closure \( \mathcal{E} \) is \( \text{Closure}_{\text{er}}(\mathcal{E}, 0) \). It collects the reachable symbolic state set after all possible internal transition in zero delay and.

9. CONCLUSION AND FUTURE WORK

In this paper, we have studied algorithms for testing real time systems applicable to LPrSwTPN models. The latter have been selected for their capacities to model suspend/resume operations in real-time systems (whereas surveyed papers on timed testing only address system/environment interactions and timeliness). First, we have reviewed the efficient algorithms derived from ordinary analysis algorithm, similar to those used in model-checking and analyzing tools and extended them to support the parallel composition of LPrSwTPN. Second, we have introduced a formal real-time correctness relation \( \text{rswtioco} \). It differs from \( \text{tioco} \) because it addresses the constraints captured by the system separately from the ones inherent to the environment. Also, \( \text{rswtioco} \) differs from both \( \text{tioco} \) and \( \text{rtioco} \) 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. Finally, we have proposed an online testing technique that makes possible to handle non determinism and partly observable systems and ensures thoroughness through volume and brute-force. The approach proposed in the paper will be soon implemented in the TINA model analyzer. We plan to address other types of testing in the near future (in particular, robustness testing).

REFERENCES


**Noureddine Adjir is an assistant Professor at the Computer Science Department, University of Saida, Algeria. He received the magister in Computer science from University of Sénia, Oran, Algeria. His research interests include Real-time systems modeling, formal methods. He has a two decades experience in teaching modeling techniques and formal methods.**
Pierre de Saqui-Sannes received the Ph.D. and “Habilitation à Diriger les Recherches” in computer science from Université Paul Sabatier and Institut National Polytechnique de Toulouse, respectively. He is now a full professor at the department of Mathematics, Computer Science and Control Theory at ISAE (Institut Supérieur de l’Aéronautique et de l’Espace) in Toulouse, France. His research interests include Real-time systems modeling, bridging the gap between SysML and formal methods, and protocol engineering. He has a two decades experience in teaching modeling techniques (formal methods, real-time UML, SysML) to industry practitioners.

M.K Rahmouni is a Professor at the Computer Science Department, University of Oran-Sénia, Oran (Algeria). He holds a PhD from Southampton University (1987) and his interests include Computer Security, Modelization of Information Systems, and Real-Time Systems modeling.