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;

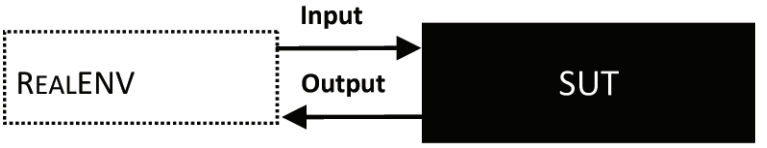
although (1) TA cannot be determinized in general (Alur,1994), and (2) that it is sometimes impossible to withdraw internal actions (Diekert, 1997). The result is that some works only address a subclass of TA. Given a restricted class of deterministic and output urgent TPN, we have showed in (Adjir, 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

Figure 1. The SUT and its real environment RealENV



"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 deliverance 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 $\bar{\ } : A \to A$ such that for all actions $\bar{\bar{a}} = a$ and $\bar{a}$ is the complementary action of a such that $\bar{a}! = a? \wedge \bar{a}? = a!$.

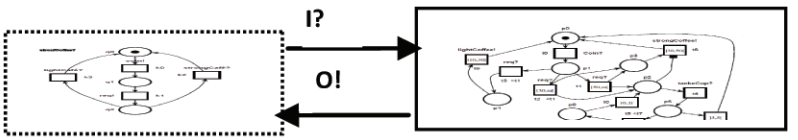
The system may perform internal events. They may result from an abstraction of low level details made to facilitate the modelling or to allow a certain freedom to the implementor or more to events which we do not want that the tester to observe them to facilitate its task. Internal actions are not observed by the environment and thus to the tester. They are denoted $\tau \left(\tau \notin A\right)$. A_τ abbreviates $A_{in} \cup A_{out} \cup \left\{\tau\right\}$.

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 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 I^+ of non-empty real intervals $[a, b]$ with bounds $a, b \in \mathbb{Q}_{\geq 0}$. We consider both open and closed bounds, and also allow a right open infinite bound as in $[1, \infty[$. For $i \in \mathrm{I}^+$, $\downarrow i$ represents its lower bound, and $\uparrow i$ its superior bound (if it exists) or ∞. For any $\theta \in \mathbb{R}_{\geq 0}$, $i \dot{-} \theta$ represents the interval $\{x - \theta \,/\, x \geq \theta \wedge \theta \geq 0\}$. $A_S = A_{in} \times A_{out} \cup A_{out} \times A_{in}$ is the set of the couples of synchronizing actions and $A_{S\tau} = A_S \cup \{\tau\}$ is the set of internal and synchronizing actions.

6.1.2 Syntax of LPrSwTPN

Formally, a LPrSwTPN over the alphabet is a tuple $N = (\mathsf{P}, \mathsf{T}, \mathsf{Pre}, \mathsf{Post}, Sw, \prec, m_0, \mathrm{I}_s, \Lambda)$ where:

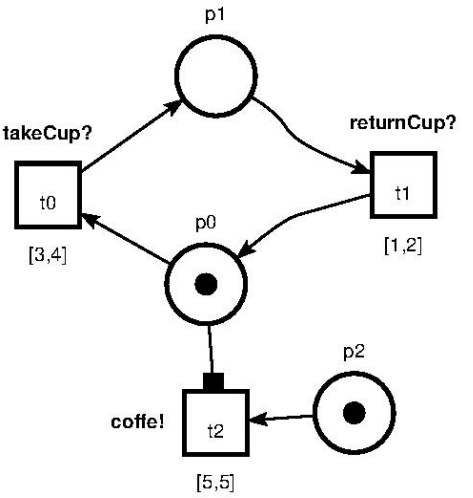
1. $(\mathsf{P}, \mathsf{T}, \mathsf{Pre}, \mathsf{Post}, m_0)$ is a Petri Net where P is a finite set of places, T is a finite set of transitions with $\mathsf{P} \cap \mathsf{T} = \phi$, $m_0 : \mathsf{P} \to \mathbb{N}^+$ is the initial marking and $\mathsf{Pre}, \mathsf{Post} : \mathsf{T} \to \mathsf{P} \to \mathbb{N}$ are respectively the precondition and post-condition functions. For $f, g \in \mathsf{P} \to \mathbb{N}^+$, $f \geq g$ means that $(\forall p \in P)(f(p) \geq g(p))$ and $f \{+,-\} g$ is $f(p) \{+,-\} g(p)$ for any p.

2. $\mathrm{I}_s : \mathsf{T} \to \mathrm{I}^+$ is the static interval function. It associates a firing temporal interval I_s with each transition. The rational $\downarrow \mathrm{I}_s(t)$ (resp. $\uparrow \mathrm{I}_s(t)$) is the static earliest (resp. latest) firing time of t after the latter was enabled. Assuming that a transition t became enabled at the last one at the time θ, then t can't be fired before $\theta + \downarrow \mathrm{I}_s(t)$ and it must be done no later than $\theta + \uparrow \mathrm{I}_s(t)$, unless disabled by firing some other transition. In this paper, intervals $[0, \infty[$ are omitted and **w** in the right end point of an interval denotes ∞.
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. $\Lambda : \mathsf{T} \to A_\tau$ is the labelling function that associates to each transition an action. The internal τ-action is indicated by an absent action-label.
5. $\boldsymbol{Sw} : \mathsf{T} \to \mathsf{P} \to \mathbb{N}$ is the stopwatch incidence function. ***Sw*** associates an integer with each $(p, t) \in \mathsf{P} \times \mathsf{T}$, 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 p0 to transition t2 is a stopwatch arc of weight 1. The firing of t0 will freeze the timing evolution of t2. t2 will be fireable when its total enabling time reaches 5 time units. If we replace the stopwatch arc by a normal arc, t2 will never be fired.

The transitions of the net $M = M_{SUT} \,||\, M_{En}$ are partitioned into purely transitions of the SUT model M_{SUT} (invisible for the environment M_{En}, normally labelled with τ and indicated by an absent label action) and synchronizing transitions between the M_{SUT} and the M_{En} models (observable for both parties). We note $\mathsf{T}_{\mathcal{SUT}}$ the set of the SUT model transitions and $\mathsf{T}_{\mathcal{En}}$ the set

Figure 3. PrSwTPN example



of the environment model transitions. The set of transitions labeled with internal actions is $\mathsf{T}_{\tau} = \left\{ t \in \mathsf{T}_{SUT} / \Lambda(t) = \tau \right\}$. They are fired individually. A couple $\left(t, t'\right) \in \left(\mathsf{T}_{SUT} - \mathsf{T}_{\tau}\right) \times \mathsf{T}_{En}$ is a synchronizing transitions if they are labeled with complementary actions $a, \overline{a}$ respectively *e.g.* $\Lambda(t) = a?$ (resp. $a!$) and $\Lambda(t') = a!$ (resp. $a?$). We assume that the first component is an action of the SUT model M_{SUT} while the second is of the environment model M_{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 $t \in \mathsf{T}_{SUT}$ is $\mathrm{CT}_{SUT}(t) = \left\{ t' \in \mathsf{T}_{En} / \Lambda(t) = a \text{ and } \Lambda(t') = \overline{a} \right\}$.

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_{Eni} shown in Figure 5, 6 and 7 respectively. We obtain three composed models $M_i = M_{SUT} \,||\, M_{Eni}$ $(i = 1,2,3)$ over $A_{in} = \{\mathsf{coin}, \mathsf{req}, \mathsf{tackeCup}, \mathsf{returnCup}\}$ and $A_{out} = \{\mathsf{strongCoffee}, \mathsf{lightCoffee}\}$. 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_{SUT}: a specification LPrSwTPN of the SUT coffee machine

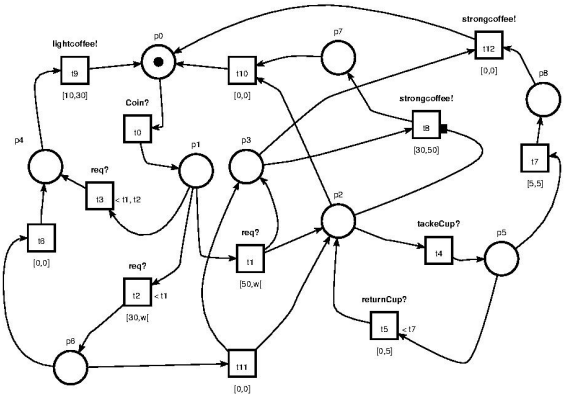


Figure 5. An environment model M_{En1}

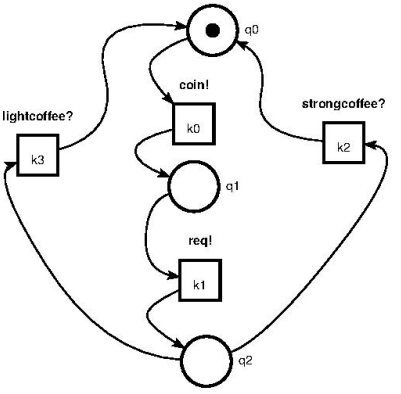


Figure 6. n other environment M_{En2}

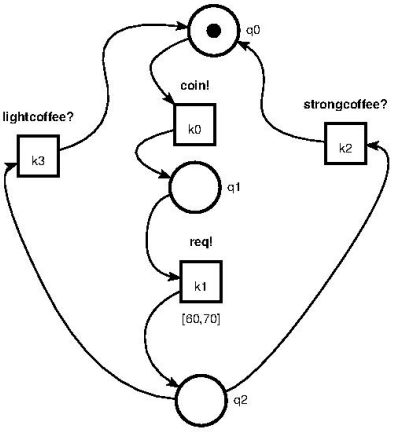
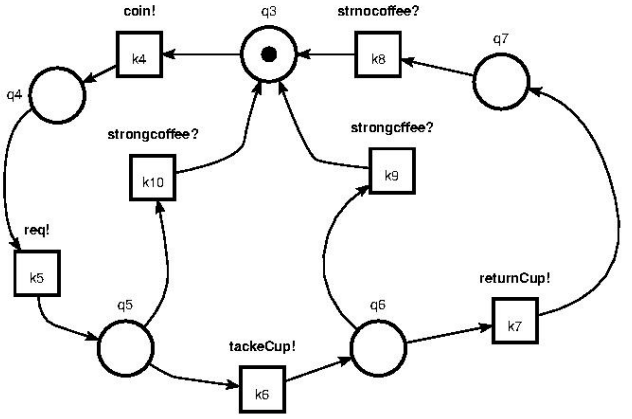


Figure 7. An other environment model M_{En3}



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 $A_{S\tau}$ is a transition system $\mathrm{E} = \left(E, e_0, A_{S\tau}, \rightarrow\right)$ where E is the, possibly infinite, set of states of the system, e_0 is the initial state, $A_{S\tau}$ is the set of actions composed of internal action τ and couples of synchronizing actions A_S. The transition relation $\rightarrow \subseteq E \times \left(\left(A_S \times A^*\right) \cup \left\{\tau\right\} \cup \mathbb{R}_{\geq 0}\right) \times E$ consists of (1) discrete transitions $e \xrightarrow{(a,\overline{a}),\lambda} e'$ or $e \xrightarrow{\tau} e'$ representing instantaneous actions ($\left(a,\overline{a}\right) \in A_S$ and $\lambda \in A^*$ is the set of eventual suspended actions that may arise by the firing of $\left(a,\overline{a}\right)$) and (2) continuous (delay) transitions $e \xrightarrow{d} e'$ representing the passage of d units of time. Moreover, we require the following standard properties for TTS: (1) *Time-determinism*: if $e \xrightarrow{d} e'$ and $e \xrightarrow{d} e''$ then $e' = e''$, (2) 0-*delay*: $e \xrightarrow{0} e$, (3) *Additivity*: if $e \xrightarrow{d} e'$ and $e' \xrightarrow{d'} e''$, $\left(d, d' \in \mathbb{R}_{\geq 0}\right)$ then $e \xrightarrow{d+d'} e''$, (4) *Continuity*: if $e \xrightarrow{d} e'$ then $\left(\forall d', d'' \in \mathbb{R}_{\geq 0} : d = d' + d''\right)\left(\exists e''\right)$ such that $e \xrightarrow{d'} e'' \xrightarrow{d''} e'$.

6.1.3.2 Notations for TTS

Let.. $\alpha,\ \alpha_{0\ldots n} \in A_S$,

$\beta,\ \beta_{0\ldots n} \in \left\{\tau\right\} \cup \mathbb{R}_{\geq 0}$

$\omega,\ \omega_{0\ldots n} \in \left(A_S \times A^*\right) \cup \left\{\tau\right\}$

$\lambda,\ \lambda_{0\ldots n} \in A^*,\ d, d_{0\ldots n} \in \mathbb{R}_{\geq O}$.

We write $\mathrm{e} \xrightarrow{\alpha,\lambda}$ iff $e \xrightarrow{\alpha,\lambda} e'$ and $\mathrm{e} \xrightarrow{\beta}$ iff $e \xrightarrow{\beta} e'$ for some e'. The transition relation $\longrightarrow\!\!\triangleright$ is the relation $\rightarrow$ where internal actions were abstracted ($\longrightarrow\!\!\triangleright \in \left(\left(A_S \times A^*\right) \cup \mathbb{R}_{\geq 0}\right)^*$). We write $\mathrm{e} \xrightarrow{\alpha,\lambda}\!\!\triangleright e'$

iff $e \xrightarrow{\tau} * \xrightarrow{\alpha,\lambda} \xrightarrow{\tau} * \ e'$

and $\mathrm{e} \xrightarrow{d}\!\!\triangleright e'$

iff $e \xrightarrow{\tau} * \xrightarrow{d_0} \xrightarrow{\tau} * \xrightarrow{d_1} \xrightarrow{\tau} * \cdots \xrightarrow{d_n} \xrightarrow{\tau} * \ e'$ ($d = d_0 + d_1 + \cdots + d_n$). We write $\mathrm{e} \xrightarrow{\alpha,\lambda}\!\!\triangleright$

iff $e \xrightarrow{\alpha,\lambda} e'$ and $e \xrightarrow{d}$ iff $e \xrightarrow{d} e'$ for some e'. We extend $\longrightarrow$ to sequences in the usual manner $e \xrightarrow{\sigma} e'$ iff $e = e_0$, $e_n = e' \wedge e_{i-1} \xrightarrow{\alpha_i,\lambda_i} e_i$ or $e_{i-1} \xrightarrow{d_i} e_i$ where $\sigma = d_0\alpha_0\lambda_0\, d_1\alpha_1\lambda_1 \ldots d_n\alpha_n\lambda_n$.

A SUT (resp. an environment) model is *strongly input enabled* iff $\prod_{SUT} e \xrightarrow{a?}$ (resp. $\prod_{Env} e \xrightarrow{a?}$) for all states e and for all input (resp. output) actions $a?$. It is *weakly input enabled* iff $\prod_{SUT} e \xrightarrow{a?}$ 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 $\sigma = d_0\alpha_0\lambda_0 \ldots \alpha_n\lambda_n d_{n+1}$ where $\alpha_i = (a_i, \overline{a}_i)$ 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 α_i is a couple of synchronizing actions and λ_i is an eventual set, may be empty, of suspended actions which may appear after the firing of α_i. We define the observable timed traces of a state e as:

$$\mathbf{TTr}(e) = \left\{ \sigma \in \left(\left(A_S \times A^*\right) \cup \mathbb{R}_{\geq 0}\right)^* \mid e \xrightarrow{\sigma} \right\}.$$

For a state e (and a subset $E' \subseteq E$) and a timed trace σ, $After(e,\sigma)$ is the set of states which can be reached after σ. $After(e,\sigma) = \left\{ e' \mid e \xrightarrow{\sigma} e' \right\}$, $After(E',\sigma) = \bigcup_{e \in E'} After(e,\sigma)$. The set $\mathbf{Out}(e)$ of observable active and suspended outputs or delays from states $e \in E' \subseteq E$ is defined as:

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

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

6.1.3.3 States of an LPrSwTPN

A state of an LPrSwTPN is a pair $e = (m, \mathrm{I})$, where m is a marking of the net. A *marking* is a function $m : \mathsf{P} \to \mathbb{N}^+$ with $m(p)$ the number of tokens in place p. A transition t is enabled at marking m iff $m \geq \mathsf{Pre}(t)$. We denote by $En(m)$ the set of transitions enabled at m. It is then equal to $En(m) = \left\{ t \in \mathsf{T} \,/\, m \geq \mathsf{Pre}(t) \right\}$. In addition, an enabled transition t at m is *active* iff $m \geq \boldsymbol{Sw}(t)$, otherwise it is said "*suspnded*". The set of active (resp. suspended) transitions at *m* is denoted by

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

(resp. $Su(m) = \left\{ t \mid t \in En(m) \wedge m < Sw(t) \right\}$). $\mathcal{M}(e)$ 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 ($\mathrm{I} : En(m) \to \mathrm{I}^+$). $\mathrm{I}(t)$ represents the firing interval of the enabled transition *t*. Intuitively, if $t \in Ac(\mathcal{M}(e))$, $\mathrm{I}(t)$ is shifted towards the origin as time elapses, and truncated to no negative times while a suspended transition $t' \in Su(\mathcal{M}(e))$ has its temporal interval $\mathrm{I}(t')$ unchanged. Assuming that the amount of time that has elapsed since

t is enabled for the last one is θ then $\mathrm{I}(t) = \mathrm{I}_s(t) \dot{-} \theta$ if t is an active transition. An enabled transition t is fireable if (1) it is active, (2) it is immediately fireable ($0 \in \mathrm{I}(t)$) and, (3) no other transition with higher priority is fireable at the same instant, (4) if t is sychronizing transition then its complementary transition is also fireable. After the firing, some transitions are associated with their intervals $\mathrm{I}_s(t)$ and we say that they are newly enabled. The initial state is $e_0 = (m_0, \mathrm{I}_0)$

where $\mathrm{I}_0 = \left[\downarrow \mathrm{I}_s\left[En(m_0)\right], \uparrow \mathrm{I}_s\left[En(m_0)\right]\right]$ (the interval function I_0 is I_s restricted to the enabled transitions $En(m_0)$).

The initial state of the LPrSwTPN $\mathrm{M}_1 = \mathrm{M}_{SUT} \| \mathrm{M}_{En1}$ is $e_0 = \left(p_0 q_0, \left\{(t_0, [0, \infty[), (k_0, [0, \infty[)\right\}\right)$ where (places p_0, q_0 are both marked with one token),

$En(m_0) = \{t_0, k_0\}$, $Ac(\mathcal{M}(e_0)) = \{t_0, k_0\}$ and $Su(\mathcal{M}(e_0)) = \phi$. The transitions (t_0, k_0) labeled respectively by (coin?, 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 $e = (m, \mathrm{I})$ 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 $D = \left\{\underline{\phi} \mid \left(\forall t \in En(m)\right)\left(\underline{\phi}_t \in \mathrm{I}(t)\right)\right\}$. The state $e_0 = (m_0, D_0)$ of M_1 is $e_0 = \left(m_0, \left\{\underline{\phi} \mid 0 \le \underline{\phi}_{t_0}, 0 \le \underline{\phi}_{k_0}\right\}\right)$.

6.1.3.4 Newly Enabled Transition

For $m \in \mathbb{N}^+$, $l \in \mathsf{T}_{SU\mathcal{F}}$ and $t \in \mathsf{T}_\tau$ such that $t \in En(m)$ and $t \in Ac(m)$ we define a predicate $ne_\tau(l, m, t)$ which is true if l is newly enabled by the firing of t from m, and false otherwise. Formally, the predicate is defined by:

$ne_\tau(l, m, t) =$

$$\left\{ \begin{array}{l} l \in En\left(m - \mathsf{Pre}(t) + \mathsf{Post}(t)\right) \wedge \\ \left(l \notin En\left(m - \mathsf{Pre}(t)\right) \vee l = t\right) \end{array} \right\}.$$

For $m \in \mathbb{N}^+$, $k \in \mathsf{T}$, $t \in \left(\mathsf{T}_{SU\mathcal{F}} - \mathsf{T}_\tau\right)$ and $t' \in CT_{SU\mathcal{F}}(t)$ such that $t, t' \in En(m)$ and $t, t' \in Ac(m)$ the predicate $ne_{a,\bar{a}}(k, m, (t, t'))$ which is true if k is newly enabled by the firing of t and t' simultaneously from m, and false otherwise by:

$ne_{a,\bar{a}}(k, m, (t, t')) =$

$$\left\{ \begin{array}{l} k \in En\left(m - \mathsf{Pre}(t) - \mathsf{Pre}(t') + \mathrm{Post}(t) + \mathrm{Post}(t')\right) \wedge \\ \left(k \notin En\left(m - \mathsf{Pre}(t) - \mathsf{Pre}(t')\right) \vee k = t \vee k = t'\right) \end{array} \right\}.$$

The predicate $ne_{a,\bar{a}}(k, m, (t, t'))$ (resp. $ne_\tau(k, m, l)$) 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 $\mathcal{N} = (\mathsf{P}, \mathsf{T}, \mathsf{Pre}, \mathsf{Post}, Sw, \prec, m_0, \mathrm{I}_s, \Lambda)$ is a TTS $[\![\mathcal{N}]\!] = (E, e_0, A_{S\tau}, \rightarrow)$ where E is the set of states (m, I) of $\mathcal{N}$, e_0 its initial state and $\rightarrow \subseteq E \times \left(\left(A_S \times A^*\right)\{\tau\} \cup \mathbb{R}_{\ge 0}\right) \times E$ 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 $e = (m, \mathrm{I}) \xrightarrow{d} e' = (m, \mathrm{I}')$ iff

1. $d \in \mathbb{R}_{\ge 0}$
2. $\left(\forall t \in \mathsf{T}\right)\left(t \in En(m) \wedge t \in Ac(m) \Rightarrow d \le \uparrow \mathrm{I}(t)\right)$

3.

$$\left(\forall t\in \mathsf{T}\right)\begin{pmatrix} t\in En\left(m\right)\Rightarrow \mathrm{I}'\left(t\right)= \\ \text{if } \ t\in Ac\left(m\right) \text{ then } \mathrm{I}\left(t\right)\dot{-}d \end{pmatrix} \text{else } \mathrm{I}\left(t\right)$$

A continuous transition of size d is possible iff d is not greater than the latest firing time of all enabled and active transitions. (2) Prevents time to elapse as soon as the latest firing time of some active transition is reached. All firing intervals of active transitions are shifted synchronously towards the origin as time elapses, and truncated to non negative times (3). The elapsing of time has sense only for active transitions and changes of dates are thus made only for these transitions. Priorities do not modify the time-elapsing rules: all enabled transitions are considered in (3), whether or not t has priority over them.

The discrete transitions are the result of the transitions firings of the Petri net. As it is showed above, they may be partitioned into internal independent and synchronizing transitions.

- the internal independent transition relation is defined by
  $e=\left(m,\mathrm{I}\right)\xrightarrow{\ \tau\ } e'=\left(m',\mathrm{I}'\right)$ iff

1. $\left(\exists t\in \mathsf{T}_{\mathcal{SUT}}\right)\left(t\in \mathsf{T}_{\tau}\wedge t\in En\left(m\right)\wedge t\in Ac\left(m\right)\right)$
2. $0\in \mathrm{I}\left(t\right)$
3. $\left(\forall k\in \mathsf{T}_{\mathcal{SUT}}\right)\begin{pmatrix} k\in En\left(m\right)\wedge k\in \\ Ac\left(m\right)\wedge t\prec k\Rightarrow 0\notin \mathrm{I}\left(k\right) \end{pmatrix}$
4. $m'=m-\mathsf{Pre}\left(t\right)+\mathsf{Post}\left(t\right)$
5. $\left(\forall k\in \mathsf{T}_{\mathcal{SUT}}\right)\left(m'\geq \mathsf{Pre}\left(k\right)\Rightarrow \mathrm{I}'\left(k\right)=\right.$
   $\left.\text{if } \ ne_{\tau}\left(k,m,t\right) \ \text{ then } \ \mathrm{I}_{\mathrm{s}}\left(k\right) \text{ else } \mathrm{I}\left(k\right)\right.$

An internal transition t of the model $\mathrm{M}_{\mathrm{SUT}}$ may fire from a state $\left(m,\mathrm{I}\right)$ if it is enabled and active at m (1), immediately firable (2) and no internal or synchronizing transition of the SUT model with higher priority satisfies these conditions (3). The conjunction $\left(1\right)\wedge\left(2\right)\wedge\left(3\right)$ is called the fireability predicate of an internal transition t from state e and is noted $\prod_{SUT} e\xrightarrow{\ \tau\ }$. (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
  $e=\left(m,\mathrm{I}\right)\xrightarrow{\left(a,\bar{a}\right),Su\left(m'\right)} e'=\left(m',\mathrm{I}'\right)$ iff

1. $\left(\exists t\in \mathsf{T}_{\mathcal{SUT}}-\mathsf{T}_{\tau}\right)\left(\Lambda\left(t\right)=a\wedge t\in En\left(m\right)\wedge t\in Ac\left(m\right)\right)$
2. $0\in \mathrm{I}\left(t\right)$
3. $\left(\forall k\in \mathsf{T}_{\mathcal{SUT}}\right)\left(k\in En\left(m\right)\wedge k\in Ac\left(m\right)\wedge t\prec k\Rightarrow 0\notin \mathrm{I}\left(k\right)\right)$
4. $\left(\exists t'\in \mathsf{T}_{\mathcal{En}}\right)\left(\ \Lambda\left(t\right)=\bar{a}\wedge t'\in En\left(m\right)\right)$
5. $0\in \mathrm{I}\left(t'\right)$
6. $\left(\forall k'\in \mathsf{T}_{\mathcal{En}}\right)\left(k'\in En\left(m\right)\wedge t'\prec k'\Rightarrow 0\notin \mathrm{I}\left(k'\right)\right)$
7. $m'=m-\left(\mathsf{Pre}\left(t\right)+\mathsf{Pre}\left(t'\right)\right)+\mathsf{Post}\left(t\right)+\mathsf{Post}\left(t'\right)$
8. $\left(\forall k\in \mathsf{T}\right)\left(k\in En\left(m'\right)\Rightarrow \mathrm{I}'\left(k\right)=\right.$
   $\left.\text{if } \ ne_{a,\bar{a}}\left(k,m,\left(t,t'\right)\right) \ \text{ then } \ \mathrm{I}_{\mathrm{s}}\left(k\right) \text{ else } \mathrm{I}\left(k\right)\right.$

The synchronizing transitions $\left(t,t'\right)\in \mathsf{T}_{\mathcal{SUT}}\times \mathsf{T}_{\mathcal{En}}$ labeled respectively $\left(a,\bar{a}\right)$ 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 $\mathrm{M}_{\mathrm{SUT}}$ nor a transition of M_{En} with higher priorities compared to t and t' respectively satisfies these conditions (3 and 6). The conjunction $\left(1\right)\wedge\left(2\right)\wedge\left(3\right)$ (resp. $\left(4\right)\wedge\left(5\right)\wedge\left(6\right)$) 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 $\prod_{SUT} e\xrightarrow{\ a\ }$ (resp. $\prod_{Env} e\xrightarrow{\ \bar{a}\ }$). (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* ρ of $[\![\mathcal{N}]\!]$ can be defined as a finite sequence of moves

$e_0 \xrightarrow{d_0} e_0' \xrightarrow{\omega_0} e_1 \xrightarrow{d_1} e_1' \xrightarrow{\omega_1} e_2 \cdots$
$e_n \xrightarrow{\omega_n} e_{n+1}$

where discrete and continuous transitions (possibly of duration 0) alternate. The discrete transitions are either synchronizing transitions followed by the eventual suspended actions ($\omega_i = \alpha_i, \lambda_i$ where $\alpha_i = (a, \overline{a}) \in A_s$ and $\lambda_i \in A^*$ are suspended transitions) or pure transitions ($\omega_i = \tau$), and $d_{i,0\le i\le n}$ 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 $e_1 = (p_0 q_0, \{0 \le t_0,\ 0 \le k_0\})$. The firing of (t_0, k_0) labeled with (coin?, coin!) from e_1 leads to $e_2 = (m_1, D_2)$ where $m_1 : \mathrm{p_1 q_1}$; D_2 : $\begin{cases} 50 \le t_1 \le \infty \\ 30 \le t_2 \le \infty \\ 0 \le t_3 \le \infty \\ 0 \le k_1 \le \infty \end{cases}$ and $t_2 \prec t_1$ and $t_3 \prec t_1, t_2$. Suppose that the user requests a coffee at 10*t.u.* then the transitions (t_3, k_1) labeled with (req?,req!) will be fired thus defining the run $e_0 \xrightarrow{0.85} e_1 \xrightarrow{(\text{coin?},\text{coin!}),\phi} e_2 \xrightarrow{10} e_3 \xrightarrow{(\text{req?},\text{req!}),\phi} e_4$. At this time, t_1 and t_2 can't fire (the predicate $\prod_{SUT} e_3 \xrightarrow{t_3}$ is true while $\prod_{SUT} e_3 \xrightarrow{t_{i(i=1,2)}}$ are both false because $0 \notin \mathrm{I}(t_1)$ and $0 \notin \mathrm{I}(t_2)$)Thestates $e_3 = (m_1, D_3)$ and $e_4 = (m_2, D_4)$ where D_3 : $\begin{cases} 40 \le t_1 \le \infty \\ 20 \le t_2 \le \infty \\ 0 \le t_3 \le \infty \\ 0 \le k_1 \le \infty \end{cases}$ and

$m_2 : \mathrm{p_4 q_2}$; D_4 : $\begin{cases} 10 \le t_9 \le 30 \\ 0 \le k_2 \le \infty \\ 0 \le k_3 \le \infty \end{cases}$

From e_4 only the synchronizing transitions (t_9, k_3) can be fired while k_2 can't be fired ($\prod_{Env} e_4 \xrightarrow{k_2}$ is not true because the complementary transition $\overline{\Lambda(k_2)}$ 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_4 leads to e_0 passing by $e_5 = (m_2, D_5)$ $\left(e_4 \xrightarrow{20} e_5 \xrightarrow{(\text{lightCoffee!},\text{lightcoffee?}),\phi} e_0\right)$ where

$D_5 : \begin{cases} 0 \le t_9 \le 10 \\ 0 \le k_2 \le \infty \\ 0 \le k_3 \le \infty \end{cases}$.

Suppose now that the user requests a coffee at 55*t.u.* then the transitions (t_1, k_1) labeled with (req?,req!) will be fired at this time. The corresponding run is $e_2 \xrightarrow{55} e_6 \xrightarrow{(\text{req?},\text{req!}),\phi} e_7$. Observe that from e_6, despite $t_i \in En(m_1)$, $t_i \in Ac(m_1)$ and $0 \in \mathrm{I}(t_i)$ $(i = 1,2,3)$ only t_1 can be fired and the user can only have a strong coffee. The predicate $\prod_{SUT} e_6 \xrightarrow{t_1}$ is true while $\prod_{SUT} e_6 \xrightarrow{t_{i(i=2,3)}}$ are both false because $t_2, t_3 \prec t_1$. The states $e_6 = (m_1 : \mathrm{p_1 q_1}, D_6)$ where

$D_6 : \begin{cases} 0 \le t_1 \le \infty \\ 0 \le t_2 \le \infty \\ 0 \le t_3 \le \infty \\ 0 \le k_1 \le \infty \end{cases}$ and $e_7 = (m_3 : \mathrm{p_2 p_3 q_2}, D_7)$

where $D_7 : \begin{cases} 0 \le t_4 \le \infty \\ 30 \le t_8 \le 50 \\ 0 \le k_2 \le \infty \\ 0 \le k_3 \le \infty \end{cases}$.

The transitions t_4 and k_3 can't fire ($\prod_{SUT} e_7 \xrightarrow{t_4}$ and $\prod_{Env} e_7 \xrightarrow{k_3}$ are false (the complementary transition $\overline{\Lambda(k_3)}$ is not enabled and there is no complementary transition of t_4). The firing of (t_8, k_2) after

35*t.u.*(e.g. the brewing of a strong coffee takes 35*t.u.*) from e_7 leads to e_0 passing by $e_8=\left(m_3,D_8\right)$ and $e_9=\left(m_4 = p_2p_7q_0,D_9\right)$ $\left(e_7 \xrightarrow{35} e_8 \xrightarrow{(\text{strongCoffee!},\text{strongcoffee?}),\phi} e_9 \xrightarrow{\tau} e_0\right)$

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

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

Consider now the model M_2 composed of M_{SUT} and M_{En3} (Figure 4 and 7). Starting from the initial state e'_0, the timed trace $\sigma = 0.85\left(\left(\text{coin?},\text{coin!}\right),\phi\right)55\left(\left(\text{req?},\text{req!}\right),\phi\right)$ is performed by M_2. We have the sequence of transitions

$e'_0 \xrightarrow{0.85} e'_1 \xrightarrow{(\text{coin?},\text{coin!}),\phi} e'_2 \xrightarrow{55} e'_3 \xrightarrow{(\text{req?},\text{req!}),\phi} e'_4$

and then $\sigma \in \mathbf{TTr}\left(e'_0\right)$; $\text{After}\left(e'_0,\sigma\right)=\left\{e'_4\right\}$ where $e'_4=\left(m'_2,D'_4\right)$ and m'_2: $p_2p_3q_5$ and

D_4' : $\begin{cases} 0 \le t_4 \le \infty \\ 0 \le t_8 \le 15 \\ 0 \le k_6 \le \infty \\ 0 \le k_{10} \le \infty \end{cases}$.

As above, the user cans only be served a strong coffee because he has requesting for a coffee at 50*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 *5t.u* as specified in Figure 4. suppose that he takes it at 35*t.u* and return it at 4*t.u*. We have the following run

$e'_4 \xrightarrow{35} e'_5 \xrightarrow{(\text{tackeCup?},\text{tackeCup!}),\text{strongcoffee!}}$

$e'_6 \xrightarrow{4} e'_7 \xrightarrow{(\text{returnCup?},\text{returnCup!}),\phi} e'_8 \xrightarrow{0.5} e'_9$

$\xrightarrow{\tau} e'_{10} \xrightarrow{10} e'_{11} \xrightarrow{(\text{strongcoffee!},} e'_{12}$

$\xrightarrow{\text{strongcoffee?}),\phi} e'_0$

and:

2.

$Ac\left(\mathcal{M}\left(e'_6\right)\right) = \left\{\begin{array}{l} t_5,t_7,k_7,k_9 : \text{labeled respectively by returnCup?,} \\ \tau,\text{returnCup!},\text{strongcoffee?} \end{array}\right\}$

3.

$Su\left(\mathcal{M}\left(e'_6\right)\right) = \left\{t_8 \,\middle|\, \Lambda\left(t_8\right) = \text{strongCoffee!}\right\}$

4.

$\mathbf{Out}\left(e'_5\right)=\left\{\left(\phi,\text{strongCoffee!}\right)\right\} \cup \left\{d \in \mathbb{R}_{\ge 0} \,\middle|\, d \le 5\right\}$

(a suspended output strongCoffee! and a delay *d*).

6.1.4 Non-Determinism and Time

For many real-time systems the ordering or timing of 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 LPrSwTPN 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 $\mathfrak{C}$ 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, $d.(\mathsf{strongCoffee!}, \mathsf{strongcoffee?}), 30 \le d \le 50$. Note that $\mathbf{Out}(e_7) = \{(\mathsf{strongCoffee!}, \phi)\} \cup \{d \mid d \in [30,50]\}$

(An active output 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. $\mathsf{After}(e_2, 40(\mathsf{req?}, \mathsf{req!}))$ 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 γ. 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 $\underline{\gamma}$ indexed over the enabled transitions ($\underline{\gamma}_t$ is the time elapsed since *t* was last enabled). The initial state of the LPrSwTPN $M_1 = M_{SUT} \| M_{En1}$ is $s_0 = \left(p_0\, q_0,\ \underline{\gamma}_{t_0} = 0,\ \underline{\gamma}_{k_0} = 0\right)$.

Informally, the system leaves the initial state $s_0 = \left(m_0, \underline{\gamma}_t = 0 \mid t \in En(m_0)\right)$ 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 $\rightsquigarrow$ is also decomposed to:

1. The continuous transition defined by $s = (m, \underline{\gamma}) \overset{d}{\rightsquigarrow} s' = (m, \underline{\gamma}')$ iff
   a. $d \in \mathbb{R}_{\ge 0}$
   b. $(\forall t \in \mathsf{T})\left(t \in En(m) \wedge t \in Ac(m) \Rightarrow d \le \uparrow \mathrm{I}_s(t) - \underline{\gamma}_t\right)$
   c. $(\forall t \in \mathsf{T})\begin{pmatrix} t \in En(m) \Rightarrow \underline{\gamma}'_t = \\ \text{if } t \in Ac(m)\ \ \underline{\gamma}_t + d \text{ else } \underline{\gamma}_t \end{pmatrix}$
2. The internal independent transition relation defined by $s = (m, \underline{\gamma}) \overset{\tau}{\rightsquigarrow} s' = (m', \underline{\gamma}')$ iff
   a. $(\exists t \in \mathsf{T}_\tau)\left(t \in En(m) \wedge t \in Ac(m)\right)$
   b. $\underline{\gamma}_t \in \mathrm{I}_s(t)$
   c. $(\forall k \in \mathsf{T}_{sus})\begin{pmatrix} k \in En(m) \wedge k \in Ac(m) \wedge \\ t \prec k \Rightarrow \underline{\gamma}_k \notin \mathrm{I}_s(k) \end{pmatrix}$
   d. $m' = m - \mathsf{Pre}(t) + \mathsf{Post}(t)$
   e. $(\forall k \in \mathsf{T}_{sus})\left(k \in En(m') \Rightarrow \underline{\gamma}'_k = \right.$ $\left.\text{if } ne_\tau(k, m, t) \text{ then } 0 \text{ else } \underline{\gamma}_k\right)$

3. The synchronizing transition relation defined by $(m,\underline{\gamma}) \overset{(a,\bar{a}),Su(m')}{\rightsquigarrow} (m',\underline{\gamma}')$ iff

a. $\left(\exists t\in \mathsf{T}_{su\tau}-\mathsf{T}_{\tau}\right)\left(\Lambda(t)=a\wedge t\in En(m)\wedge t\in Ac(m)\right)$

b. $\underline{\gamma}_t \in \mathrm{I}_s(t)$

c. $\left(\forall k\in \mathsf{T}_{su\tau}\right)\begin{pmatrix} k\in En(m)\wedge k\in \\ Ac(m)\wedge t\prec k\Rightarrow \underline{\gamma}_k\notin \mathrm{I}_s(k)\end{pmatrix}$

d. $\left(\exists t'\in \mathsf{T}_{\mathcal{E}n}\right)\left(\Lambda(t')=\bar{a}\wedge t'\in En(m)\wedge t'\in Ac(m)\right)$

e. $\underline{\gamma}_{t'}\in \mathrm{I}_s(t')$

f. $\left(\forall k'\in \mathsf{T}_{\mathcal{E}n}\right)\left(k'\in En(m)\wedge t'\prec k'\Rightarrow \underline{\gamma}_{k'}\notin \mathrm{I}_s(k')\right)$

g. $m'=m-\left(\mathsf{Pre}(t)+\mathsf{Pre}(t')\right)+\mathsf{Post}(t)+\mathsf{Post}(t)$

h. $\left(\forall k\in \mathsf{T}\right)\left(k\in En(m')\Rightarrow \underline{\gamma}'_k=\right.$ iff $\left.ne_{a,\bar{a}}\left(k,m,(t,t')\right)\ \text{then}\ 0\ \text{else}\ \underline{\gamma}_k\right)$

From the initial state in the model M_1, if the coffee machine receives an input request from the user at $0.85t.u.$ then we have a transition $s_0 \overset{0.85}{\rightsquigarrow} s_1$ ($s_1=\left(p_0q_0,\ \underline{\gamma}_{t_0}=0.85,\ \underline{\gamma}_{k_0}=0.85\right)$). The firing of the synchronizing transitions (t_0,k_0) from s_1 leads to s_2

$$\left(s_1 \overset{(\text{coin?},\text{coin!}),\phi}{\rightsquigarrow} s_2=\left(p_1q_1,\underline{\gamma}_{t_1}=0,\underline{\gamma}_{t_2}=0,\underline{\gamma}_{t_3}=0,\ \underline{\gamma}_{k_1}=0\right)\right).$$

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 $Q=\{G\underline{\gamma}\le g\}$. The set of states denoted by $(m,Q=\{G\underline{\gamma}\le g\})$ is the set $\left\{m,\phi(\underline{\gamma})\middle|\ \underline{\gamma}\in\langle Q\rangle\right\}$, where $\langle Q\rangle$ is the solution set of Q and firing domain $\phi(\underline{\gamma})$ is the solution set in $\underline{\phi}$ of: $\underline{0}\le\underline{\phi}$ and $\underline{er}\le\underline{\phi}+\underline{\gamma}\le\underline{lt}$ where $\underline{er}_k\le\downarrow \mathrm{I}_s(k)$ and $\underline{lt}_k\le\uparrow \mathrm{I}_s(k)$. The initial symbolic state (m_0,Q_0) is obtained from the state (m_0,Q_ε) where

$Q_0=\left\{\ G\underline{\gamma}'\le g\middle|\ \left(m_0,0\le\underline{\gamma}_t\le 0\right)\overset{d}{\rightsquigarrow}\left(m_0,\underline{\gamma}'_t\right)\wedge t\in En(m_0)\ \right\}$ and $Q_\varepsilon=\left\{0\le\underline{\gamma}_t\le 0\middle|\ t\in En(m_0)\right\}$. The solution set $\langle Q_\varepsilon\rangle$ 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 $(m,Q)\overset{(a,\bar{a}),Su(m')}{\rightarrowtail}(m',Q')$ and $(m,Q)\overset{\tau}{\rightarrowtail}(m',Q')$, followed by some delay. It is performed iff $(m,\underline{\gamma})\overset{(a,\bar{a}),Su(m')}{\rightsquigarrow}(m',\underline{\gamma}'')$ or $(m,\underline{\gamma})\overset{\tau}{\rightsquigarrow}(m',\underline{\gamma}'')$ and

$Q'=\left\{\begin{matrix} G\underline{\gamma}'\le g'\middle|\ (m,\underline{\gamma})\overset{(a,\bar{a}),Su(m')}{\rightsquigarrow}\\ \left(m',\underline{\gamma}''\right)\wedge\left(m',\underline{\gamma}''\right)\overset{d}{\rightsquigarrow}(m',\underline{\gamma}')\end{matrix}\right\}$ or

$Q'=\left\{G\underline{\gamma}'\le g'\middle|\ (m,\underline{\gamma})\overset{\tau}{\rightsquigarrow}\left(m',\underline{\gamma}''\right)\wedge\left(m',\underline{\gamma}''\right)\overset{d}{\rightsquigarrow}(m',\underline{\gamma}')\right\}$.

Symbolic transition relation between classes $\rightarrowtail$ are:

1. The internal symbolic transition $(m,Q)\overset{\tau}{\rightarrowtail}(m',Q')$ iff

a. $\left(\exists t\in \mathsf{T}_{su\tau}\right)\left(t\in \mathsf{T}_{\tau}\wedge t\in En(m)\wedge t\in Ac(m)\right)$

b. Q augmented with

i. $\downarrow \mathrm{I}_s(t)-\underline{\gamma}_t\le 0$

ii.

$$\left\{\downarrow \mathrm{I}_s(k) - \underline{\gamma}_k > 0 \;\middle|\; k \in \mathsf{T}_{su\mathcal{T}} \wedge k \in En(m) \wedge k \in Ac(m) \wedge t \prec k\right\}$$

is consistent

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

d. Q'' obtained by

i. The constraints (b) above, and

ii.

$$\left(\forall k \in En(m')\right)\left(\underline{\gamma''}_k = \text{if } ne_{\tau}(k,m,t) \text{ then } 0 \text{ else } \underline{\gamma}_k\right)$$

iii. The variables $\underline{\gamma}$ are eliminated

e. Q'' augmented with

i. $d \geq 0$

ii.

$$\left\{d \leq \uparrow \mathrm{I}_s(k) - \underline{\gamma''}_k \;\middle|\; k \in En(m') \wedge k \in Ac(m')\right\}$$

is consistent

f. Q' obtained by the constraints (5) above, and

i.

$$\left(\forall k \in En(m')\right)\begin{Bmatrix}\underline{\gamma'}_k = \text{if } k \in Ac(m') \text{ then } \underline{\gamma''}_k + \\ d \text{ else } \underline{\gamma'}_k\left(k \in Su(m')\right)\end{Bmatrix}$$

ii. The variables $\underline{\gamma''}$ and d are eliminated

2. The synchronizing symbolic transition $(m,Q) \overset{(a,\bar{a}),Su(m')}{\longmapsto} (m',Q')$ iff

a.

$$\left(\exists t \in \mathsf{T}_{su\mathcal{T}}\right)\begin{pmatrix} t \in \mathsf{T} - \mathsf{T}_{\tau} \wedge \Lambda(t) = \\ a \wedge t \in En(m) \wedge t \in Ac(m)\end{pmatrix}$$

b. Q augmented with

i. $\downarrow \mathrm{I}_s(t) - \underline{\gamma}_t \leq 0$

ii.

$$\left\{\begin{array}{l}\downarrow \mathrm{I}_s(k) - \underline{\gamma}_k > \\ 0 \;\middle|\; k \in \mathsf{T}_{su\mathcal{T}} \wedge k \in En(m) \wedge k \in Ac(m) \wedge t \prec k\end{array}\right\}$$

c.

$$\left(\exists t' \in \mathsf{T}_{\mathcal{E}n}\right)\left(\Lambda(t') = \bar{a} \wedge t' \in En(m) \wedge t' \in Ac(m)\right)$$

d. Q augmented with

i. $\downarrow \mathrm{I}_s(t') - \underline{\gamma}_{t'} \leq 0$

ii. $\left\{\begin{array}{l}\downarrow \mathrm{I}_s(k') - \underline{\gamma}_{k'} > 0 \\ \;\middle|\; k' \in \mathsf{T}_{\mathcal{E}n} \wedge k' \in En(m) \wedge t' \prec k'\end{array}\right\}$ is consistent

e. $m' = m - \left(\mathsf{Pre}(t) + \mathsf{Pre}(t')\right) + \mathsf{Post}(t) + \mathsf{Post}(t)$

f. Q'' obtained by (a) The constraints (2) and (4) above, and

i.

$$\left(\forall k \in En(m')\right)\left\{\begin{array}{l}\underline{\gamma''}_k = \\ \text{if } ne_{a,\bar{a}}(k,m,(t,t')) \text{ then } 0 \text{ else } \underline{\gamma}_k\end{array}\right)$$

ii. The variables $\underline{\gamma}$ are eliminated

g. Q'' augmented with

i. $d \geq 0$

ii.

$$\left\{d \leq \uparrow \mathrm{I}_s(k) - \underline{\gamma''}_k \;\middle|\; k \in En(m') \wedge k \in Ac(m')\right\}$$

is consistent

h. Q' obtained by

i. The constraints (g) above, and

ii.

$$\left(\forall k \in En(m')\right)\begin{Bmatrix}\underline{\gamma'}_k = \\ \text{if } k \in Ac(m') \text{ then } \underline{\gamma''}_k + \\ d \text{ else } \underline{\gamma''}_k\left(k \in Su(m')\right)\end{Bmatrix}$$

iii. The variables $\underline{\gamma''}$ 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* (*rswtioco*). This relation supports reactivity, timing, suspension/resumption principal characteristics of real time systems and tacks environment assumptions into account. So, it is indexed by the name of the considered environment ($rswtioco_{\mathrm{En}}$). 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 $rswtioco_{En}$ 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 *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 *SUT* may delay only if the specification also may delay). Unlike *tioco* (Krichen, 2009) *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 *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 *a* (see Algorithm 1).

The *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 *rswtioco* relation makes it possible to answer another question: "does some action *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 *LPrSwTPN* denoted M_{HI} (hypothesis implementation model). For the SUT to be testable the *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 *LPrSwTPN*, and that the SUT can be modelled by some unknown weak-input enabled and non-blocking *LPrSWTPN*. We allow for the SUT to be reset to its initial state.

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

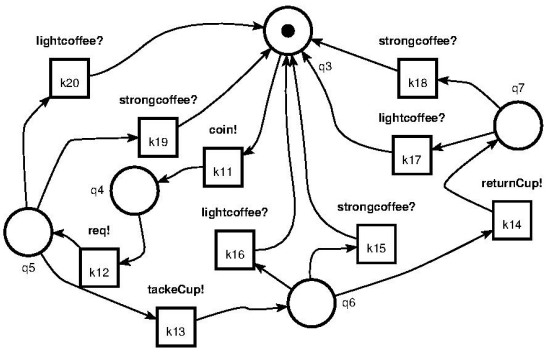
$$I\ rswtioc_{En}\ S \quad iff \quad \forall \sigma \in TTr(M) : Out(After(e_0,\sigma)) \subseteq Out(After(s_0,\sigma)) \tag{1}$$

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

$\mathcal{E}n \sqsubseteq \mathcal{E}n'$ iff $rswtioc_{\mathcal{E}n'} \subseteq rswtioc_{\mathcal{E}n}$ (To be read En' is more discriminating than En).

It follows from the definition of $rswtioco_{En}$ that $\mathcal{E}n \sqsubseteq \mathcal{E}n'$ iff the behaviour of $\mathcal{E}n$ is included in the behaviour of $\mathcal{E}n'$. There is a most

Figure 8. An environment model M_{En4}



(least) discriminating input enabled and non-blocking environment Enu (Eno) (See Figure 10(a) and 10(b)) given by $TTr(\mathcal{E}nu) = (A \cup \mathbb{R}_{\geq 0})^* \left(TTr(\mathcal{E}no) = (A_{out} \cup \mathbb{R}_{\geq 0})^* \right)$. The corresponding conformance relation $rswtioco_{\text{Enu}}$ ($rswtioco_{\text{Eno}}$) specializes to simple trace inclusion (timed output trace inclusion).

Moreover, because we treat environment constraints explicitly and separately, $rswtioco_{\text{En}}$ 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 $\mathcal{I}\ rswtioc_{\mathcal{E}n}\ \mathcal{S}$, then without re-testing

$$\text{if } \mathcal{S} \sqsubseteq \mathcal{S}' \text{ then } \mathcal{I}\ rswtioc_{\mathcal{E}n}\ \mathcal{S}' \tag{2}$$

$$\text{if } \mathcal{E}n' \sqsubseteq \mathcal{E}n \text{ then } \mathcal{I}\ rswtioc_{\mathcal{E}n'}\ \mathcal{S} \tag{3}$$

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_1(Il, Is) 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 $\left[\downarrow \mathrm{I}l, \uparrow \mathrm{I}l\right]$ ($\left[\downarrow \mathrm{I}s, \uparrow \mathrm{I}s\right]$). Notice

Figure 9. M_{SUT1}: A SUT: I_1((Il, Is) of the coffee machine

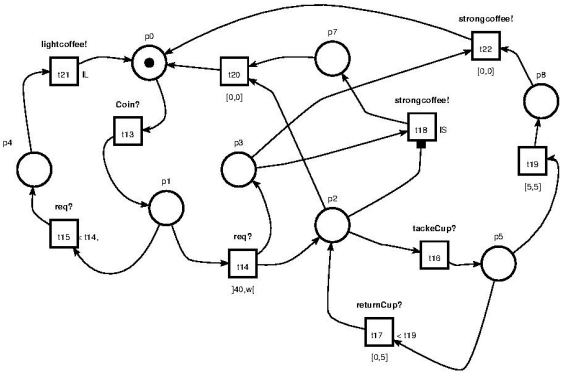
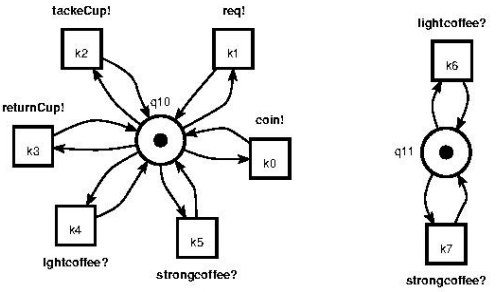


Figure 10. (a) the environment Enu (b) the environment Eno

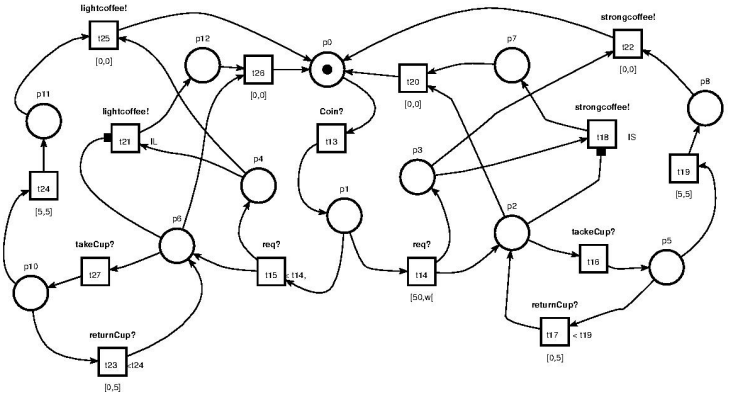


that $\mathcal{I}_I\left(\left[\downarrow I\!I, \uparrow I\!I\right], \left[\downarrow \mathrm{I}s, \uparrow \mathrm{I}s\right]\right)$ does not allow the user requesting light coffee to take his cup before the brewing time I*I*. Observe that any trace of the SUT I_1([20, 25],[40, 45]) in E_{nu} can be matched by the specification; hence I_1([20, 25],[40, 45]) *rswtioco*$_{Enu}$ *S*. Thus also I_1([20, 25],[40, 45]) *rswtioco*$_{Eni}$ *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.(coin?,coin!)ø.30.(req?,req!)ø.3.(lightCoffee!,lightcoffee?)ø 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.(coin?,coin!)ø.50.(req?,req!)ø.65.(strongCoffee!,strongcoffee?)ø 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_1([2, 5],[40, 45]) is used in the restricted environment E_{n2}; here despite the remaining light coffee error in E_{nu}, I_1([2, 5],[40, 45]) *rswtioco*$_{En2}$ *S* because E_{n2} never requests light coffee. The SUT I_2 shown in Figure 11 is different from I_1. It allows all users to take their coffee during its preparation (including those requesting light coffee). We have I_2([20, 25],[40, 45]) rswtioco$_{Enu}$ *S* and I_2([20, 25],[45, 45]) rswtioco$_{Eni}$ *S* for *i=3,4* because it has the timed trace 10.(coin?,coin!)ø.30.(req?,req!)ø.10.(tackeCup?,tackeCup!)lightCoffee.4.(returnCup?,returnCup!)lightCoffee.10.(lightCoffee!,lightcoffee?)ø that *S* does not. 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 $\left]50, 60\right]$ then we have I_2([20, 25],[45, 45]) *rswtioco*$_{Eni}$ *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_{n1} never requests light coffee.

8. TEST GENERATION AND EXECUTION

The Main idea of the algorithm1 is the following. Start where *Q* contains the initial symbolic state $\left(m_0, Q_0\right)$. 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

Figure 11. M_{SUT2}: Another SUT: I_2((Il, Is) of the coffee machine



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 $\mathcal{Q}$ to the SUT, then updates $\mathcal{Q}$ according to $\mathsf{After}\left(\mathcal{Q},\left(a?,a!\right)\right)$. (2) Waits for an output o after a delay δ. Randomly chooses how long it will wait. If an active output $active\left(o\right)$ and/or suspended output(s) $suspend\left(o\right)$ 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 $\mathcal{Q}$ according to how long it has waited ($\mathsf{After}\left(\mathcal{Q},\delta'\right)$. It checks also if the outputs are legal ones according to $\mathcal{Q}$. If they are legal outputs, $\mathcal{Q}$ is updated according to $\mathsf{After}\left(\mathcal{Q},\left(active(o),\overline{active(o)}\right)\right)$; else a fail verdict is assigned. If no output is observed during sleeping the set $\mathcal{Q}$ is updated according to the passing of time ($\mathsf{After}\left(\mathcal{Q},\delta\right)$). (3) Resets the SUT and restarts.

The functions used in Algorithm 1 are defined as:

$$EnvOutput\left(\mathcal{Q}\right) = \left\{a! \in A_{in} \;\middle|\; \left(\exists t \in \mathrm{T}_{\mathcal{E}n} \wedge \exists e \in \mathcal{Q}\right)\begin{pmatrix}\Lambda\left(t\right) = \\ a! \wedge \prod_{Env} e \xrightarrow{a!}\end{pmatrix}\right\}$$

$$Delays\left(\mathcal{Q}\right) = \left\{d \leq \delta \;\middle|\; \begin{pmatrix}\left(\exists t \in \mathrm{T}_{\mathcal{E}n} \wedge \exists e \in \mathcal{Q}\right) \\ \left(\Lambda\left(t\right) = a! \wedge e \xrightarrow{\delta} e' \wedge \prod_{Env} e' \xrightarrow{a!}\right)\end{pmatrix}\right\}$$

$$ImpOutput\left(\mathcal{Q}\right) = \left\{a! \in A_{out} \;\middle|\; \begin{pmatrix}\left(\exists t \in \mathrm{T}_{\mathcal{SUT}} \wedge \exists e \in \mathcal{Q}\right) \\ \left(\Lambda\left(t\right) = a! \wedge \prod_{SUT} e \xrightarrow{a!}\right)\end{pmatrix}\right\}$$

$$ImpSuspend\left(\mathcal{Q}\right) = \left\{a \in A \;\middle|\; \begin{pmatrix}\left(\exists t \in \mathrm{T}_{\mathcal{SUT}} \wedge \exists e \in \mathcal{Q}\right) \\ \left(\Lambda\left(t\right) = a \wedge t \in Su\left(\mathcal{M}\left(e\right)\right)\right)\end{pmatrix}\right\}$$

$EnvOutput\left(\mathcal{Q}\right)$ is the set of input actions (enabled environment outputs) that are allowed by the environment in the current symbolic state set $\mathcal{Q}$, and is empty if the environment model has no outputs to offer. *RandomlyChooseAction* selects randomly an input from $EnvOutput\left(\mathcal{Q}\right)$ 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, *Delays* must

pick a real number from the interval that fulfils those constraint. $ImpOutput(\mathcal{Q})$ is the set of active output actions that are allowed according to SUT specification in the current state set. $ImpSuspend(\mathcal{Q})$ 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 $active(o)$ and $suspend(o)$. Any illegal occurrence or absence of a standard output or a delay from the SUT is detected if the set $\mathcal{Q}$ update by the function 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. *GenExeTest*$(\mathcal{S}, \mathcal{E}, SUT, N)$ *// Generation and execution of test.*

```
Initially Q := {(m0, Q0)}
while Q ≠ φ ∧ iterations ≤ N do RandomlyChoose(Action, Delay, Restart)
    case Action:  // offer an input to the SUT
     If EnvOutput(Q) ≠ ∅ then
                    a!:= RandomlyChooseAction(EnvOutput(Q))
                    send a!  to the SUT
                    Q := After(Q,(a?,a!))
    case Delay: // wait for an output of the SUT. Sleep for δ and wake up on
output o
                                (o contains eventually suspended actions) sent
by the SUT.
      δ := RandomlyChooseDelay(Delays(Q))
      if  o occurs at  δ' ≤ δ  then  do
                                Q := After(Q,δ')
                                a!:= active(o)
                                if  Suspend(o) ⊄ ImpSuspend(Q)  then
                                     do return fail
                                          For each
b! ∈ ImpOutput(Q) ∩ Suspend(o)
                                               then "b! must be a an active
output"
                                          done
                                if  a! ∉ ImpOutput(Q)  then do return fail
                                          if  a! ∈ ImpSuspend(Q) then "a!
must
                                                be a suspended action"
                                          done
                                          else  Q := After(Q,(a!,a?))
                           done
          else  Q := After(Q,δ)
    case Restart:        Q := {(m0, Q0)}  and Reset SUT          // reset and
restart
If  Q = ∅ then return fail  else return  pass
```

Algorithm 2. After $\left(\mathfrak{Q},\left(a,\overline{a}\right)\right)$

Passed $:=\varnothing$, Waiting $:=$ Closure$_{\tau}\left(\mathfrak{Q}\right)$

For each symbolic state $\left(m,Q\right)\in$ Waiting

For each symbolic transition $\left(m,Q\right)\overset{\left(a,\overline{a}\right),Su\left(m'\right)}{\rightarrowtail}\left(m',Q'\right)$ **if** $\neg contains\left(Passed,\left(m',Q'\right)\right)$

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

return Closure$_{\tau}\left(Passed\right)$

Algorithm 3. After $\left(\mathfrak{Q},\delta\right)$

After $\left(\mathfrak{Q},\delta\right)=\left(\left\{\left(m,Q'\right)\middle|\left(m,Q\right)\in\text{Closure}_{\delta\tau}\left(\mathfrak{Q},\delta\right)\right\}\right)$

SUT is detected if the outputs $suspend\left(o\right)$ are not (resp. are) in $ImpSuspend\left(\mathfrak{Q}\right)$.

The function After $\left(\mathfrak{Q},_{\alpha}\right)$ (resp. After $\left(\mathfrak{Q},_{\delta}\right)$) computes the set of states $\mathfrak{Q}'$ reachable after the action $\alpha\in A_S$ (resp. time delay δ). 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 Closure$_{\delta\tau}\left(\mathfrak{Q},d\right)$ that collects the reachable symbolic state set within a delay of d. The

Algorithm 4. Closure$_{\delta\tau}\left(\mathfrak{Q},d\right)$

Passed $:=\varnothing$, Waiting $:=\mathfrak{Q}$

while Waiting $\neq\varnothing$ **do**

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

if $\left(m,Q\right)\overset{d'}{\rightarrowtail}\left(m,Q'\right)$ where $d'\leq d$ **then**

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

For each symbolic transition $\left(m,Q'\right)\overset{\tau}{\rightarrowtail}\left(m',Q''\right)$ **if** $\neg contains\left(Passed,\left(m',Q''\right)\right)$ **then**

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

return *Passed*

Algorithm 5. $\text{Contains}\left(\mathcal{Q},\left(m,Q\right)\right)$

```
For each state (m,Q') ∈ Q  if  ⟨Q⟩ ⊆ ⟨Q'⟩  then return true
return false
```

predicate $\text{Contains}\left(\mathcal{Q},\left(m,Q\right)\right)$ tests whether a symbolic state $\left(m,Q\right)$ is covered by a symbolic state in $\mathcal{Q}$. $\left\langle Q\right\rangle$ is the set of solutions of the temporal variables associated with the enabled transitions. $\text{Closure}_{\tau}\left(\mathcal{Q}\right)$ is $\text{Closure}_{\delta\tau}\left(\mathcal{Q},0\right)$. 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 *rswtioco*. 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. 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

Adjir, N., de Saqui-Sannes, P., & Rahmouni, K. M. (2009). Testing real-time systems using TINA. [). Springer-Verlag.]. *TESTCOM-FATES, LNCS*, *5826*, 1–15.

Alur, R., & Dill, D. (1994). A theory of timed automata. *Theoretical Computer Science*, *126*, 183–235. doi:10.1016/0304-3975(94)90010-8

Bérard, B., Cassez, F., Haddad, S., Lime, D., & Roux, O. H. (2005). When are timed automata weakly timed bisimilar to time petri nets? In FSTTCS, LNCS (Vol. 3821, pp. 273-284). Springer.

Bérard, B., & Dufourd, C. (2000). Timed automata and additive clock constraints. *Information Processing Letters*, *75*(1–2), 1–7. doi:10.1016/S0020-0190(00)00075-2

Bérard, B., & Haddad, S., & Sassolas. (2012). Interrupt timed automata: Verification and expressiveness. *Formal Methods in System Design*, *40*(1), 41–87. doi:10.1007/s10703-011-0140-2

Berthomieu, B., Lime, D., Roux, O. H., & Vernadat, F. (2007). Reachability problems and abstract state space for timed petri nets with stopwatches. *Discrete Event Dynamic Systems*, *17*(2), 133–158. doi:10.1007/s10626-006-0011-y

Berthomieu, B., Peres, F., & Vernadat, F. (2006). Bridging the gap between timed automata and bounded time petri nets. In FORMATS, LNCS, (Vol. 4202, pp. 82-97). Springer-Verlag.

Berthomieu, B., Peres, F., & Vernadat, F. (2007). Model checking bounded prioritized time petri nets. In *ATVA, LNCS* (Vol. 4762, pp. 523–532). Tokyo, Japan: Springer. doi:10.1007/978-3-540-75596-8_37

Berthomieu, B., Peres, F., & Vernadat, F. (2008). Abstract state spaces for time petri nets analysis. In ISORC, IEEE computer Society, Orlando, FL (pp. 298–304).

Berthomieu, B., Ribet, P. O., & Vernadat, F. (2004). The tool TINA: Construction of abstract state spaces for petri nets and time petri nets. *International Journal of Production Research*, *42*(14), 2741–2756. doi:10.1080/00207540412331312688

Bertrand, N., Jéron, T., Stainer, A., & Krichen, M. (2012). Off-line test selection with test purposes for non-deterministic timed automata. *Logical Methods in Computer Science, 8*(4).

Bouyer, P. (2004). Forward analysis of updatable timed automata. *Formal Methods in System Design*, *24*(3), 281–320. doi:10.1023/B:FORM.0000026093.21513.31

Bouyer, P., Serge, H., & Reynie, P. A. (2006). *Extended timed automata and time petri nets* (pp. 91–100). Turku, Finland: IEEE Computer Society Press. doi:10.1109/ACSD.2006.6

Brinksma, E., & Tretmans, J. (2000). Testing transition systems: An annotated bibliography. In *Proceedings of the MOVEP 2000, LNCS,* (vol. 2067, pp. 187-195), Nantes, France. Springer-Verlag

Cassez, F., & Larsen, K. G. (2000). The impressive power of stopwatches. *CONCUR, LNCS*, *1877*, 138–152.

Choffrut, C., & Goldwurm, M. (2000). Timed automata with periodic clock constraints. *JALC*, *5*(4), 371–404.

de Vries, R., & Tretmans, J. (2000). On-the-fly conformance testing using SPIN. *Software Tools for Technology Transfer*, *2*(4), 382–393. doi:10.1007/s100090050044

Diekert, V., Gastin, P., & Petit, A. (1997). Removing epsilon-transitions in timed automata. *STACS, LNCS*, *1200*, 583–594. doi:10.1007/BFb0023491

En-Nouary, A., Dssouli, R., Khendek, F., & Elqortobi, A. (1998). Timed test cases generation based on state characterization technique. In *RTSS* (pp. 220–229). IEEE Computer Society. doi:10.1109/REAL.1998.739748

Hessel, A. Larsen k, G., Mikucionis, M., Nielsen, B., Petterson, P., & Skou, A. (2008). Testing real-time systems using UPPAAL. In Formal methods and testing, LNCS (Vol 4949, pp. 77-117). Springer.

Krichen, M., & Tripakis, S. (2009). Conformance testing for real-time systems. *Formal Methods in System Design*, *34*(3), 238–304. doi:10.1007/s10703-009-0065-1

Lin, J. C., & Ho, I. (2000). Generating real-time software test cases by time petri nets. *IJCA (EI journal)*. *ACTA Press*, *22*(3), 151–158.

Merlin, P. M., & Farber, J. (1976). Recoverability of communication protocols: Implications of a theoretical study. *IEEE Transactions on Communications*, *24*(9), 1036–1043. doi:10.1109/TCOM.1976.1093424

Nielsen, B., & Skou, A. (2001). Automated test generation from timed automata. In *TACAS* (Vol. 2031, pp. 343–357). Genova, Italy: LNCS.

Tretmans, J. (1999). Testing concurrent systems: A formal approach. In J. C. M. Baeten, & S. Mauw (Eds.), *CONCUR, ICCT, LNCS* (Vol. 1664, pp. 46–65). Springer-Verlag.

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.