Formal testing of systems presenting soft and hard deadlines *

Mercedes G. Merayo, Manuel Núñez, and Ismael Rodríguez

Dept. Sistemas Informáticos y Computación
Universidad Complutense de Madrid, E-28040 Madrid, Spain
e-mail: mgmerayo@fdi.ucm.es,mn@sip.ucm.es,isrodrig@sip.ucm.es

Abstract. We present a formal framework to specify and test systems presenting both soft and hard deadlines. While hard deadlines must be always met on time, soft deadlines can be sometimes met in a different time, usually higher, from the specified one. It is this characteristic (to formally define *sometimes*) what produces several reasonable alternatives to define appropriate *implementation relations*, that is, relations to decide wether an implementation is correct with respect to a specification. In addition to introduce these relations, we define a testing framework to test implementations.

1 Introduction

Formal methods refer to techniques based on mathematics for the specification, development, and verification of systems. The use of formal methods is especially relevant in reliable systems where, due to safety and security reasons, it is important to ensure that errors are not included during the development process. Formal methods are particularly effective when used early in the development process, at the requirements and specification levels, but can be used for a complete formal development of a system. In this regard, and considering specification formalism, we may mention the (original) notions of process algebras, Petri nets, and Moore/Mealy machines. Once the roots were well consolidated other considerations were taken into account. The next step was to deal with quantitative information such as the *time* underlying the performance of systems or the *probabilities* resolving the non-deterministic choices that a system may undertake. These characteristics gave raise to new models where time and/or probabilities were included (for example, [1–8] among many others).

The formal representation of systems allows to rigorously analyze their properties. In particular, it allows to establish the *correctness* of the system with respect to a specification or the fulfillment of a specific set of required conditions, to check the semantic *equivalence* of two systems, to analyze the *preference*

* This research was partially supported by the Spanish MEC projects MASTER/TERMAS TIC2003-07848-C02-01 and WEST/FAST TIN2006-15578-C02-01, the Junta de Castilla-La Mancha project PAC06-0008, the Comunidad de Madrid project to fund research groups CAM-910606, and the Marie Curie project MRTN-CT-2003-505121/TAROT.

of a system to another with respect to a given criterion, to predict the possibility of *incorrect behaviors*, to establish the *performance* level of a system, etc. In this line, formal testing techniques allow to test the correctness of a system with respect to a specification. Formal testing originally targeted the functional behavior of systems, such as determining whether the tested system can, on the one hand, perform certain actions and, on the other hand, does not perform some unexpected ones. The application of formal testing techniques to check the correctness of a system requires to identify the *critical* aspects of the system, that is, those aspects that will make the difference between correct and incorrect behaviors. While the relevant aspects of some systems only concern *what* they do, in some other systems it is equally relevant *how* they do what they do. Thus, formal testing techniques are recently also dealing with *non-functional* properties such as the probability of an event to happen or the time that it takes to perform a certain action (for example, [9–16]).

One of the problems when specifying timed systems is that it is not always easy to precisely establish the time bounds associated with the tasks that the system performs. Thus, it is sometimes useful to allow some degree of *indecision* in such specifications. In this line, *stochastic models* (for example, [17–23]) allow to specify constraints such as *with probability p the task must finish before t time units have passed*. So, the specifier does not need to provide the precise point of time associated with a task, but a probabilistic estimation of the time value(s). However, there are situations where the specifier either does not have such probabilistic information or does not want to provide such information because it might unnecessarily complicate the model. In this case, it seems that the most appropriate way to specify time constraints is to use *time intervals*, that is, the specifier provides a set of possible time values, instead of just one, but without quantifying the probability that each value of the interval has. Moreover, it may happen that while testing the correctness of a system the tester allows some *imprecision* in the temporal behavior of the system. For example, if the specifier cannot precisely define the temporal constraints of a system, the tester can also have problems to determine what is the exact notion of passing a test *on time*. Moreover, it can be admissible that the execution of a task sometimes lasts more than expected: If most of the times the task is performed on time, a couple of delays can be tolerated. This is the idea of a *soft* deadline, in contrast with *hard* deadlines that have to be always met on time. Finally, another reason for the tester to allow imprecisions, it may happen that the artifacts measuring time while testing a system are not as precise as desirable. In this case, an apparent wrong behavior due to bad timing can be in fact correct since it may happen that the *watches* are not working properly.

In this paper we propose a formal framework to specify and test systems where time considerations can fall in some of the cases commented in the previous paragraph. Time will be introduced in specifications by extending classical finite state machines with time intervals associated to the performance of actions. Intuitively, transitions in finite state machines indicate that if the machine is in a state s and receives and input i then it will produce and output o and it will

change its state to s'. An appropriate notation for such a transition could be $s \xrightarrow{i/o} s'$. If we consider our timed extension of finite state machines, a transition such as $s \xrightarrow{i/o}_{[t_1,t_2]} s'$ means that if the machine is in state s and receives the input i, it will perform the output o and reach the state s', and it will take a time greater than or equal to t_1 but smaller than or equal to t_2.

Testing, as well as the definition of implementation relations, will depend on measuring time values and accepting the performance of the system if the time behavior is correct *up to* an admissible error. The possible definition of *admissible* will give raise to several alternative implementation relations and several notions of passing a test. However, there is still a last issue that must be taken into account when dealing with systems where time requirements are given by means of intervals. Since we assume a black-box testing framework, we cannot check that the intervals governing the behavior of the implementation are correctly related with the ones corresponding to the implementation. In fact, the execution of a test will return the time that it took to be performed, not the associated time interval. As a consequence, since we assume that time intervals are non-negative real numbers, we would need an infinite number of observations from a transition of the implementation (with an unknown time interval) to assure that its time interval is correct with the respect to the one of the specification (which it is accessible).

Even though there are several papers devoted to formal testing of timed systems [10, 9, 11, 12, 15, 16], we are aware of only one work where the topic of *non-strict* deadlines is considered in a testing framework. In [24], a probabilistic formalism is used to approximate the idea of soft deadline. However, their approach is not very related to ours since, on the one hand, they are based on [25, 26], and, on the other hand, they use a probabilistic approach, based on [27], to deal with soft deadlines. Testing relations to compare processes are based on the responses of the processes to *all* the tests, while we apply tests, derived from specifications, to implementations to determine whether the implementation is *somehow* correct with respect to the specification. As we mentioned before, stochastic models allow to partially simulate soft deadlines. In this line, there are two proposals to test stochastic systems [28, 29]. Since they are also inspired by [25, 26], these contributions are not related to the one presented in this paper.

The rest of the paper is organized as follows. In Section 2 we introduce our notion of timed finite state machine and give some auxiliary notation. In Section 3 we give our timed conformance relations. In Section 4 we show how tests are defined and applied to implementations. Finally, in Section 5 we give our conclusions and some lines for future work.

2 Extending Finite State Machines with Time Intervals

In this section we introduce our notion of timed finite state machine, that we call `IFSM`, and some concepts that will be used along the paper. The main difference with respect to usual `FSM`s consists in the addition of *time* to indicate the lapse

between offering an input and receiving an output. First we introduce notation related to time intervals, sets, and multisets.

Definition 1 We say that $d = [a_1, a_2]$ is a *time interval* if $a_1 \in \mathbb{R}_+$, $a_2 \in \mathbb{R}_+ \cup \{\infty\}$, and $a_1 \leq a_2$. From now on we assume that for all $r \in \mathbb{R}_+$ we have $r < \infty$, $r + \infty = \infty$, and $\frac{r}{\infty} = 0$. We consider that $\mathcal{I}_{\mathbb{R}_+}$ denotes the set of time intervals. We write $\pi_i(d)$, for $i \in \{1, 2\}$, to denote the value a_i.

Given two time intervals $d_1 = [a_{11}, a_{12}]$ and $d_2 = [a_{21}, a_{22}]$, $d_1 + d_2$ denotes the time interval $[a_{11}+a_{21}, a_{12}+a_{22}]$. Addition of time intervals can be generalized to n summands in the expected way. Given n time intervals $d_1 = [a_{11}, a_{12}], \ldots, d_n = [a_{n1}, a_{n2}]$, we have that $\sum d_i$ denotes the time interval $[\sum a_{i1}, \sum a_{i2}]$.

Given a set S, we consider that $|S|$ denotes the cardinal of S, $\mathcal{P}(S)$ denotes the powerset of S, and $\wp(S)$ denotes the powermultiset of S, that is, the set of multisets conformed from elements belonging to S. We will use the symbols $\{\!|$ and $|\!\}$ to denote multisets. Given a multiset $\mathcal{M}$, we write $r \in \mathcal{M}$ if r appears in $\mathcal{M}$ (that is, r has multiplicity greater than 0). We write $\|\mathcal{M}\|$ to denote the cardinal of $\mathcal{M}$ including multiplicity of its elements. For example, $\|\{\!|1, 2, 3, 1, 2|\!\}\| = 5$. □

A temporal requirement such as $[t_1, t_2]$ indicates that the associated task should take at least t_1 time units and at most t_2 units to be performed. Intervals like $[0, t_2]$, $[t_1, \infty]$, or $[0, \infty]$ denote the absence of a temporal lower/upper bound and the absence of any bound, respectively. Let us note that in the case of $[t_1, \infty]$ and $[0, \infty]$ we are abusing the notation since these intervals represent, in fact, the intervals $[t_1, \infty)$ and $[0, \infty)$, respectively.

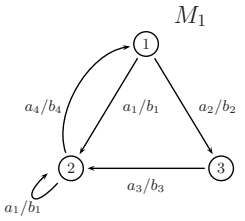
Definition 2 An *Interval Finite State Machine*, in the following `IFSM`, is a tuple $M = (S, I, O, Tr, s_{in})$ where S is a finite set of states, I is the set of input actions, O is the set of output actions, Tr is the set of transitions, and s_{in} is the initial state.

A transition belonging to Tr is a tuple (s, s', i, o, d) where $s, s' \in S$ are the initial and final states of the transition, $i \in I$ and $o \in O$ are the input and output actions, and $d \in \mathcal{I}_{\mathbb{R}_+}$ is the time interval associated with the transition.

We say that the `IFSM` M is *input-enabled* if for all state $s \in S$ and input $i \in I$, there exist s', o, and d such that $(s, s', i, o, d) \in Tr$. We say that the `IFSM` M is *observable* if there do not exist two different transitions (s, s_1, i, o, d_1) and (s, s_2, i, o, d_2). □

Intuitively, a transition (s, s', i, o, d) indicates that if the machine is in state s and receives the input i then, after a time belonging to the interval d has passed, the machine emits the output o and moves to s'. In Figure 1 we give a graphical example of an `IFSM`.

Next, we introduce the notion of *trace*. As usual, a trace is a sequence of input/output pairs. In addition, we have to record the possible time values, that is a time interval, where the trace can be performed. An *evolution* is a trace starting at the initial state of the machine.



$I = \{a_1, a_2, a_3, a_4\}, O = \{b_1, b_2, b_3, b_4\}$
$t_{12} = (s_1, s_2, a_1, b_1, [1, 3])$
$t_{13} = (s_1, s_3, a_2, b_2, [2, 6])$
$t_{32} = (s_3, s_2, a_3, b_3, [1, 2])$
$t_{21} = (s_2, s_1, a_4, b_4, [3, 9])$
$t_{22} = (s_2, s_2, a_1, b_1, [4, 5])$

Fig. 1. Example of `IFSM`.

Definition 3 Let $M = (S, I, O, Tr, s_{in})$ be an `IFSM`. A *timed trace*, or simply *trace*, of M is a tuple $(s, s', (i_1/o_1, \ldots, i_r/o_r), d)$ if we have that there exist transitions (s, s_1, i_1, o_1, d_1),..., $(s_{r-1}, s', i_r, o_r, d_r) \in Tr$, such that $d = \sum d_i$. We say that $(i_1/o_1, \ldots, i_r/o_r)$ is a *non-timed evolution*, or simply *evolution*, of M if we have that $(s_{in}, s', (i_1/o_1, \ldots, i_r/o_r), d)$ is a trace of M for some $d \in \mathcal{I}_{\mathbb{R}_+}$ and $s' \in S$. We denote by $\texttt{NTEvol}(M)$ the set of non-timed evolutions of M.

We say that the pair $((i_1/o_1, \ldots, i_r/o_r), d)$ is a *timed evolution* of M if we have that $(s_{in}, s', (i_1/o_1, \ldots, i_r/o_r), d)$ is a trace of M. We denote by $\texttt{TEvol}(M)$ the set of timed evolutions of M. □

Let us consider again the `IFSM` depicted in Figure 1 and its transitions t_{13}, t_{32}, and t_{21}. We can build the trace $(s_1, s_1, (a_2/b_2, a_3/b_3, a_4/b_4), [6, 17])$ based on these transitions. This trace represents that from state 1 the machine can accept the sequence of inputs (a_2, a_3, a_4) and it will emit the sequence of outputs (b_2, b_3, b_4) after a time belonging to the interval $[6, 17]$ has passed.

3 Implementation Relations

In this section we introduce our implementation relations. Following the classical pattern, we consider that an implementation *conforms* to a specification if for all possible sequence of inputs that the specification can perform, the outputs emitted by the implementation are a subset of those for the specification. Intuitively, this means that the implementation cannot *invent* a behavior (that is, an output) for those traces that the specification can perform. This pattern is borrowed from `ioco` [30] and was introduced in the context of finite state machines in [31].

A specification is an `IFSM`. Regarding implementations, we consider that they are also given by means of `IFSM`s. Besides, we assume that input actions are always enabled in any state of the implementation, that is, implementations are input-enabled according to Definition 2. This is a usual condition to assure that the implementation will react (somehow) to any input appearing in the specification. In order to simplify the presentation, we will consider that both specifications and implementations are given by observable `IFSM`s (see Definition 2). Let us note that even restricting to this kind of machines we may still have two

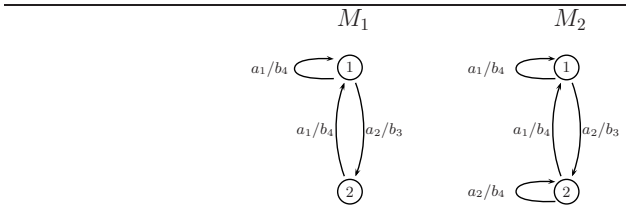


Fig. 2. Examples of non-timely conformance.

transitions (s, s_1, i, o_1, d_1) and (s, s_2, i, o_2, d_2), as far as $o_1 \neq o_2$. Thus, we allow some degree of non-determinism.

Definition 4 Let S and I be two IFSMs. We say that I *non-timely conforms* to S, denoted by $I\,\mathtt{conf}_{nt}\,S$, if for all $e = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o_r) \in \mathtt{NTEvol}(S)$, with $r \geq 1$, we have that

$$e' = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o'_r) \in \mathtt{NTEvol}(I) \Longrightarrow e' \in \mathtt{NTEvol}(S)$$

□

In the previous definition, let us note that if the specification would have also the property of input-enabled then we may remove the condition "for all $e = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o_r) \in \mathtt{NTEvol}(S)$, with $r \geq 1$ ", so that we simply have to check trace inclusion.

Example 1 Let us consider the systems M_1 and M_2 depicted in Figure 2 where time information has been omitted. We have $M_2\ \mathtt{conf}_{nt}\ M_1$. Let us note that the non-timed evolutions of M_2 having as prefix the sequence $(a_2/b_3, a_2/b_4)$ are not checked because M_1 (playing the role of specification) cannot perform those evolutions.

Let us now consider that M_1 is extended with the transition $(2, 2, a_2, \mathtt{null}, d)$ so that M_1 is input-enabled. Then, M_1 does not conform to M_2. For example, M_2 may perform the non-timed evolution $e = (a_2/b_3, a_2/b_4)$, M_1 has the non-timed evolution $e' = (a_2/b_3, a_2/\mathtt{null})$, but e' does not belong to the set of non-timed evolutions of M_2. Note that e and e' share the common prefix $a_2/b_3, a_2$. □

Next we introduce our first timed implementation relation. In addition to the non-timed conformance of the implementation, we require a time condition to hold: The time intervals of the implementation correspond to those of the specification.

Definition 5 Let I and S be IFSMs. We say that I *conforms in time* to S, denoted by $I\,\mathtt{conf}_{int}\,S$, if $I\,\mathtt{conf}_{nt}\,S$ and for all $e \in \mathtt{NTEvol}(I) \cap \mathtt{NTEvol}(S)$ we have that for all time interval $d \in \mathcal{I}_{\mathbb{R}_+}$

$$(e, d) \in \mathtt{TEvol}(S) \implies (e, d) \in \mathtt{TEvol}(S)$$

□

Despite its neat definition, this relation suffers from practical problems due to our assumption that the implementation under test is a black box. Even though this is a very reasonable notion of conformance, the fact that we assume a black-box testing framework disallows us to check whether the corresponding intervals coincide indeed. In fact, since we are considering that time intervals are defined over the set of non-negative real numbers, we would need an infinite number of observations from a transition of the implementation (with an unknown time interval) to assure that its time interval coincides with the one from the specification (which it is accessible). Thus, we have to give more *realistic* implementation relations that are less accurate but are *checkable*. We only need to suppose that we can actually record the time that the implementation needs to perform a given sequence. In order to that, we introduce the concept of *timed execution*. They are simply input/output sequences together with the time that it took to perform the sequence. In a certain sense, timed executions can be seen as *instances* of the timed evolutions that the implementation can perform. Regarding the definition of observed time values, we just associate with each evolution the corresponding time values.

Definition 6 Let I be an `IFSM`. We say that $((i_1/o_1, \ldots, i_n/o_n), t)$ is an *observed timed execution* of I, or simply *timed execution*, if the observation of I shows that the sequence $(i_1/o_1, \ldots, i_n/o_n)$ is performed in time t.

Let $H = \{\!\{(e'_1, t_1), \ldots, (e'_n, t_n)\}\!\}$ be a multiset of observed timed executions and $\Phi = \{e \mid \exists\, t : (e, t) \in H\}$ be a set of input/output sequences. We say that $\texttt{Obs\_Time}_H : \Phi \longrightarrow \wp(\mathbb{R}^+)$ is the *multiset of observed time values* of H for Φ if for all $e \in \Phi$ we have $\texttt{Obs\_Time}_H(e) = \{\!\{t \mid (e, t) \in H\}\!\}$. □

Next, we introduce several conformance relations where we check that the observed time values fulfill, in each case, certain conditions with respect to the appropriate time intervals. The purpose of this paper is to introduce implementation relations where the time behavior of the implementation does not exactly correspond to what we expect, that is, it partially deviates from the behavior defined in the specification. In this case, we have to take into account this possible divergence. Intuitively, we will determine whether the amount of *incorrect* time values is relevant to ensure the possible conformance of the implementation to the specification. Moreover, we measure the degree of the deviation of the observed time values, with respect to the interval. So, by considering that there cannot be any error we test a hard deadline; soft deadlines will allow a certain error, as long as it is kept under a certain bound. First, we introduce some notation to relate a set of observed time values and a time interval.

Definition 7 Let $d = [a_1, a_2] \in \mathcal{I}_{\mathbb{R}_+}$ be a time interval, $\mathcal{R}$ be a non-empty multiset of non-negative real numbers, and $0 \leq \alpha \leq 1$.

– We write $\mathcal{R} \subseteq_\alpha d$ if we have

$$\frac{\|\{\!\{r \mid r \in \mathcal{R} \ \wedge \ (r < a_1 \ \vee \ r > a_2)\}\!\}\|}{\|\mathcal{R}\|} \leq 1 - \alpha$$

– We write $\mathcal{R} \preceq_\alpha d$ if we have

$$\frac{\|\{\!\{r \mid r \in \mathcal{R} \ \wedge \ r < a_1\}\!\}\|}{\|\mathcal{R}\|} \leq 1-\alpha \text{ and } \|\{\!\{r \mid r \in \mathcal{R} \ \wedge \ r > a_2\}\!\}\| = 0$$

– We write $\mathcal{R} \ll_\alpha d$ if we have

$$\frac{\|\{\!\{r \mid r \in \mathcal{R} \ \wedge \ a_1 \leq r \leq a_2\}\!\}\|}{\|\mathcal{R}\|} \leq 1-\alpha \text{ and } \|\{\!\{r \mid r \in \mathcal{R} \ \wedge \ r > a_2\}\!\}\| = 0$$

– We define three notions of *distance* of an observed time value $r \in \mathbb{R}^+$ to an interval $d = [a_1, a_2] \in \mathcal{I}_{\mathbb{R}_+}$, and their generalization to sets of values as

$$\texttt{dist}(r,d) = \begin{cases} 0 & \text{if } \ r \in d \\ r - a_2 & \text{if } \ r > a_2 \\ a_1 - r & \text{if } \ r < a_1 \end{cases} \qquad \texttt{dist}(C,d) = \textstyle\sum_{r \in C} \texttt{dist}(r,d)^2$$

$$\texttt{dist\_up}(r,d) = \begin{cases} 0 & \text{if } \ r \leq a_2 \\ r - a_2 & \text{if } \ r > a_2 \end{cases} \qquad \texttt{dist\_up}(C,d) = \textstyle\sum_{r \in C} \texttt{dist\_up}(r,d)^2$$

$$\texttt{dist\_low}(r,d) = \begin{cases} 0 & \text{if } \ r < a_1 \\ r - a_1 & \text{if } \ r \geq a_1 \end{cases} \qquad \texttt{dist\_low}(C,d) = \textstyle\sum_{r \in C} \texttt{dist\_low}(r,d)^2$$

□

Let us remark that bigger values of α denote smaller *tolerance* to have unexpected values. The first relation, $\subseteq_\alpha$, denotes that the number of values outside the considered interval is not big. There is no distinction between values being smaller/greater than the lower/upper bound of the interval. The second relation, $\preceq_\alpha$, can be used to indicate that we do not allow values greater than the upper bound and that the number of values smaller than the lower bound is acceptable. Finally, $\ll_\alpha$ is useful in situations where most of the values have to be smaller than the lower bound of the interval, while values greater than the upper bound are again not allowed. This last relation will be used to check that the system is fast. The previous relations count the number of errors but do not quantify how big the errors are. Regarding distance functions, they measure the error degree of wrong values. The first one, `dist`, considers both time values greater and smaller than the bounds of the interval. The `dist_up` function considers as wrong only values greater than the upper bound of the interval. Finally, we will use the `dist_low` function for measuring the values that are not fast enough, that is, bigger than the lower bound of the interval. By combining inclusion relations and distance functions, we can evaluate the conformance of the implementation with respect to the specification in different ways.

Definition 8 Let I and S be two `IFSMs`, H be a multiset of timed executions of I, $\Phi = \{e \mid \exists\, t : (e,t) \in H\} \cap \texttt{NTEvol}(S)$, $0 \leq \alpha \leq 1$, and $\beta \in \mathbb{R}^+$. We define the following implementation relations:

– I (H,α)-*timely conforms* to S, denoted by $I\ \mathtt{conf}_{int}^{(H,\alpha)}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all $e \in \Phi$ we have that for all time interval $d \in \mathcal{I}_{\mathbb{R}_+}$

$$(e,d) \in \mathtt{TEvol}(S) \Longrightarrow \mathtt{Obs\_Time}_H(e) \subseteq_\alpha d$$

– I (H,α)-*preferable timely conforms* to S, denoted by $I\,\mathtt{conf}_{intp}^{(H,\alpha)}\,S$, if $I\,\mathtt{conf}_{nt}$ S and for all $e \in \Phi$ we have that for all time interval $d \in \mathcal{I}_{\mathbb{R}_+}$

$$(e,d) \in \mathtt{TEvol}(S) \Longrightarrow \mathtt{Obs\_Time}_H(e) \preceq_\alpha d$$

– I (H,α)-*fast timely conforms* to S, denoted by $I\ \mathtt{conf}_{intf}^{(H,\alpha)}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all $e \in \Phi$ we have that for all time interval $d \in \mathcal{I}_{\mathbb{R}_+}$

$$(e,d) \in \mathtt{TEvol}(S) \Longrightarrow \mathtt{Obs\_Time}_H(e) \ll_\alpha d$$

– I (H,β)-*global timely conforms* to S, denoted by $I\ \mathtt{conf}_{intgb}^{(H,\beta)}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all $e \in \Phi$ we have that for all time interval $d \in \mathcal{I}_{\mathbb{R}_+}$

$$(e,d) \in \mathtt{TEvol}(S) \Longrightarrow \mathtt{dist}(\mathtt{Obs\_Time}_H(e), d) \leq \beta$$

– I (H,β)-*up-timely conforms* to S, denoted by $I\ \mathtt{conf}_{intup}^{(H,\beta)}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all $e \in \Phi$ we have that for all time interval $d \in \mathcal{I}_{\mathbb{R}_+}$

$$(e,d) \in \mathtt{TEvol}(S) \Longrightarrow \mathtt{dist\_up}(\mathtt{Obs\_Time}_H(e), d) \leq \beta$$

– I (H,β)-*low-timely conforms* to S, denoted by $I\ \mathtt{conf}_{intlw}^{(H,\beta)}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all $e \in \Phi$ we have that for all time interval $d \in \mathcal{I}_{\mathbb{R}_+}$

$$(e,d) \in \mathtt{TEvol}(S) \Longrightarrow \mathtt{dist\_low}(\mathtt{Obs\_Time}_H(e), d) \leq \beta$$

– I (H,α,β)-*timely conforms* to S, denoted by $I\ \mathtt{conf}_{int}^{(H,\alpha,\beta)}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all $e \in \Phi$ we have that for all time interval $d \in \mathcal{I}_{\mathbb{R}_+}$

$$(e,d) \in \mathtt{TEvol}(S) \Longrightarrow \big(\mathtt{Obs\_Time}_H(e) \subseteq_\alpha d \ \wedge\ \mathtt{dist}(\mathtt{Obs\_Time}_H(e), d) \leq \beta\big)$$

– I (H,α,β)-*preferable timely conforms* to S, denoted by $I\ \mathtt{conf}_{intp}^{(H,\alpha,\beta)}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all $e \in \Phi$ we have that for all time interval $d \in \mathcal{I}_{\mathbb{R}_+}$

$$(e,d) \in \mathtt{TEvol}(S) \Longrightarrow \begin{pmatrix} \mathtt{Obs\_Time}_H(e) \preceq_\alpha d \\ \wedge \\ \mathtt{dist\_low}(\mathtt{Obs\_Time}_H(e), d) \leq \beta \end{pmatrix}$$

– I (H,α,β)-*fast timely conforms* to S, denoted by $I\,\mathtt{conf}_{intf}^{(H,\alpha,\beta)}\,S$, if $I\,\mathtt{conf}_{nt}\,S$ and for all $e \in \Phi$ we have that for all time interval $d \in \mathcal{I}_{\mathbb{R}_+}$

$$(e,d) \in \mathtt{TEvol}(S) \Longrightarrow \begin{pmatrix} \mathtt{Obs\_Time}_H(e) \ll_\alpha d \\ \wedge \\ \mathtt{dist\_low}(\mathtt{Obs\_Time}_H(e), d) \leq \beta \end{pmatrix}$$

□

Intuitively, the new relations establish that the implementation must conform to the specification in the usual way (that is, $I\ \mathtt{conf}_{nt}\ S$). In addition, the observed execution time values corresponding to an evolution must *mostly* belong to the time interval indicated by the specification for that evolution (*timely conforms*), or be less than or equal to the lower/upper bound (*fast timely conforms/preferable timely conforms*) respectively. The relations *global timely, uptimely, and low-timely* require that the errors presented by the observed execution time values do not exceed a established threshold. Finally, in the last three relations, we consider both requests simultaneously, that is, the relations demand conditions both over the number of observed time values out of the interval and over the allowed deviation.

Let us remark that to have the previously defined relations parameterized by the set H is somehow similar to consider the, widely used, *fairness assumption* in formal testing: If we test a system enough, we can be sure that we go through all the possible paths of the tested machine. In the case of the fairness assumption, we would have something like $H = \mathtt{TEvol}(I)$ while in our setting we have that an implementation is correct up to the submultiset of $\mathtt{TEvol}(I)$ that we consider.

4 Definition and Application of Tests

A test represents a sequence of inputs applied to the implementation. After applying each input, we check whether the received output is expected or not. In the latter case, a fail signal is produced. In the former case, either a pass signal is emitted (indicating successful termination) or the testing process continues by applying another input. If we are testing an implementation with input and output sets I and O, respectively, tests are deterministic acyclic I/O labelled transition systems (i.e. trees) with a strict alternation between an input action and the set of output actions. After an output action we may find either a leaf (indicating either failure or successful termination) or another input action. Leaves are labelled either by *pass* or by *fail*. In the first case we add a *time stamp*. The time stamp will be a time interval. The idea is that we will record the time that the implementation takes to arrive to that point and compare it with the time stamp.

Definition 9 A *test* is a tuple $T = (S, I, O, Tr, s_0, S_I, S_O, S_F, S_P, C_T)$ where S is the set of states, I and O are disjoint sets of input and output actions, respectively, $Tr \subseteq S \times (I \cup O) \times S$ is the transition relation, $s_0 \in S$ is the initial state, and the sets $S_I, S_O, S_F, S_P \subseteq S$ are a partition of S. The transition relation and the sets of states fulfill the following conditions:

- S_I is the set of *input* states. We have that $s_0 \in S_I$. For all input state $s \in S_I$ there exists a unique outgoing transition $(s, a, s') \in Tr$. For this transition we have that $a \in I$ and $s' \in S_O$.

- S_O is the set of *output* states. For all output state $s \in S_O$ we have that for all $o \in O$ there exists a unique state s' such that $(s, o, s') \in Tr$. In this case, $s' \notin S_O$. Moreover, there do not exist $i \in I$ and $s' \in S$ such that $(s, i, s') \in Tr$.
- S_F and S_P are the sets of *fail* and *pass* states, respectively. We say that these states are *terminal*. That is, for all state $s \in S_F \cup S_P$ we have that there do not exist $a \in I \cup O$ and $s' \in S$ such that $(s, a, s') \in Tr$.

Finally, $C_T : S_P \longrightarrow \mathcal{I}_{\mathbb{R}_+}$ is a function associating time stamps, that is, a time intervals, with passing states.

Let $e = i_1/o_1, \ldots, i_r/o_r$. We write $T \stackrel{e}{\Longrightarrow} s$ if $s \in S_F \cup S_P$ and there exist states $s_{12}, s_{21}, s_{22}, \ldots s_{r1}, s_{r2} \in S$ such that $\{(s_0, i_1, s_{12}), (s_{r2}, o_r, s)\} \subseteq Tr$, for all $2 \leq j \leq r$ we have $(s_{j1}, i_j, s_{j2}) \in Tr$, and for all $1 \leq j \leq r-1$ we have $(s_{j2}, o_j, s_{(j+1)1}) \in Tr$.

We say that a test case T is *valid* if the graph induced by T is a tree with root at the initial state s_0. We say that a set of tests $\mathcal{T} = \{T_1, \ldots, T_n\}$ is a *test suite*. □

From now on we will assume that when we talk about tests we refer only to valid tests. Next we define the application of a test to an implementation. We will say that the test suite $\mathcal{T}$ is *passed* if, for all test, the terminal states reached by the composition of implementation and test belong to the set of *passing* states. Let us remark that since we are assuming that implementations are input-enabled, the testing process will conclude only when the test reaches either a fail or a success state.

Definition 10 Let I be an implementation under test and T be a test. We denote the application of the test T to the implementation I by $I \parallel T$.

Let I be a `IFSM`, T be a test, and s be a state of T. We write $I \parallel T \stackrel{e}{\Longrightarrow} s$ if $T \stackrel{e}{\Longrightarrow} s$ and $e \in \texttt{NTEvol}(I)$.

We say that I *passes* the test suite $\mathcal{T}$, denoted by $\texttt{pass}(I, \mathcal{T})$, if for all test $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C_T) \in \mathcal{T}$ and $e \in \texttt{NTEvol}(I)$ there do not exist $s \in S_F$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s$. □

The previous definition of passing tests did not take into account the time values that will be collected during the application of tests. We apply time conditions to the set of *observed timed executions*. In fact, we need a set of test executions associated to each evolution in order to evaluate if they match, in a certain sense, the time interval associated to the corresponding state of the test. In order to increase the reliability degree, we will not take the classical approach where passing a test suite is defined according only to the results for each test. In our approach, we will put together all the observations, for each test, so that we have more samples for each evolution. In particular, some observations will be used several times. In other words, an observation from a given test may be used to check the validity of another test sharing the same observed sequence.

Definition 11 Let I be an $\texttt{IFSM}$, T be a test, and s be a state of T. We write $I \parallel T \stackrel{e}{\Longrightarrow}_t s$ if $T \stackrel{e}{\Longrightarrow} s$ and (e,t) is an observed timed execution of I. In this case we say that (e,t) is a *test execution* of I and T. Let I be an $\texttt{IFSM}$ and $\mathcal{T} = \{T_1, \ldots, T_n\}$ be a test suite. Let $H_1, \ldots, H_n$ be sets of test executions of I and $T_1, \ldots, T_n$, respectively. Let $H = \bigcup_{i=1}^{n} H_i$, $\Phi = \{e \mid \exists\ t : (e,t) \in H\}$, $\beta \in \mathbb{R}^+$, and $0 \leq \alpha \leq 1$. We say that

- I (H,α)-*timely passes* the test suite $\mathcal{T}$ if $\texttt{pass}(I,\mathcal{T})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s$, we have that
$$\texttt{Obs\_Time}_H(e) \subseteq_\alpha C_T(s)$$
- I (H,α)-*preferable passes* the test suite $\mathcal{T}$ if $\texttt{pass}(I,\mathcal{T})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s$, we have that
$$\texttt{Obs\_Time}_H(e) \preceq_\alpha C_T(s)$$
- I (H,α)-*fast passes* the test suite $\mathcal{T}$ if $\texttt{pass}(I,\mathcal{T})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s$, we have that
$$\texttt{Obs\_Time}_H(e) \ll_\alpha C_T(s)$$
- I (H,β)-*global timely passes* the test suite $\mathcal{T}$ if $\texttt{pass}(I,\mathcal{T})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s$, we have that
$$\texttt{dist}(\texttt{Obs\_Time}_H(e), C_T(s)) \leq \beta$$
- I (H,β)-*up-timely passes* the test suite $\mathcal{T}$ if $\texttt{pass}(I,\mathcal{T})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s$, we have that
$$\texttt{dist\_up}(\texttt{Obs\_Time}_H(e), C_T(s)) \leq \beta$$
- I (H,β)-*low-timely passes* the test suite $\mathcal{T}$ if $\texttt{pass}(I,\mathcal{T})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s$, we have that
$$\texttt{dist\_low}(\texttt{Obs\_Time}_H(e), C_T(s)) \leq \beta$$
- I (H,α,β)-*timely passes* the test suite $\mathcal{T}$ if $\texttt{pass}(I,\mathcal{T})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s$, we have that
$$\texttt{Obs\_Time}_H(e) \subseteq_\alpha C_T(s) \ \wedge\ \texttt{dist}(\texttt{Obs\_Time}_H(e), C_T(s)) \leq \beta$$
- I (H,α,β)-*preferable passes* the test suite $\mathcal{T}$ if $\texttt{pass}(I,\mathcal{T})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s$, we have that
$$\texttt{Obs\_Time}_H(e) \preceq_\alpha C_T(s) \ \wedge\ \texttt{dist\_low}(\texttt{Obs\_Time}_H(e), C_T(s)) \leq \beta$$
- I (H,α,β)-*fast passes* the test suite $\mathcal{T}$ if $\texttt{pass}(I,\mathcal{T})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s$, we have that
$$\texttt{Obs\_Time}_H(e) \ll_\alpha C_T(s) \ \wedge\ \texttt{dist\_low}(\texttt{Obs\_Time}_H(e), C_T(s)) \leq \beta$$

□

Let us remark that an observed timed execution does not return the time interval associated with performing the evolution (that is, the addition of all the intervals corresponding to each transition of the implementation) but the time that it took to perform the evolution. Let us also note that in a fix time values framework, these two notions (addition of time values corresponding to the transitions of the implementation and observed time) do in fact coincide.

Intuitively, an implementation passes a test if there does not exist an evolution leading to a fail state. Once we know that the functional behavior of the implementation is correct with respect to the test, we need to check time conditions. The set H corresponds to the observations of the (several) applications of the tests belonging to the test suite $\mathcal{T}$ to I. Thus, we have to decide whether, for each evolution e, the observed time values (that is, $\texttt{Obs\_Time}_H(e)$) *match* the definition of the time intervals appearing in the successful state of the tests corresponding to the execution of that evolution (that is, $C_T(s)$).

Due to space limitations, we cannot include in this paper the algorithm that we propose to derive tests from a specification. In spite of the differences, our algorithm is an adaptation of that in [32]. We get a test suite extracted from the specification S. We denote this test suite by $\texttt{tests}(S)$.

Next, we present a result to establish the application of the test suite $\texttt{tests}(S)$ for determining whether an implementation, for a sample H, conforms to a specification with respect to the relations given in Definition 8.

Theorem 1 (Soundness and Completeness) Let I and S be IFSMs. Given a multiset of timed executions H, $\beta \in \mathbb{R}_+$, and $0 \leq \alpha \leq 1$ we have

- $I\ \texttt{conf}_{int}^{(H,\alpha)}\ S$ iff I (H,α)-*timely passes* $\texttt{tests}(S)$.
- $I\ \texttt{conf}_{intf}^{(H,\alpha)}\ S$ iff I (H,α)-*fast passes* $\texttt{tests}(S)$.
- $I\ \texttt{conf}_{intp}^{(H,\alpha)}\ S$ iff I (H,α)-*preferable passes* $\texttt{tests}(S)$.
- $I\ \texttt{conf}_{intgb}^{(H,\beta)}\ S$ iff I (H,β)-*global timely passes* $\texttt{tests}(S)$.
- $I\ \texttt{conf}_{intup}^{(H,\beta)}\ S$ iff I (H,β)-*up-timely passes* $\texttt{tests}(S)$.
- $I\ \texttt{conf}_{intlw}^{(H,\beta)}\ S$ iff I (H,β)-*low-timely passes* $\texttt{tests}(S)$.
- $I\ \texttt{conf}_{int}^{(H,\alpha,\beta)}\ S$ iff I (H,α,β)-*timely passes* $\texttt{tests}(S)$.
- $I\ \texttt{conf}_{intf}^{(H,\alpha,\beta)}\ S$ iff I (H,α,β)-*fast passes* $\texttt{tests}(S)$.
- $I\ \texttt{conf}_{intp}^{(H,\alpha,\beta)}\ S$ iff I (H,α,β)-*preferable passes* $\texttt{tests}(S)$.

□

5 Conclusions and Future Work

In this paper we have presented a novel framework to specify and test timed systems showing both soft and hard deadlines. We have defined nine conformance relations that take into account the different considerations of what a *slightly*

erroneous system is, that is, that soft deadlines are *almost* always met. We have also developed a testing theory by introducing a notion of test and by defining how tests are applied to implementations and what is the meaning of passing a test. Finally, we have stated that testing a system with the appropriate test suite is equivalent to establish that it is related with the specification from which the test suite was extracted.

There is still some room for future work. First, it would be interesting to study the precise relation between the different implementation relations that we define in this paper. Second, we would like to take this paper as a first step, together with [33], to define a testing theory for systems presenting both time and probabilistic information expressed by means of intervals.

References

1. Sifakis, J.: Use of Petri nets for performance evaluation. In: 3rd Int. Symposium on Measuring, Modelling and Evaluating Computer Systems, North-Holland (1977) 75–93
2. Zuberek, W.: Timed Petri nets and preliminary performance evaluation. In: 7th Annual Symposium on Computer Architecture, ACM Press (1980) 88–96
3. Reed, G., Roscoe, A.: A timed model for communicating sequential processes. Theoretical Computer Science **58** (1988) 249–261
4. Nicollin, X., Sifakis, J.: An overview and synthesis on timed process algebras. In: 3rd Int. Conf. on Computer Aided Verification, CAV'91, LNCS 575, Springer (1991) 376–398
5. Glabbeek, R.v., Smolka, S., Steffen, B.: Reactive, generative and stratified models of probabilistic processes. Information and Computation **121**(1) (1995) 59–80
6. Baeten, J., Middelburg, C.: Process algebra with timing. EATCS Monograph. Springer (2002)
7. Bravetti, M., Aldini, A.: Discrete time generative-reactive probabilistic processes with different advancing speeds. Theoretical Computer Science **290**(1) (2003) 355–406
8. Núñez, M.: Algebraic theory of probabilistic processes. Journal of Logic and Algebraic Programming **56**(1–2) (2003) 117–177
9. Higashino, T., Nakata, A., Taniguchi, K., Cavalli, A.: Generating test cases for a timed I/O automaton model. In: 12th Int. Workshop on Testing of Communicating Systems, IWTCS'99, Kluwer Academic Publishers (1999) 197–214
10. Springintveld, J., Vaandrager, F., D'Argenio, P.: Testing timed automata. Theoretical Computer Science **254**(1-2) (2001) 225–257 Previously appeared as Technical Report CTIT-97-17, University of Twente, 1997.
11. Fecko, M., Uyar, M., Duale, A., Amer, P.: A technique to generate feasible tests for communications systems with multiple timers. IEEE/ACM Transactions on Networking **11**(5) (2003) 796–809
12. En-Nouaary, A., Dssouli, R.: A guided method for testing timed input output automata. In: 15th Int. Conf. on Testing Communicating Systems, TestCom'03, LNCS 2644, Springer (2003) 211–225
13. Núñez, M., Rodríguez, I.: Towards testing stochastic timed systems. In: 23rd IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'03, LNCS 2767, Springer (2003) 335–350

14. Stoelinga, M., Vaandrager, F.: A testing scenario for probabilistic automata. In: 30th Int. Colloquium on Automata, Languages and Programming, ICALP'03, LNCS 2719, Springer (2003) 464–477
15. Brandán Briones, L., Brinksma, E.: Testing real-time multi input-output systems. In: 7th Int. Conf. on Formal Engineering Methods, ICFEM'05, LNCS 3785, Springer (2005) 264–279
16. Merayo, M., Núñez, M., Rodríguez, I.: Extending EFSMs to specify and test timed systems with action durations and timeouts. In: 26th IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'06, LNCS 4229, Springer (2006) 372–387
17. Götz, N., Herzog, U., Rettelbach, M.: Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. In: 16th Int. Symp. on Computer Performance Modelling, Measurement and Evaluation, PERFORMANCE'93, LNCS 729, Springer (1993) 121–146
18. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)
19. Bernardo, M., Gorrieri, R.: A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. Theoretical Computer Science **202**(1-2) (1998) 1–54
20. Harrison, P., Strulo, B.: SPADES – a process algebra for discrete event simulation. Journal of Logic Computation **10**(1) (2000) 3–42
21. Hermanns, H., Herzog, U., Katoen, J.P.: Process algebra for performance evaluation. Theoretical Computer Science **274**(1-2) (2002) 43–87
22. Bravetti, M., Gorrieri, R.: The theory of interactive generalized semi-Markov processes. Theoretical Computer Science **282**(1) (2002) 5–32
23. López, N., Núñez, M., Rubio, F.: An integrated framework for the analysis of asynchronous communicating stochastic processes. Formal Aspects of Computing **16**(3) (2004) 238–262
24. Cleaveland, R., Lee, I., Lewis, P., Smolka, S.: A theory of testing for soft real-time processes. In: 8th Int. Conf. on Software Engineering and Knowledge Engineering, SEKE'96. (1996) 474–479
25. de Nicola, R., Hennessy, M.: Testing equivalences for processes. Theoretical Computer Science **34** (1984) 83–133
26. Hennessy, M.: Algebraic Theory of Processes. MIT Press (1988)
27. Yuen, S., Cleaveland, R., Dayar, Z., Smolka, S.: Fully abstract characterizations of testing preorders for probabilistic processes. In: 5th Int. Conf. on Concurrency Theory, CONCUR'94, LNCS 836, Springer (1994) 497–512
28. Bernardo, M., Cleaveland, W.: A theory of testing for markovian processes. In: 11th Int. Conf. on Concurrency Theory, CONCUR'2000, LNCS 1877, Springer (2000) 305–319
29. López, N., Núñez, M.: A testing theory for generally distributed stochastic processes. In: 12th Int. Conf. on Concurrency Theory, CONCUR'01, LNCS 2154, Springer (2001) 321–335
30. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. Software – Concepts and Tools **17**(3) (1996) 103–120
31. Núñez, M., Rodríguez, I.: Encoding `PAMR` into (timed) `EFSMs`. In: 22nd IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'02, LNCS 2529, Springer (2002) 1–16

32. Núñez, M., Rodríguez, I.: Conformance testing relations for timed systems. In: 5th Int. Workshop on Formal Approaches to Software Testing, FATES'05, LNCS 3997, Springer (2006) 103–117
33. López, N., Núñez, M., Rodríguez, I.: Specification, testing and implementation relations for symbolic-probabilistic systems. Theoretical Computer Science **353**(1–3) (2006) 228–248