Advantages of mutation in passive testing: An empirical study. *

César Andrés, Mercedes G. Merayo, Carlos Molinero
Deptartamento Sistemas Informáticos y Computación
Universidad Complutense de Madrid
E-28040 Madrid. Spain.
{c.andres,mgmerayo,molinero}@fdi.ucm.es

Abstract

This paper presents an empirical study of the mutation techniques used by the tool PASTE. *This tool allows the automation of our passive testing methodology for systems that present stochastic-time information. In our proposal, invariants express the fact that each time the implementation under test performs a given sequence of actions, then it must exhibit a behavior according to the probability distribution functions reflected in the invariant. We briefly review the theoretical framework of our methodology and the main features of our tool. Next, we present in detail the* `Mutants module` *that provides us with a functionality to test the effectiveness of invariants for detecting errors. Finally, we present a study of the results obtained from the performed experiments.*

1 Introduction

Formal testing techniques [6, 18, 16, 24] provide systematic procedures to check implementations with respect to a specification in such a way that the coverage of critical parts/aspects of the system under test is increased. Initially, these techniques focused on 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. Nevertheless, there exist many systems where non-functional aspects, such as the probability of an event to happen, the time that it takes to perform a certain action, or the time when a certain action happens, make the difference between correct and incorrect behaviors. During the last two decades there has been a lot of interest in extending formal techniques to cope with time. Even though there exist several proposals for timed testing [19, 10, 14, 25, 11, 22, 20], most of them specialize in active testing, assuming that the tester is allowed to interact directly with the system. However, sometimes testers cannot interact with the system. In those situations, testing methodologies make use of *monitoring* techniques, that extract traces from the implementation that can be analyzed in order to detect errors. This paradigm is called *passive testing*. There are several proposals for formal passive testing [4, 5, 17], but they focuses on checking wether the systems satisfy functional properties, such as "after the input i the systems always emits the output o". Our work is based on [4, 5]. In this approach, a set of properties, called *invariants*, are checked against the traces observed from the implementation to test their correctness. We extended this idea in [2] by adding stochastic time requirements in the specification and in the invariants. There, we present algorithms to verify the time-correctness of the invariants with respect to the specification and to assess the time-correctness of the recorder traces with respect to the invariants. We present a formalism based on *finite state machines*, allowing to take into account temporal aspects. The time is represented by means of probability distribution functions, it allows that instead of having expressions such as "the action o takes t time units to be performed" we will have expressions such as "with probability p the action o will be performed before t time units".

We have developed a tool, called PASTE, where the algorithms proposed for checking the correctness of the invariants and the recorder traces have been implemented. In addition to the automation of our methodology, the tool provides the tester with a functionality that allows to check the effectiveness of invariants to detect errors, by applying them to *simulated* traces. In order to generate these traces we have applied a methodology based on *mutation testing* [23, 12, 26, 21, 13]. Originally, mutation testing was applied to code [15, 7] but some work has looked at *specification mutation* [8]. Here, the specification is mutated and for each *mutant* a test is derived that distinguishes the behaviours of the mutated and original specifications. Most work on mutating specifications has used either finite

*Research supported by the Spanish MEC project WEST/FAST (TIN2006-15578-C02-01).

978-0-7695-3671-2/09 $25.00 © 2009 IEEE
DOI 10.1109/ICSTW.2009.33

state machines or extended finite state machines, but there are not many proposals to mutate non-functional properties (see [13] for such a proposal). Another use of mutation testing is the evaluation of other test techniques [3]. In particular, in order to evaluate a test *technique* we can produce test suites using this technique and estimate their effectiveness by determining what proportion of mutants they kill. In this paper we present the mutants module of our tool where the specification is mutated and traces are extracted from these mutants for evaluating a measure of the quality of the invariants proposed for detecting errors. These traces are chosen in order to simulate *real faults*. Thus, if we have an invariant that can find an error in the trace recorded from the mutant, then this invariant has more probability to find an error in a faulty implementation. We describe the different *mutation operators* that can be applied to the specification. In addition, we present approaches for measuring the level of detection of the invariants and analyze the empirical results obtained from different experiments.

The structure of this paper is as follows. In the next section we introduce our framework to perform passive testing in systems with stochastic time restrictions. In Section 3 we present the tool PASTE. Section 4 explains how the mutants module works and we introduce the mutation operators and the generation of simulated traces. In Section 5 we present the empirical result obtained from the experiments performed in PASTE. Finally, in Section 6 we present our conclusions and some lines for future work.

2 The theoretical framework

In this Section we remind the main aspects of the framework to perform passive testing in systems with temporal restrictions introduced in [1, 2]. First, we present the formalism that we use in order to represent systems with temporal restrictions. This formalism is an extension of the classical finite state machine (FSM). The main difference with respect to usual FSMs consists in the addition of *time* to indicate the lapse between offering an input and receiving an output. We use *probability distribution functions* to model this time. Essentially, these functions provide us with the probability that the system performs an action before an amount of time. Next, we introduce some basic concepts on probability distribution functions.

Definition 1 A *probability distribution function* is a function $F : \mathbf{R}_+ \longrightarrow [0, 1]$, having the following properties:

- $\lim_{t\to+\infty} F(t) = 1$.
- F is monotonically increasing, that is, for all $t_1, t_2 \in \mathbf{R}_+$ such that $t_1 \leq t_2$ we have $F(t_1) \leq F(t_2)$.
- F is a right-continuous function at any point, that is, for all $t \in \mathbf{R}_+$ we have:

$$\lim_{t'\to t^+} F(t') = F(t).$$

We denote the set of probability distribution functions by $\mathcal{F}$ (F, F_1, F_2 to range over $\mathcal{F}$). Let F_1 and F_2 be two probability distribution functions. We write $F_1 = F_2$ if for all $t \in \mathbf{R}_+$ we have $F_1(t) = F_2(t)$. We will call *sample* to any multiset of positive real numbers.

Let F be a probability distribution function and J be a sample. We denote the *confidence* of F on J by $\gamma(F, J)$. In our setting, samples will be associated with time values that implementations need to perform sequences of actions. We have that $\gamma(F, J)$ takes values in the interval $[0, 1]$. □

Intuitively, bigger values of $\gamma(F, J)$ indicate that the observed sample J is more likely to be produced by the probability distributed function F. That is, γ decides how *similar* is the probability distribution function generated by J and the one corresponding to F are.

Next, we briefly mention some well-known probability distribution functions families that we consider along the paper. They are classified in two types: continuous (*uniform*, and *exponential*) and discrete (*binomial*, *discrete* and *dirac*) probability distribution functions. For example, let us consider the following continuos distribution:

$$F_1(t) = \begin{cases} 0 & \text{if } t \leq \alpha \\ \frac{t-\alpha}{\beta-\alpha} & \text{if } \alpha \leq t < \beta \\ 1 & \text{if } \beta \leq t \end{cases}$$

We say that F_1 is uniformly distributed in the interval $[\alpha, \beta]$. Uniform distributions allow us to keep compatibility with time intervals in timed models in the sense that the same *weight* is assigned to all the times in the interval. The next function, F_2, follows a Dirac distribution in α.

$$F_2(t) = \begin{cases} 0 & \text{if } 0 \leq t < \alpha \\ 1 & \text{if } a \leq t \end{cases}$$

The idea is that the corresponding delay will be equal to α time units. Dirac distributions allow us to simulate deterministic delays appearing in timed models.

Definition 2 A *Timed Finite State Machine*, in short TFSM, is a tuple $X = (\mathcal{S}, I, O, \rightarrow, s_0)$ where $\mathcal{S}$ is a finite set of states, I is the set of input actions, O is the set of output actions, $\rightarrow$ is the set of transitions, and s_0 is the initial state.

A transition belonging to $\rightarrow$ is a tuple (s, s', i, o, F) where $s, s' \in \mathcal{S}$ are the initial and final states of the transition, $i \in I$ and $o \in O$ are the input and output actions, respectively, and $F \in \mathcal{F}$ denotes the time, in probability terms, that the transition needs to be completed.

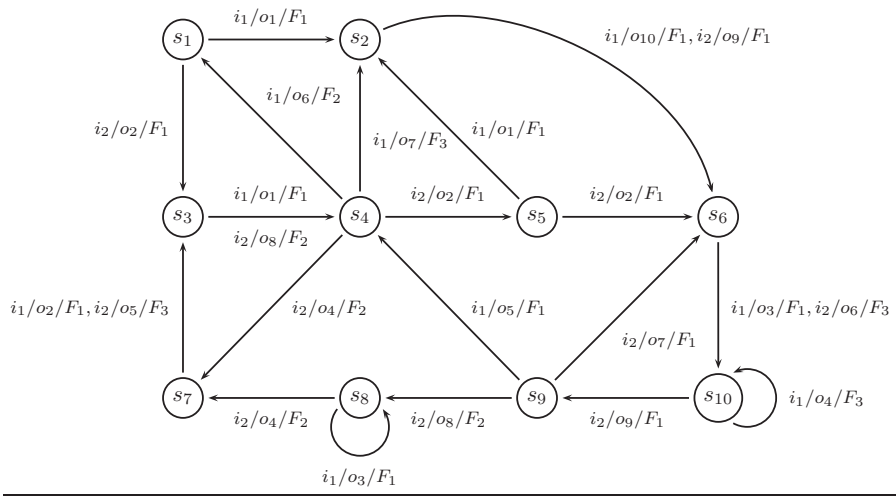

Figure 1. Example of TFSM.

We say that X is *input-enabled* if for all state $s \in \mathcal{S}$ and input $i \in I$, there exist $s' \in \mathcal{S}$, $o \in O$, and $F \in \mathcal{F}$ such that $(s, s', i, o, F) \in \rightarrow$. We say that X is *deterministically observable* if for all s, i, o there do not exist two different transitions $(s, i, o, F, s_1), (s, i, o, F, s_2) \in \delta$. We say that X has *regular stochastic information*, if there do not exist two different transitions $tr_1 = (s'_1, s''_1, i, o, F_1)$ and $tr_2 = (s'_2, s''_2, i, o, F_2)$ with $F_1 \neq F_2$. □

In our framework, we assume that we are provided with a complete input-enabled specification, and it has regular stochastic information. The idea is that actions that produce the same output may be related with respect the amount of time that they need to be performed. In addition, implementations and specifications are deterministically observable.

Intuitively, a transition (s, s', i, o, F) indicates that if the machine is in state s and receives the input i then the machine emits the output o before t time units with probability $F(t)$ and the machine changes its current state to s'.

Example 1 Let us consider the machine X depicted in Figure 1. It represents the specification of a system by using a TFSM model. Each transition has an associated probability distribution function. It is worth to note that in the transitions set, all transitions that share the same pair input-output have associated the same stochastic distribution function. The function F_1 corresponds to a continuous uniform distribution in the interval $[0, 2]$, the function F_2 follows a Dirac distribution in 4, and F_3 belongs to the family of exponential distribution functions. For example, if X is in the state s_3 and receives the input action i_2, then X triggers the transition $(s_3, s_4, i_2, o_8, F_2)$ and emits the output o_8 after a time given by F_2, in this case 4 units of time. □

Next, we introduce the notion of *time invariant*. The time invariants allow us to express temporal properties that must be fulfilled by the implementation. For example, we could express that the time the system takes to perform a transition always belongs to a specific interval having all values the same probability. We could consider that the invariants are extracted from the specification. In fact, we can do this easily by adapting the method given in [9] to our timed framework. However, this leads to a huge set of invariants, where not all of them are relevant. In our approach we assume that time invariants are given by the tester from the original requirements (for more details [5, 1]). Consequently, we need to check that the time invariants presented by the tester are correct with respect to the specification. Once we have a collection of correct time invariants, we will have to control if these invariants are satisfied by the timed-traces produced by the implementation. A trace represents a sequence of actions that the system may perform from its origin state. We provide in [1, 2] algorithms to verify the correctness of the invariants with respect to the specification, and the traces with respect to the invariants.

Definition 3 Let $X = (\mathcal{S}, I, O, \rightarrow, s_0)$ be a TFSM. We say that a sequence ϕ is a *time invariant* for X if the following two conditions hold:

1. ϕ is defined according to the following EBNF:

$$\phi ::= a/z/F, \phi \mid \star, \phi' \mid i \mapsto \mathcal{C}$$
$$\phi' ::= i/z/F, \phi \mid i \mapsto \mathcal{C}$$

   In this expression we consider $F \in \mathcal{F}$, $i \in I$, $a \in I \cup \{?\}$, $z \in O \cup \{?\}$, and $\mathcal{C} \subseteq O \times \mathcal{F}$.

2. ϕ is *correct* [1] with respect to X.

□

In order to express traces in a concise way, we will use the wild-card characters ? and $\star$. The wild-car ? represents

any value in the sets I and O, while $\star$ represents a sequence of input/output pairs.

Intuitively, the previous EBNF expresses that an invariant is either a sequence of symbols where each component, but the last one, is either an expression $a/z/F$, with a being an input action or the wild-card character ?, z being an output action or the wild-card character ?, and F being a probability distribution function, or an expression $\star$. There are two restrictions to this rule. First, an invariant cannot contain two consecutive expressions $\star$ and $\star$. In the case that such situation was needed to represent a property, the tester could simulate it by means of the expression $\star$. The second restriction is that an invariant cannot present a component of the form $\star$ followed by an expression beginning with the wildcard character ?, that is, the input of the next component must be a *real* input action $i \in \mathcal{I}$. In fact, $\star$ represents any sequence of inputs/outputs pairs such that the input is not equal to i. The last component, called `head` of the invariant, corresponding to the expression $i \mapsto \mathcal{C}$, is an input action followed by a set of pairs that associate an output with a probability distribution function. For example, the semantic of an invariant $i \mapsto \{\langle o, F\rangle\}$ is that if we observe the input i, obligatory, we will observe the output o, in a lapse of time that fits the function F. This a basic invariant, but we can build one more complex with wildcards: $i/o/F, \star, i' \mapsto \{\langle o, F_1\rangle, \langle o', F_2\rangle\}$. The meaning of this invariant is that if we observe the transition i/o in a time that can be generated by the function F, then the first occurrence of the input symbol i', after a lapse of any amount of time, must be followed by the output o or o' having associated a time from F_1 and F_2 respectively.

Given a set of time invariants and before checking them against the traces obtained from the implementation, they must be checked against the specification, in order to avoid that an invariant violates the requirements expressed in the specification. Once we have a set of *correct* invariants, we use them for checking the traces obtained from the implementation. If the time invariant detects a mismatch, then the implementation that has generated this trace is incorrect with respect to the specification.

In order to check the correctness of a trace with respect to an invariant, first, we review the non-temporal behaviours. Intuitively, an invariant will detect an error if there exists a subsequence of the trace that does not match the invariant. For example, if we consider the invariant $i/o/F, \star, i' \mapsto \{\langle o, F_1\rangle, \langle o', F_2\rangle\}$, each time we find in the trace a subsequence starting with the input i paired with the output symbol o, the first occurrence of the input i' should be paired with either the output o or the output o'. Nevertheless, if we do not find such subsequence in the trace, that is, there is no occurrence of the pair i/o, we cannot emit a verdict. Once we confirm that a trace fulfills these requirements, we will have to check if the temporal behaviour satisfies the constraints expressed in the invariant by means of the associated probability distribution functions. In order to establish if the amounts of time that the implementation takes to perform the actions *fits* the probability distribution functions, we need to collect samples that we can compare with it using a hypothesis contrast. Each sample will be associated with a pair input/output. These samples need to be compared with the probabilistic distribution functions associated to the corresponding pair. This comparison is based on statistical results. In our tool, we implement the Chi-Square Goodness-of-Fit Test. This test has the form:

$$\sum \frac{(\text{observed} - \text{expected})^2}{\text{expected}}$$

This test provides us with a grade of confidence relative to the probability of obtaining a sample from the distribution function.

3 PASTE: A Tool PASive TEsting

In addition to the theoretical framework we have developed a tool called PASsive TEsting (PASTE) which implements the algorithms defined for our framework and helps to automate of our passive testing approaches. In Figure 2 we present a graphical representation of the kernel modules and the relationships among them. The input data are stored in XML format. They include information regarding to the specification of the system, described by means of the `TFSM` formalism, the invariants that we consider, and the traces obtained from the implementation under test.

After loading the input data, the tool allows us to verify the correctness of the invariants with respect to the specification. When we have a set of correct invariants, we will check the correctness of the traces obtained from the implementation under test with respect to them. Finally, the tool will provide us with statistical results about the number of errors found, the invariant that has revealed more faults, etc.

We detected that the order the invariants are applied could be a decisive factor for this task, that is, some invariants have more power than others for detecting faults. So, it would be an advantage if we could select the *best set of invariants*. By *best* we mean that the application of this set of invariants to the traces in order to check their correctness is more efficient revealing errors. Thus, we need to be able to establish the level of detection of faults of the possible invariants for comparing them and choose the best ones. With this goal, we decided to test the invariants with respect to a specific set of traces that could present errors and compare the results. In this way, a tester has the possibility of known how good is an invariant for an specification. This fact caused the integration of a new module in PASTE that

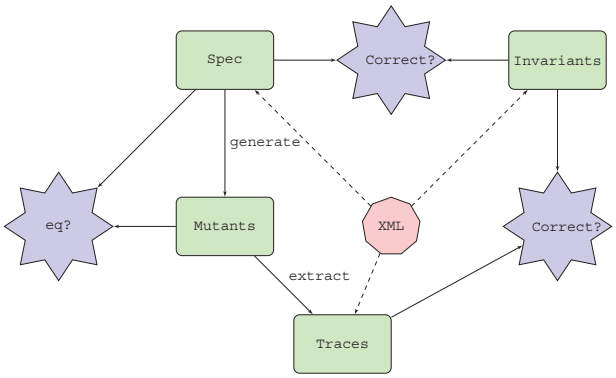

Figure 2. Core of PASTE**.**

helps us to produce traces from *simulated erroneous* implementations that we will use for obtaining the level of quality of the invariants. Next, we introduce *Mutants module* and present its functional aspects.

4 Mutants module

The main goal of this module is to generate a set of traces in order to use them for testing the power of detection of errors of the different invariants considered by the tester. In order to do it, we apply a methodology based on mutation testing. The specification is mutated in order to simulate real faults and, from each *mutant*, a set of traces is generated. These traces will be checked with the set of available invariants in order to determine , based on the obtained results, their level of detection of faults. We describe three *mutation operators* that will be applied to a `TFSM` specification. We only consider *first order mutant*, that is, mutants obtained by the application of one mutation operator.

When we generate a mutant it may be functionally *equivalent* to the original specification. In order to establish the effectiveness of the invariants detecting faults we will not consider the equivalent mutants. It is worth to remark that we are interested in checking the power of the invariants to detect faults. If we would use equivalent mutants, no error would appear in the traces generated from them, so it would not be useful for our goal. The notion of equivalence of a mutant with respect to the specification follows a criterium of *inclusion of traces*: A mutant is *equivalent* to the original specification if for all possible sequences of inputs that can be performed by the mutant and the specification the outputs produced by the mutant are a subset of those considered by the specification. Intuitively, the mutant should not *invent* behaviors when provided with inputs specified in the specification. This pattern is borrowed from `ioco` [27]. Following, we formally define the mutation operators.

4.1 Mutation operators

In PASTE we consider three possible mutation operators: *Changing the Goal State of a transition*(CGS), *Changing Output*(CO), and *Changing Time*(CT). The first one corresponds to create a mutant of an specification by changing the goal state of a transition.

Definition 4 Let $X = (\mathcal{S}, I, O, \rightarrow, s_0)$ be a specification, $tc = (s, s', i, o, F) \in \rightarrow$ a transition, and $s'' \in \mathcal{S}$ with $s' \neq s''$. We denote by $M_{cgs(tc,s'')} = (\mathcal{S}, I, O, \rightarrow', s_0)$ a new `TFSM` where

$$\rightarrow' = \{tr | tr \in \rightarrow \wedge\ tr \neq tc\} \cup \{(s, s'', i, o, F)\}.$$

□

The next mutation operator creates a mutant changing the output associated to a transition in the specification. Naturally, we must change the *probability distribution function* so that the mutant has regular stochastic information.

Definition 5 Let $X = (\mathcal{S}, I, O, \rightarrow, s_0)$ be a specification, $tc = (s, s', i, o, F) \in \rightarrow$ a transition, and $o' \in O$ such that $o' \neq o$ and there does not exist a transition $tc' = (s, s'', i, o', F') \in \rightarrow$. We denote by $M_{co(tc,o')} = (\mathcal{S}, I, O, \rightarrow', s_0)$ a new `TFSM` where

$$\rightarrow' = \{tr | tr \in \rightarrow \wedge\ tr \neq tc\} \cup \{(s, s', i, o', F')\}.$$

with

$$F' = \begin{cases} F & \text{if} \quad \nexists (s_i, s_f, i, o', F_t) \quad \in \rightarrow \\ F_t & \text{if} \quad \exists (s_i, s_f, i, o', F_t) \quad \in \rightarrow \end{cases}$$

where $s_i, s_f \in S$ and $F_t \in \mathcal{F}$. □

The condition "there does not exist a transition $tc' = (s, s'', i, o', F') \in \rightarrow$" is required in order to obtain a mutant that simulates an implementation according to our assumptions, that is, the implementation must be deterministically observable. Thus we do not accept to change an output that generates a transition with the same input/output actions that other transition outgoing from the same state.

The last mutation operator alters the probability distribution function associated to a pair of input/output actions.

Definition 6 Let $X = (\mathcal{S}, I, O, \rightarrow, s_0)$ be a specification, $tc = (s, s', i, o, F) \in \rightarrow$ a transition, and $F' \in \mathcal{F}$ with $F' \neq F$. We denote by $M_{ct(tc,F')} = (\mathcal{S}, I, O, \rightarrow', s_0)$ a new `TFSM` where

$$\rightarrow' \quad = \quad \left\{ \begin{array}{ll} (s, s', i', o', F) | & (s, s', i', o', F) \in \rightarrow \\ & \wedge \\ & (i \neq i' \vee o \neq o') \end{array} \right\}$$

$$\cup$$

$$\{ \ (s, s', i, o, F') | \quad (s, s', i, o, F) \in \rightarrow \ \}$$

□

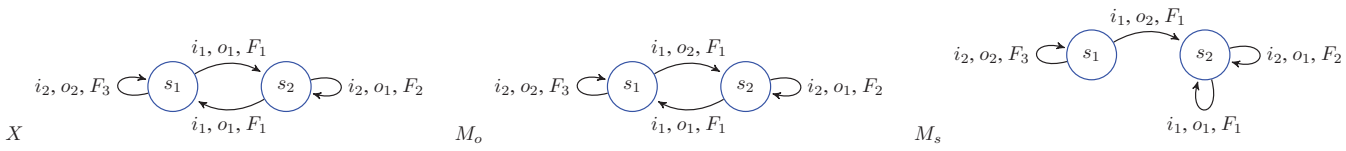

Figure 3. A specification and its mutants.

The possible mutations we can perform in the probability distribution function of a transition must correspond to possible real faults, that is, if the transition has associated an exponential or binomial distribution function, a possible mutation of them should be an exponential or binomial function where the value of the rate parameter (exponential) or the number of trials and/or the success probability (binomial) are changed. We do not consider the mutation of a probability distribution from a Dirac to a uniform, it does not reflects a natural mistake.

Example 2 Let us consider the specification X depicted in Figure 3 and the invariant $\phi = i_1/o_1/F_1, i_1 \mapsto \{\langle o_1, F_1\rangle\}$. We can generate a mutant from the specification by applying the mutation operator CO to the transition $(s_1, s_2, i_1, o_1, F_1)$ for changing the output o_1 into o_2. We would obtain the machine M_o. If we generate a mutant from the specification by applying the mutation operator CGS to the transition $(s_2, s_1, i_1, o_1, F_1)$ for changing the final state s_1 into s_2. We would obtain the machine M_s. □

Once we have obtained the mutant, we will generate randomly traces that we will test applying the invariants. These traces will have different lengths. We have established the minimum length of the traces in $|\mathcal{S}|$.

5 Empirical Results

In this section we present the results obtained from the empirical experiments we performed for estimating a measure of the effectiveness of the invariants detecting errors. The specification used in this experiment is the one depicted in Figure 1 and the invariants considered are the ones presented in Figure 4.

Although selected specification for performing experiments presents a reduced size (10 states and 21 transitions), it allowed us to generate all the possible mutants using the mutation operators described in Section 4.1. First, we applied the CO operator to all the transitions, changing the associated output by each output available in the specification. Next, we applied the CGS operator modifying the final state of the transitions to each of the states in the specification. Thus, for these two operators we generated an exhaustive set of mutants. However, this is not possible when we apply the CT operator. In this case, the number of possibilities for producing a mutant is astronomic so, we established a finite number of changes for each probability distribution function. We applied several factors of variation to the parameters that defines the distribution functions. In addition, for each function we performed a number of mutations proportional to the number of transitions labelled by it. In Figure 5 we show the amount of mutants generated by the tool that are not non-equivalent to the specification taking into account the mutation operator applied. It is worth to mention that we only considered the non-equivalent mutants. In other case, our invariants would not have detected any fault for equivalent mutants.

Non equivalent mutants			Total
CO	CGS	CT	638
201	230	207	

Figure 5. Mutants generated.

Regarding to the set of invariants, we tried to select the most representative and generic taking into account two main criteria: the length and the inclusion of the wildcard $\star$ (see ϕ_{20}). Finally, some of them were designed for representing a restriction of another one, for example, if ϕ_{12} matches a (sub)trace then, ϕ_{13} would match it too but ϕ_{12} is more restrictive than ϕ_{13}.

Following, we extracted 10 traces from each mutant with different length. In our experiment we considered traces of length $k \cdot |\mathcal{S}|$ with $k \in \{1..10\}$. All the invariants were applied to the set of traces. The results obtained were analyzed taking into account the length of the invariants, the length of the traces and the mutation operator that was applied for generating the trace.

In Figure 6 we present the data corresponding to the traces obtained from the application of the CO operator. In the $x-$axis is represented the length of the traces, and in the $y-$axis the amount of traces killed. The graph presents the results classified by the length of the invariants. We observe that the length of the trace analyzed affects the level of detection of the invariants: greater length of the trace implies better results. In addition, we observe that the invariants of length 1 are the best ones for detecting mutations due to a change of an output.

$\phi_1 =$ $i_2 \mapsto \{\langle o_9, F_1\rangle, \langle o_8, F_2\rangle, \langle o_7, F_1\rangle, \langle o_6, F_3\rangle, \langle o_5, F_3\rangle, \langle o_4, F_2\rangle, \langle o_2, F_1\rangle\}$

$\phi_2 =$ $i_2 \mapsto \{\langle o_9, F_1\rangle, \langle o_8, F_2\rangle, \langle o_7, F_1\rangle, \langle o_6, F_3\rangle, \langle o_5, F_3\rangle, \langle o_4, F_2\rangle, \langle o_2, F_1\rangle, \langle o_1, F_1\rangle\}$

$\phi_3 =$ $i_2 \mapsto \{\langle o_9, F_1\rangle, \langle o_8, F_2\rangle, \langle o_7, F_1\rangle, \langle o_6, F_3\rangle, \langle o_5, F_3\rangle, \langle o_4, F_2\rangle, \langle o_3, F_1\rangle, \langle o_2, F_1\rangle\}$

$\phi_4 =$ $i_2 \mapsto \{\langle o_9, F_1\rangle, \langle o_8, F_2\rangle, \langle o_7, F_1\rangle, \langle o_6, F_3\rangle, \langle o_5, F_3\rangle, \langle o_4, F_2\rangle, \langle o_3, F_1\rangle, \langle o_2, F_1\rangle, \langle o_1, F_1\rangle\}$

$\phi_5 =$ $i_2 \mapsto \{\langle o_{10}, F_3\rangle, \langle o_9, F_1\rangle, \langle o_8, F_2\rangle, \langle o_7, F_1\rangle, \langle o_6, F_3\rangle, \langle o_5, F_3\rangle, \langle o_4, F_2\rangle, \langle o_2, F_1\rangle\}$

$\phi_6 =$ $i_2 \mapsto \{\langle o_{10}, F_3\rangle, \langle o_9, F_1\rangle, \langle o_8, F_2\rangle, \langle o_7, F_1\rangle, \langle o_6, F_3\rangle, \langle o_5, F_3\rangle, \langle o_4, F_2\rangle, \langle o_2, F_1\rangle, \langle o_1, F_1\rangle\}$

$\phi_7 =$ $i_2 \mapsto \{\langle o_{10}, F_3\rangle, \langle o_9, F_1\rangle, \langle o_8, F_2\rangle, \langle o_7, F_1\rangle, \langle o_6, F_3\rangle, \langle o_5, F_3\rangle, \langle o_4, F_2\rangle, \langle o_3, F_1\rangle, \langle o_2, F_1\rangle\}$

$\phi_8 =$ $i_2 \mapsto \{\langle o_{10}, F_3\rangle, \langle o_9, F_1\rangle, \langle o_8, F_2\rangle, \langle o_7, F_1\rangle, \langle o_6, F_3\rangle, \langle o_5, F_3\rangle, \langle o_4, F_2\rangle, \langle o_3, F_1\rangle, \langle o_2, F_1\rangle, \langle o_1, F_1\rangle\}$

$\phi_9 =$ $i_1 \mapsto \{\langle o_{10}, F_1\rangle, \langle o_7, F_3\rangle, \langle o_6, F_2\rangle, \langle o_5, F_1\rangle, \langle o_4, F_3\rangle, \langle o_3, F_1\rangle, \langle o_2, F_3\rangle, \langle o_1, F_1\rangle\}$

$\phi_{10} =$ $i_1 \mapsto \{\langle o_{10}, F_1\rangle, \langle o_8, F_2\rangle, \langle o_7, F_3\rangle, \langle o_6, F_2\rangle, \langle o_5, F_1\rangle, \langle o_4, F_3\rangle, \langle o_3, F_1\rangle, \langle o_2, F_3\rangle, \langle o_1, F_1\rangle\}$

$\phi_{11} =$ $i_2/o_8/F_2,$ $i_2 \mapsto \{\langle o_{10}, F_3\rangle, \langle o_8, F_2\rangle, \langle o_7, F_1\rangle, \langle o_6, F_3\rangle, \langle o_5, F_3\rangle, \langle o_4, F_2\rangle, \langle o_2, F_1\rangle, \langle o_1, F_2\rangle\}$

$\phi_{12} =$ $i_2/o_8/F_2,$ $i_1 \mapsto \{\langle o_{10}, F_1\rangle, \langle o_8, F_2\rangle, \langle o_7, F_3\rangle, \langle o_6, F_2\rangle, \langle o_5, F_1\rangle, \langle o_4, F_3\rangle, \langle o_3, F_1\rangle\}$

$\phi_{13} =$ $i_2/o_8/F_2,$ $i_1 \mapsto \{\langle o_{10}, F_1\rangle, \langle o_8, F_2\rangle, \langle o_7, F_3\rangle, \langle o_6, F_2\rangle, \langle o_5, F_1\rangle, \langle o_4, F_3\rangle, \langle o_3, F_1\rangle, \langle o_1, F_1\rangle\}$

$\phi_{14} =$ $i_2/o_8/F_2,$ $i_2 \mapsto \{\langle o_{10}, F_3\rangle, \langle o_8, F_2\rangle, \langle o_7, F_1\rangle, \langle o_6, F_3\rangle, \langle o_5, F_3\rangle, \langle o_4, F_2\rangle, \langle o_3, F_1\rangle, \langle o_2, F_1\rangle\}$

$\phi_{15} =$ $i_1/o_3/F_1,$ $i_2 \mapsto \{\langle o_9, F_1\rangle, \langle o_7, F_1\rangle, \langle o_5, F_3\rangle, \langle o_4, F_2\rangle, \langle o_3, F_1\rangle, \langle o_2, F_1\rangle, \langle o_1, F_2\rangle\}$

$\phi_{16} =$ $i_1/o_3/F_1,$ $i_1 \mapsto \{\langle o_9, F_3\rangle, \langle o_7, F_3\rangle, \langle o_5, F_1\rangle, \langle o_4, F_3\rangle, \langle o_3, F_1\rangle, \langle o_2, F_3\rangle, \langle o_1, F_1\rangle\}$

$\phi_{17} =$ $i_2/o_8/F_2,$ $i_1/o_3/F_1,$ $i_2 \mapsto \{\langle o_9, F_1\rangle, \langle o_8, F_2\rangle, \langle o_7, F_1\rangle, \langle o_6, F_3\rangle, \langle o_5, F_3\rangle, \langle o_4, F_2\rangle, \langle o_2, F_1\rangle\}$

$\phi_{18} =$ $i_2/o_8/F_2,$ $i_1/o_3/F_1,$ $i_2 \mapsto \{\langle o_9, F_1\rangle, \langle o_8, F_2\rangle, \langle o_7, F_1\rangle, \langle o_6, F_3\rangle, \langle o_5, F_3\rangle, \langle o_4, F_2\rangle, \langle o_2, F_1\rangle, \langle o_1, F_1\rangle\}$

$\phi_{19} =$ $i_2/o_8/F_2,$ $i_1/o_3/F_1,$ $i_1 \mapsto \{\langle o_{10}, F_1\rangle, \langle o_7, F_3\rangle, \langle o_6, F_2\rangle, \langle o_5, F_1\rangle, \langle o_4, F_3\rangle, \langle o_3, F_1\rangle, \langle o_2, F_3\rangle, \langle o_1, F_1\rangle\}$

$\phi_{20} =$ $i_2/o_8/F_2,$ $\star,$ $i_1 \mapsto \{\langle o_{10}, F_1\rangle, \langle o_9, F_1\rangle, \langle o_7, F_3\rangle, \langle o_6, F_2\rangle, \langle o_5, F_1\rangle, \langle o_4, F_3\rangle, \langle o_3, F_1\rangle, \langle o_2, F_3\rangle, \langle o_1, F_1\rangle\}$

$\phi_{21} =$ $i_1/o_3/F_1,$ $i_1/o_3/F_1,$ $i_1 \mapsto \{\langle o_{10}, F_1\rangle, \langle o_8, F_2\rangle, \langle o_7, F_3\rangle, \langle o_6, F_3\rangle, \langle o_5, F_1\rangle, \langle o_4, F_3\rangle, \langle o_3, F_1\rangle, \langle o_2, F_3\rangle, \langle o_1, F_1\rangle\}$

Figure 4. Invariant suite.

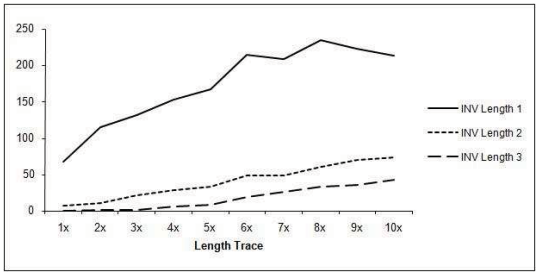

Figure 6. Killed mutants CO operator.

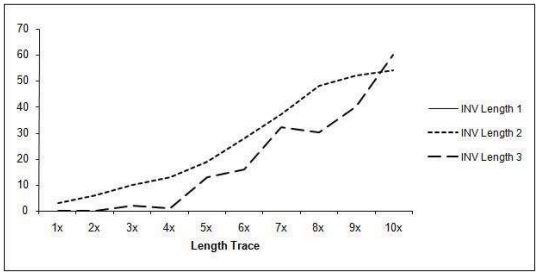

Figure 7. Killed mutants CGS operator.

In Figure 7 we present the results obtained when we consider the traces derived from the application of the mutation operator CGS. Let us note that there exists a set of invariants which are not able to kill any mutant, the invariants of length 1. The reason is that the application of the CGS operator does not affect the functional behaviour of the system, that is, the mutation only changes the final state of a transition, not the input/output/function that labels it. The invariants of length 1 only require conditions over a pair of input/output actions, and they have not been modified as result of the mutation. Thus, these invariants do not detect any fault in the possible pairs of input/output actions in the traces or the time values associated to them. We require at least invariants of length 2 to find an error in a trace, due to the fact that this kind of mutations are reflected in the behaviour of the system following the execution of the modified transition by the operator.

If we analyze the results obtained when we consider the traces derived from the application of the mutation operator

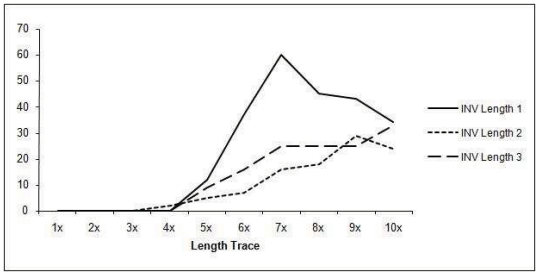

Figure 8. Killed mutants CT operator.

CT, represented in Figure 8, we only can conclude that the traces must have a minimum length. This length depends on the minimal size of the sample that we require in order to check that it fits with the associated distribution function. In this study we established a fix size of 10 time values for the sample. However we cannot deduce a clear relation between the length of the invariant and its power of detection.

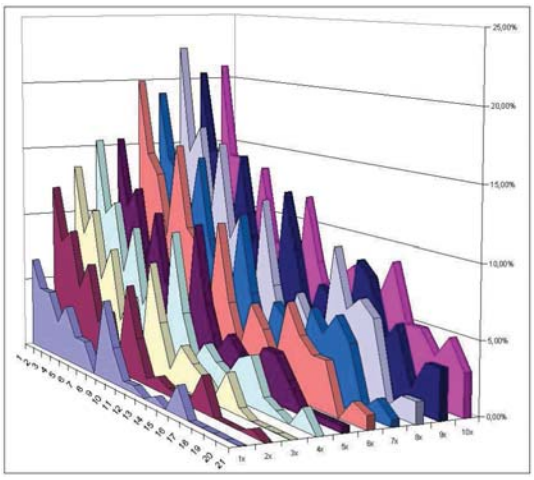

Figure 9. Killed CO mutants by invariant.

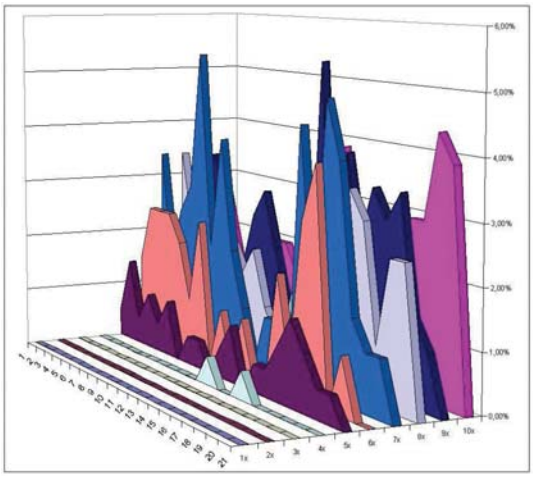

Figure 11. Killed CT mutants by invariant.

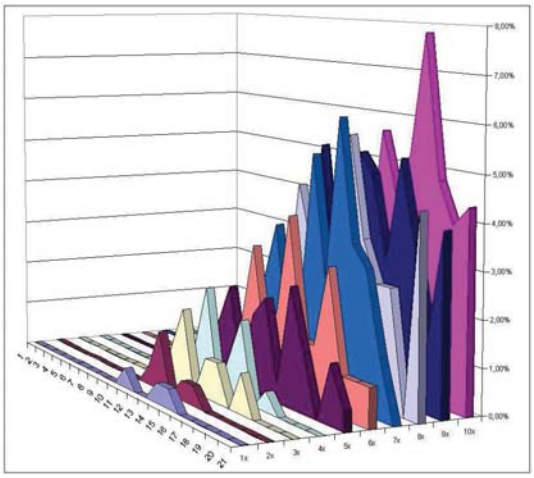

Figure 10. Killed CGS mutants by invariant.

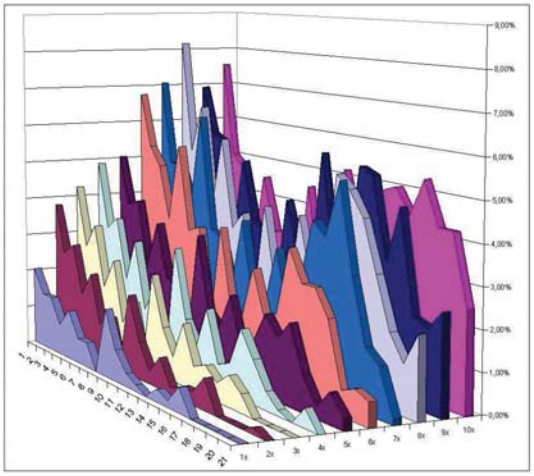

Figure 12. Killed mutants by invariant.

As we mentioned, in order to check the temporal requirements we apply a hypothesis contrast. It requires to choose a level of confidence, that in our study has been established to a fix value.

We also present the analysis of the behaviour of each of the invariants when they are applied to the traces obtained for the different mutation operators, Figures 9, 10 and 11 for CO operator, CGS operator and CT operator respectively. In addition, we have also considered the global behaviour of them, that is, the power of detection taking into account the mutants obtained from the application of all the possible operators (see Figure 12). The graphs present the percentage of traces killed by each of the 21 invariants, considering the different lengths.

After performing these experiments, we have assigned a level of effectiveness for each of the invariants. It is computed by means of the next formula:

$$\frac{w_{CO} \cdot M_{CO}(\phi_i) + w_{CT} \cdot M_{CT}(\phi_i) + w_{CGS} \cdot M_{CGS}(\phi_i)}{NumTracesMut \cdot M_{TOT}}$$

where $\sum_{i \in \{CO,CT,CGS\}} w_i = 1$.

The variables $M_{CO}(\phi_i)$, $M_{CGS}(\phi_i)$ and $M_{CT}(\phi_i)$ represent the number of errors detected by the invariant ϕ_i in the traces extracted from mutants obtained by applying the CO, CGS and CT mutation operators, respectively. The variable M_{TOT} corresponds to the number of mutants generated, and $NumTracesMut$ represents the number of traces extracted from each mutant. In this case study we have 638 mutants and 10 traces have been generated from

w_{co}	w_{ct}	w_{cgs}	Invariant Suite
$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\phi_1 > \phi_3 > \phi_2 > \phi_5 > \phi_9 > \phi_{15} > \phi_{12} > \phi_{16} > \phi_4 \simeq \phi_7$
1	0	0	$\phi_1 > \phi_3 > \phi_2 > \phi_5 > \phi_9 > \phi_4 > \phi_7 > \phi_6 > \phi_{10} > \phi_{16}$
0	1	0	$\phi_{17} > \phi_{16} > \phi_{14} > \phi_{12} > \phi_1 > \phi_3 > \phi_2 \simeq \phi_5 \simeq \phi_7 > \phi_{19}$
0	0	1	$\phi_{15} > \phi_{12} > \phi_{18} > \phi_{17} > \phi_{16} > \phi_{14} \simeq \phi_{19} > \phi_{21} > \phi_{11} > \phi_{13}$
0.2	0.2	0.6	$\phi_1 > \phi_{15} > \phi_{12} > \phi_3 \simeq \phi_{16} > \phi_{17} > \phi_2 \simeq \phi_5 > \phi_{18} > \phi_9$
0.4	0.4	0.2	$\phi_1 > \phi_3 > \phi_2 > \phi_5 > \phi_9 > \phi_7 > \phi_4 > \phi_{16} > \phi_{15} > \phi_{12}$
0.5	0.2	0.3	$\phi_1 > \phi_3 > \phi_2 > \phi_5 > \phi_9 > \phi_4 > \phi_{15} > \phi_7 > \phi_{12} \simeq \phi_{16}$
0.5	0.2	0.3	$\phi_1 > \phi_3 > \phi_2 \simeq \phi_5 > \phi_{12} \simeq \phi_{15} > \phi_{16} > \phi_{17} > \phi_9 > \phi_{18}$

Figure 13. Invariants Suite selected by preference.

each of them.

This formula provide us with the percentage of faults detected by each invariant, according to the different weights associated to the three kinds of mutants. The weights are indicated by the values w_{CO}, w_{CT} and w_{CGS}. These values are selected by the tester according to her preferences for detecting the different kinds of errors. In our case study we have considered different values in order to obtain the most adequate set of invariants for each of them. The results are depicted in Figure 13. The expression $\phi_i > \phi_j$ indicates that the level of detection of ϕ_i is greater than the one of ϕ_j. The difference may be very small, but there exist. If the power of detection of both invariants is equal we represent it by $\phi_i \simeq \phi_j$. For example we can observe that if we assume that the functional behaviour of the system is correct and we are only interesting in checking the temporal behaviour, we would select the third suite of invariants.

6 Conclusions

In this paper we have presented the mutation testing methodology that is applied in the tool PASTE. The tool automates the passive testing approach for systems that present stochastic timed restrictions introduced in previous works. The invariants represent properties that the specifications, described by means of the formalism TFSM, must fulfill. In order to check the effectiveness of the invariants for detecting errors, the tool provides us with a functionality based on a mutation testing methodology. The specifications are mutated using the mutation operators available that are described in this work. Then, different traces are extracted from the mutants that simulate real faults. These traces are used to test if the invariants proposed by the tester can find the errors and a measure of its effectiveness is estimated. In addition to introduce the formal details and the description of how this module works we have presented the analysis of the results obtained from the experiments performed.

Acknowledgements

We would like to thank the anonymous reviewers for the careful reading and their useful suggestions that have improved the quality of the final version of the paper. In particular, we have redone all our experiments, by considering more complex systems, so that we have obtained additional lessons from the application of our framework.

References

[1] C. Andrés, M.G. Merayo, and M. Núñez. Passive testing of timed systems. In *6th Int. Symposium on Automated Technology for Verification and Analysis, ATVA'08, LNCS 5311*, pages 418–427. Springer, 2008. An extended version is available at http://kimba.mat.ucm.es/manolo/papers/atva08-passive-extended.pdf.

[2] C. Andrés, M.G. Merayo, and M. Núñez. Passive testing of stochastic timed systems. In *2nd Int. Conf. on Software Testing, Verification, and Validation, ICST'09 (in press)*. IEEE Computer Society Press, 2009.

[3] J.H. Andrews, L.C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? In *27th Int. Conf. on Software Engineering, ICSE'05*, pages 402–411. ACM Press, 2005.

[4] J.A. Arnedo, A. Cavalli, and M. Núñez. Fast testing of critical properties through passive testing. In *15th Int. Conf. on Testing Communicating Systems, TestCom'03, LNCS 2644*, pages 295–310. Springer, 2003.

[5] E. Bayse, A. Cavalli, M. Núñez, and F. Zaïdi. A passive testing approach based on invariants: Application to the WAP. *Computer Networks*, 48(2):247–266, 2005.

[6] B.S. Bosik and M.Ü. Uyar. Finite state machine based formal methods in protocol conformance test-

ing. *Computer Networks & ISDN Systems*, 22:7–33, 1991.

[7] L. Bottaci and E.S. Mresa. Efficiency of mutation operators and selective mutation strategies: An empirical study. *Software Testing, Verification and Reliability*, 9(4):205–232, 1999.

[8] D.A. Carrington and P.A: Stocks. A tale of two paradigms: Formal methods and software testing. In *Z User Workshop*, pages 51–68. Springer, Workshops in Computing, 1994.

[9] A. Cavalli, C. Gervy, and S. Prokopenko. New approaches for passive testing using an extended finite state machine specification. *Information and Software Technology*, 45:837–852, 2003.

[10] D. Clarke and I. Lee. Automatic generation of tests for timing constraints from requirements. In *3rd Workshop on Object-Oriented Real-Time Dependable Systems, WORDS'97*, pages 199–206. IEEE Computer Society Press, 1997.

[11] A. En-Nouaary, R. Dssouli, and F. Khendek. Timed Wp-method: Testing real time systems. *IEEE Transactions on Software Engineering*, 28(11):1024–1039, 2002.

[12] S.C.P.F. Fabbri, J.C. Maldonado, T. Sugeta, and P.C. Masiero. Mutation testing applied to validate specifications based on statecharts. In *10th IEEE Int. Symposium on Software Reliability Engineering, ISSRE'99*, pages 210–219. IEEE Computer Society Press, 1999.

[13] R.M. Hierons and M.G. Merayo. Mutation testing from probabilistic finite state machines. In *3rd Workshop on Mutation Analysis, Mutation'07*, pages 141–150. IEEE Computer Society Press, 2007.

[14] T. Higashino, A. Nakata, K. Taniguchi, and A. Cavalli. Generating test cases for a timed I/O automaton model. In *12th Int. Workshop on Testing of Communicating Systems, IWTCS'99*, pages 197–214. Kluwer Academic Publishers, 1999.

[15] W.E. Howden. Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering*, 8:371–379, 1982.

[16] R. Lai. A survey of communication protocol testing. *Journal of Systems and Software*, 62:21–46, 2002.

[17] D. Lee, D. Chen, R. Hao, R.E. Miller, J. Wu, and X. Yin. Network protocol system monitoring: a formal approach with passive testing. *IEEE/ACM Transactions on Networking*, 14:424–437, 2006.

[18] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines: A survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[19] D. Mandrioli, S. Morasca, and A. Morzenti. Generating test cases for real time systems from logic specifications. *ACM Transactions on Computer Systems*, 13(4):356–398, 1995.

[20] M.G. Merayo, M. Núñez, and I. Rodríguez. Extending EFSMs to specify and test timed systems with action durations and timeouts. *IEEE Transactions on Computers*, 57(6):835–848, 2008.

[21] R. Nilsson, J. Offutt, and J. Mellin. Test case generation for mutation-based testing of timeliness. In *2nd Workshop on Model Based Testing, MBT'06*, pages 97–114. Electronic Notes in Theoretical Computer Science 164(4), 2006.

[22] M. Núñez and I. Rodríguez. Conformance testing relations for timed systems. In *5th Int. Workshop on Formal Approaches to Software Testing, FATES'05, LNCS 3997*, pages 103–117. Springer, 2006.

[23] J. Offutt. A practical system for mutation testing: Help for the common programmer. In *7th International Test Conference, ITC'94*, pages 824–830. IEEE Computer Society Press, 1994.

[24] I. Rodríguez, M.G. Merayo, and M. Núñez. HOTL: Hypotheses and observations testing logic. *Journal of Logic and Algebraic Programming*, 74(2):57–93, 2008.

[25] J. Springintveld, F. Vaandrager, and P.R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1-2):225–257, 2001. Previously appeared as Technical Report CTIT-97-17, University of Twente, 1997.

[26] T. Sugeta, J.C. Maldonado, and W.E. Wong. Mutation testing applied to validate SDL specifications. In *16th IFIP Int. Conf. on Testing of Communicating Systems,TestCom'04, LNCS 2978*, pages 193–208. Springer, 2004.

[27] J. Tretmans. Testing concurrent systems: A formal approach. In *10th Int. Conf. on Concurrency Theory, CONCUR'99, LNCS 1664*, pages 46–65. Springer, 1999.