Constraint-Based Oracles for Timed Distributed Systems

Nassim Benharrat, Christophe Gaston, Robert M. Hierons, Arnault Lapitre, Pascale Le Gall

Constraint-based oracles for timed distributed systems

Nassim Benharrat[1,3], Christophe Gaston[1],
Robert M. Hierons[2], Arnault Lapitre[1] and Pascale Le Gall[3]

[1] CEA, LIST, Laboratory of Model Driven Engineering for Embedded Systems,
P.C. 174, Gif-sur-Yvette, 91191, France
firstname.lastname@cea.fr
[2] Brunel University London, Uxbridge, Middlesex, UK, UB8 3PH
rob.hierons@brunel.ac.uk
[3] Laboratoire MICS, CentraleSupélec, Université Paris-Saclay,
92295 Châtenay-Malabry, France
firstname.lastname@centralesupelec.fr

Abstract. This paper studies the situation in which the system under test and the system model are distributed and have the same structure; they have corresponding remote components that communicate asynchronously. In testing, a component with interface C_i has its own local tester that interacts with C_i and this local tester observes a local trace consisting of inputs, outputs and durations as perceived by C_i. An observation made in testing is thus a *multi-trace*: a tuple of (timed) local traces, one for each C_i. The conformance relation for such distributed systems combines a classical unitary conformance relation for localised components and the requirement that the communication policy was satisfied. By expressing the communication policy as a constraint satisfaction problem, we were able to implement the computation of test verdicts by orchestrating localised off-line testing algorithms and the verification of constraints defined by message passing between components. Lastly, we illustrate our approach on a telecommunications system.
Keywords: Model-based testing, Distributed testing, Timed Input Output Transition Systems, Off-line testing, Constraint-based testing.

1 Introduction

Distributed systems can be seen as collections of physically remote reactive systems communicating through communication media. The classical approach to testing such systems involves placing a local tester at each localised interface, with each local tester only observing the events at its interface. If testers do not exchange synchronisation messages and there is no global clock, this corresponds to the ISO standardised distributed test architecture [11]. The result of test execution can be modelled as a collection of logs (local traces); each is a sequence of messages involving a given localised system.

Model-Based Testing (MBT) [8, 17, 21] aims to automate three central processes in testing, namely: the *test generation* process whose purpose is to extract test cases from a behavioural model of the System Under Test (SUT), the *test execution* process whose purpose is to orchestrate the stimulation of the SUT with input test data and the collection of the SUT's reactions and finally, the *verdict (oracle) computation* phase whose purpose is to analyse the results of test case executions, given as execution traces, in order to identify faults by checking traces against the model[4]. This comparison is based on a conformance relation that relates traces of SUTs and traces of their associated models.

MBT was first explored in a centralised context but extensions to distributed SUTs have been defined, initially motivated by protocol conformance testing [19]. This includes work that uses Input Output Transition System (IOTS) as the modelling formalism [3, 9, 10]. In the context of distributed testing from Timed IOTS (TIOTS), in [6], we extended the *tioco* conformance relation [14, 15, 20] to define a conformance relation *dtioco* for timed distributed testing. The model of a distributed SUT is given as a tuple of TIOTSs, each modelling one of the localised subsystems of the SUT. The result of test case execution is a tuple of timed traces (a timed trace is a trace in which delays between consecutive interactions of the tester with the localised SUT are recorded). Under the hypothesis that localised systems communicate in a multi-cast mode, we have shown that the verdict computation process can be conducted by combining centralised MBT techniques for each localised system, using the *tioco* conformance relation, and a step-by-step algorithm whose purpose is to check that the tuple of timed traces is consistent with the underlying communication hypothesis [6].

The goal of this paper is twofold. First, we propose an improvement of the algorithm presented in [6] by formulating the oracle problem as a constraint solving problem. While the previous algorithm analyses a multi-trace by mimicking step by step emissions and receptions of messages, as well as the passage of time, in this article, we reformulate the verification of message passing as a set of inequality constraints that can be supported by a constraint solver. Compared to the one introduced in [6], the new algorithm treats durations between communication actions as real numbers. In [6] those durations had to be representable as multiples of a basic time unit, which only allowed us to consider durations in a set isomorphic to the set of natural numbers. The previous approach also included backtracking. In the new algorithm, durations may be any real number that falls in the theory addressed by the considered solver. Second, we present the tool that solves the oracle problem using both a localised verdict computation approach for *tioco* (presented in [1]) and the verification of constraints to check internal communications between localised systems. We consider a case study modelling a telecommunication system, named *PhoneX* [18] specified as a collection of Timed Input Output Symbolic Transition Systems (TIOSTS), which are symbolic representations of TIOTS.

Section 2 introduces the types of models used and Section 3 presents the PhoneX case study. Section 4 recalls the testing framework and introduces the

[4] When the processes are intertwined testing is on-line; otherwise it is off-line.

verification of message passing using constraints. Section 5 describes a scalability study, based on the PhoneX example, that applied mutation techniques to generate correct and faulty trace tuples. Section 6 discusses related works and Section 7 gives concluding remarks.

2 Modelling framework

2.1 TIOSTS

Timed Input Output Symbolic Transition Systems (TIOSTS) are symbolic automata built over a signature $\Sigma = (\Omega, A, T, C)$ where $\Omega = (S, F)$ is an equational logic signature with S a set of types and F a set of functions provided with an arity. The functions are interpreted in a model M as usual. A is a set of data variables used to store input values, to denote system state evolutions and to define guards. T is a set of clocks, which are variables whose values are elements of a set D isomorphic to non-negative real numbers and that are used to denote durations. D^+ will denote $D \setminus \{0\}$. M is supposed to contain D. Variables in $A \cup T$ are assigned values by interpretations of the form $\nu : A \cup T \to M$; $M^{A \cup T}$ is the set of all interpretations. Finally, C is a set of channels partitioned as $C^{in} \coprod C^{out}$ where elements of C^{in} (resp. C^{out}) are *input* (resp. *output*) channels. The set of terms $T_\Omega(A)$ over Ω and A is inductively defined as usual and variable interpretations are canonically extended to terms. The set of symbolic actions $Act(\Sigma)$ is $I(\Sigma) \cup O(\Sigma)$ with $I(\Sigma) = \{c?x | x \in A, c \in C^{in}\}$ and $O(\Sigma) = \{c!t | t \in T_\Omega(A), c \in C^{out}\}$. In order to simplify the exposition, at the level of our modelling framework, we consider messages that contain only a single piece of data. However, at the tooling level, without adding any particular difficulties, messages contain 0 (signals $c!$ or $c?$), 1 or n data ($c!(t_1, \ldots, t_n)$ or $c?(x_1, \ldots, x_n)$, the x_i being different variables of A).

A TIOSTS is a triple $\mathbb{G} = (Q, q_0, Tr)$ where Q is a set of states, q_0 is a distinguished element of Q called the initial state, and Tr is a set of labeled transitions. A transition is defined by a tuple $(q, \phi, \psi, \mathbb{T}, act, \rho, q')$ where q (resp. q') is the source (resp. target) state of the transition, ϕ is a formula, called time guard, of the form $z \leq Cst$ or $z \geq Cst$ where $z \in T$ and Cst is a constant interpreted in D (ϕ constrains the delay at which the action act occurs), ψ is an equational logic formula, called data guard (ψ is a firing condition on attribute variables), $\mathbb{T} \subseteq T$ is a set of time variables (to be reset to 0 when the transition is executed), act is a communication action and ρ assigns terms of $T_\Omega(A)$ to variables in A in order to represent state evolutions. In the sequel, we use $M \models_\nu \varphi$ to say that φ holds for interpretation ν. The set of paths of $\mathbb{G}$ contains the empty sequence ε and all sequences $tr_1 . \cdots . tr_n$ of transitions of Tr such that $source(tr_1) = q_0$ and for all $i < n$, $target(tr_i) = source(tr_{i+1})$.

Concrete actions are values exchanged through channels. The set of concrete actions over C is thus $Act(C) = I(C) \cup O(C)$ where $I(C) = \{c?v \mid c \in C^{in}, v \in M\}$ are inputs and $O(C) = \{c!v \mid c \in C^{out}, v \in M\}$ are outputs. Given $act \in Act(C)$ of the form $c\Delta v$ with $\Delta \in \{!, ?\}$, $chan(act)$ refers to c, $\overline{act}$ refers to its mirror action, $c\overline{\Delta}v$ with $\overline{!} = ?$ and $\overline{?} = !$. Variable interpretations are canonically extended to symbolic actions ($\nu(c?x) = c?(\nu(x))$ and $\nu(c!t) = c!\nu(t)$).

A concrete action is generally observed after a delay has occurred since the previous occurrence of a concrete action. This is captured by the notion of *events*.

Definition 1 (Events). *The set of (resp.* initialised*)* events *over C is defined as $Evt(C) = (D^+ \cup \{\_\}) \times (Act(C) \cup \{\delta\})$ (resp. $IEvt(C) = D^+ \times (Act(C) \cup \{\delta\})$).*

Pair (d, a) represents the observation of concrete action a after delay d. Following [21], symbol 'δ' is used to denote the absence of observation of a concrete action (*i.e.* quiescence). Let us point out that usually, in a pure timed framework, δ may be useless (e.g. [6,13,14]). Here, the use of δ is a side effect of considering atomic actions as events. Indeed, expressing that a system is quiescent after a duration d has to be representable as an event, and thus, we need a symbol to represent these quiescent situations as a couple (d, δ). Symbol '$\_$' is introduced to denote the absence of the observation of a delay (i.e $(\_, a)$). We require this so that the first action of a localised trace need not be stamped with a duration. In addition, between two consecutive concrete actions on one location, we require that the delay is greater than zero so that two events do not occur simultaneously. Given $ev \in Evt(C)$, we let $act(ev) = a$ and $delay(ev) = d$ if $ev = (d, a)$ with $d \in D^+$, else $delay(ev) = 0$ $(ev = (\_, a))$. In the sequel $\delta Evt(C)$ denotes $\{ev \in Evt(C) \backslash act(ev) = \delta\}$.

Definition 2 (Timed traces). *The* set $ITraces(C)$ of initialised traces over C *is*[5] $(IEvt(C) \setminus \delta Evt(C))^*.(\varepsilon + \delta Evt(C))$.
The set $UTraces(C)$ of uninitialised traces over C *is* $\{u(\sigma) \mid \sigma \in ITraces(C)\}$ *where* $u(\sigma)$ *denotes* ε *if* $\sigma = \varepsilon$ *and* $(\_, a).\sigma'$ *if* $\sigma = (d, a).\sigma'$.
The set $TTraces(C)$ of timed traces over C *is* $UTraces(C) \cup ITraces(C)$.

Any event of an initialised trace contains a duration and a concrete action. For the first event, this duration represents a delay between some distinguished moment (e.g. since the time at which a tester started to measure the duration) and the first observed action. Uninitialised traces are timed traces for which no initial instant is identified. Finally, note that quiescence is only observed at the end of traces, when no communication action follows it. Indeed when a communication action a occurs after a period of time where an implementation remains silent, this period of time is captured by the delay of the event introducing a.

For $\sigma \in TTraces(C)$, $dur(\sigma)$ denotes the *duration of* σ, which is 0 if σ is ε, and otherwise is the sum of all delays of events in σ. $Pref(\sigma)$ denotes the set of *prefixes of* σ defined as $\{\varepsilon\}$ if σ is ε and $Pref(\sigma') \cup \{\sigma\}$ if σ is of the form $\sigma'.ev$. Moreover, for an action a in $Act(C)$, $|\sigma|_a$ denotes the number of occurrences of a in σ. $pref(\sigma, a, n)$ stands for the smallest prefix of σ that contains n occurrences of a when this prefix exists. Finally, using the $pref$ operation, we introduce an operation that measures the elapsed time at the nth occurrence of an event a from the beginning of the trace. By convention, if a trace contains strictly fewer

[5] E^* is the set of finite sequences of elements in E with ε as neutral element for sequence concatenation.

than n occurrences of a, then the associated duration is that of the entire trace.

$$dur\_occ(\sigma, a, n) = \begin{cases} dur(pref(\sigma, a, n)) & \text{if } pref(\sigma, a, n) \text{ exists} \\ dur(\sigma) & \text{else} \end{cases}$$

We now define *runs* of transitions of TIOSTS:

Definition 3 (Runs of transitions). *Let $\mathbb{G} = (Q, q_0, Tr)$ be a $TIOSTS$ over Σ. The set $Snp_M(\mathbb{G})$ of* snapshots *of $\mathbb{G}$ is the set $Q \times M^{A \cup T}$. For $tr = (q, \phi, \psi, \mathbb{T}, act, \rho, q') \in Tr$, the* set of runs of *$tr$ is the set $Run(tr) \subseteq Snp_M(\mathbb{G}) \times Evt(C) \times Snp_M(\mathbb{G})$ s.t. $((q, \nu), ev, (q', \nu')) \in Run(tr)$ iff there exist $d \in D$ and $\xi : A \cup T \to M$ satisfying:*

- *for all $w \in T$, $\xi(w) = \nu(w) + d$,*
- *if $act = c!t$ then for all $x \in A$, $\xi(x) = \nu(x)$, else ($act = c?x$) for all $y \in A \setminus \{x\}$, $\xi(y) = \nu(y)$,*

and such that we have either $ev = (d, \xi(act))$ or $ev = (\_, \xi(act))$, $\forall x \in A, \nu'(x) = \xi(\rho(x))$, $\forall w \in \mathbb{T}$, $\nu'(w) = 0$, $\forall w \in (T \setminus \mathbb{T}), \nu'(w) = \xi(w)$, $M \models_\xi \phi$ and $M \models_\xi \psi$.

In Definition 3, ξ is an intermediate interpretation whose purpose is to let time pass from ν for all clocks ($\xi(w) = \nu(w)+d$) and take into account a potential input value (denoted by $\xi(x)$ if $act = c?x$). Guards of the transition should be satisfied by ξ and if it is the case then the transition can be fired resulting on a new interpretation ν' updating data variables according to ρ and resetting clocks occurring in $\mathbb{T}$.

For a path p of $\mathbb{G}$, the *set of timed traces of p*, denoted $TTraces(p)$ is $\{\varepsilon\}$ if $p = \varepsilon$ and if p is of the form $tr_1.\cdots.tr_n$, $TTrace(p)$ contains all sequences of events $ev_1 \cdots ev_n$ such that there exists a sequence of runs $r_1 \cdots r_n$ satisfying: for all $i \leq n$, r_i is a run of tr_i of the form $(snp_i, ev_i, snp'_{i+1})$ and for all $j < n$ we have $snp'_j = snp_{j+1}$ and such that all events are initialised except for $i = 1$, i.e. ev_1 is of form $(\_, a_1)$ and for all $i > 1$, ev_i is of form (d_i, a_i).

By taking into account the particular action δ, the set of timed traces of $\mathbb{G}$, denoted $TTraces(\mathbb{G})$, is defined as:

- For all $p \in Path(\mathbb{G})$ we have $TTraces(p) \subseteq TTraces(\mathbb{G})$,
- For all $\sigma \in TTraces(\mathbb{G})$ such that there exists no path p and no event ev with $act(ev) \in O(C)$ satisfying $\sigma.ev \in TTraces(p)$, we have $\sigma.(d, \delta) \in TTraces(\mathbb{G})$ if $\sigma \neq \varepsilon$ and $(\_, \delta) \in TTraces(\mathbb{G})$ if $\sigma = \varepsilon$.

2.2 Communication and systems

We now define a *distributed interface* as a collection of localised interfaces.

Definition 4 (Distributed interface). *A distributed interface is a tuple $\Lambda = (C_1, \cdots, C_n)$, with $n \geq 1$, where for all $i \leq n$, C_i is a set of channels such that for any $i \neq j$ we have $C_i^{out} \cap C_j^{out} = \emptyset$. $C(\Lambda)$, which is equal to $\bigcup_{i \leq n} C_i$, is the set of channels of Λ with $C(\Lambda)^{in} = \bigcup_{i \leq n} C_i^{in}$ and $C(\Lambda)^{out} = \bigcup_{i \leq n} C_i^{out}$.*

$C_i^{out} \cap C_j^{out} = \emptyset$ ensures that for a channel c, messages emitted through c can only be emitted from one sender. This is a simplification hypothesis that makes the later formalisation lighter. In a distributed architecture, for a given localised interface C_i of $\Lambda = (C_1, \cdots, C_n)$, C_i^{int} (resp. C_i^{ext}), defined as $\bigcup\{C_i \cap C_j | j \leq n \wedge j \neq i\}$ (resp. $C_i \backslash C_i^{int}$), denotes the set of internal channels (resp. external channels) that can be used to exchange messages with other localised subsystems (resp. exchange messages with the system environment). We let $C^{int}(\Lambda)$ denote $\bigcup_{i \leq n} C_i^{int}$, $C^{ext}(\Lambda)$ denote $\bigcup_{i \leq n} C_i^{ext}$, and $Act(\Lambda)$ denote $I(\Lambda) \cup O(\Lambda)$ with $I(\Lambda) = \bigcup_{i \leq n} I(C_i)$ and $O(\Lambda) = \bigcup_{i \leq n} O(C_i)$. $I^{int}(\Lambda)$ (resp. $O^{int}(\Lambda)$) is the subset of $I(\Lambda)$ (resp. $O(\Lambda)$) whose elements are inputs (resp. outputs) through internal channels. For any $c!v \in O(\Lambda)$, $Sender(\Lambda, c!v)$ stands for the index j such that $c \in C_j^{out}$. We let $Act^{int}(\Lambda) = I^{int}(\Lambda) \cup O^{int}(\Lambda)$, $Evt(\Lambda) = Evt(C(\Lambda))$, and $Evt_{in}^{int}(\Lambda)$ be the set of events whose action is an internal input. We define $Tup(\Lambda)$ to be $TTraces(C_1) \times \cdots \times TTraces(C_n)$. In the sequel, a distributed interface $\Lambda = (C_1, \cdots, C_n)$ is given. An observation made in a system will be a tuple of timed traces where each timed trace represents a local observation. We first introduce the notion of a *multi-trace*, which is a tuple of timed traces characterising compatible communications between a collection of localised subsystems.

Definition 5 (Multi-traces). *The set of* multi-traces of Λ with initial instants*, denoted $IMTraces(\Lambda)$, is the subset of $ITraces(C_1) \times \cdots \times ITraces(C_n)$ defined as follows:*

- ***Empty multi-trace:*** $(\varepsilon, \cdots, \varepsilon) \in IMTraces(\Lambda)$,
- ***multi-trace Extension:*** *for any $\mu = (\sigma_1, \ldots, \sigma_n) \in IMTraces(\Lambda)$, for $ev \in IEvt(C_i)$ for $i \leq n$, $(\sigma_1, \ldots, \sigma_i.ev, \ldots, \sigma_n) \in IMTraces(\Lambda)$ provided that: if $act(ev) \in I(C_i) \cap I^{int}(\Lambda)$, we have $|\sigma_j|_{\overline{ev}} \geq |\sigma_i|_{act(ev)} + 1$ and $dur\_occ(\sigma_j, \overline{ev}, |\sigma_i|_{act(ev)} + 1) < dur(\sigma_i.ev)$ with $j = Sender(\Lambda, \overline{act(ev)})$.*

The set $UMTraces(\Lambda)$ (resp. $MTraces(\Lambda)$) of uninitialised multi-traces *(resp. of* multi-traces*) of Λ is $\{(u(\sigma_1), \cdots, u(\sigma_n)) | (\sigma_1, \cdots, \sigma_n) \in IMTraces(\Lambda)\}$ (resp. $UMTraces(\Lambda) \cup IMTraces(\Lambda)$).*

Initialised multi-traces denote tuples of traces, each trace of the tuple being a partial centralised vision of a common distributed execution. The nature of communication considered is multicast, as captured by the property that an internal message can be received at some C_i only if C_i has consumed fewer occurrences of this message than the number of the corresponding output occurrences. Each trace occurring in an initialised multi-trace starts with an event introducing a duration. All those durations are supposed to start at a common initial instant. Of course, in the context of distributed executions it is generally not possible to observe such a common initial instant. Therefore, we defined uninitialised multi-traces in which the initial durations are not observable. Similar rules have been proposed in [16] to express component composition in a distributed setting.

In distributed testing, we assume that there is a separate tester at each localised interface and there is no global clock for globally ordering distributed

events. Hence, we cannot make any assumption on the different moments at which the different local testers stop observing their associated interfaces. To capture this, we accept as valid observations, tuples made of multi-trace prefixes.

Definition 6 (Observable multi-traces). *The* set of initialised observable multi-traces of Λ, denoted $IOTraces(\Lambda)$, is the smallest set containing $IMTraces(\Lambda)$ and such that for any $(\sigma_1, \cdots, \sigma_i.ev, \cdots, \sigma_n) \in IOTraces(\Lambda)$ we have $(\sigma_1, \cdots, \sigma_n) \in IOTraces(\Lambda)$.

The set of *uninitialised observable multi-traces of* Λ, denoted $UOTraces(\Lambda)$, is the set $\{(u(\sigma_1), \cdots, u(\sigma_n)) | (\sigma_1, \cdots, \sigma_n) \in IOTraces(\Lambda)\}$.

Initialised observable multi-traces characterise observations starting at a common initial instant but ending at different instants depending on the considered component of the interface. Of course, since there is a common initial instant it is possible to order the moments at which the observations of the different traces of the tuple occur (σ_i ends before σ_j if $dur(\sigma_i) < dur(\sigma_j)$). However, in general such an initial instant cannot be identified in testing. Therefore, real observations of system executions should be defined by tuples containing only uninitialised traces, which is captured by uninitialised observable multi-traces.

Definition 7. *Let* $\Lambda = (C_1, \cdots, C_n)$ *be a distributed interface. A* system over Λ *is a tuple* $Sys = (\mathbb{G}_1, \cdots \mathbb{G}_n)$ *such that for all* $i \leq n$ *we have* $\mathbb{G}_i$ *is a TIOSTS over a signature of the form* $(\Omega_i, A_i, T_i, C_i)$. *The* semantics of Sys, *denoted* $TTraces(Sys)$ *is defined as* $(TTraces(\mathbb{G}_1) \times \cdots \times TTraces(\mathbb{G}_n)) \cap UOTraces(\Lambda)$.

3 The PhoneX case study

PhoneX [18] is a central telecommunication system model describing a protocol to establish sessions between phones. It was initially used as a reference to investigate the test case generation capacities of the platform Diversity[6] by the Ericsson company. In our context, PhoneX is interesting since it allows the number of communicating actors to be parameterised, even though there is only one time constraint in the subsystem models. Fig. 1 depicts a scenario of a successful session setup and call establishment between 2 phones. A caller with $Phone_{112}$ initiates a call ($doCall(113)$) to the user of $Phone_{113}$. The PhoneX server, after receiving $Calling(112, 113)$, checks if $Phone_{113}$ is registered, available, and then starts $StartSession(112, 113)$ for communication management and remains available. $Session_{112}^{113}$ informs $Phone_{113}$ that $Phone_{112}$ tried to get in contact ($CalledBy(112)$). The user of $Phone_{113}$ can accept the call ($doAcceptCall$) and informs $Session_{112}^{113}$ using $AcceptingCall$ which can establish communication (multicasting $InitCall$). Each user can end the call (the user of $Phone_{112}$ hangs up, $doEndCall$) and report it ($EndingCall$) to $Session_{112}^{113}$ that closes the connection by multicasting $TermCall$ and becomes available ($EndSession(112, 113)$) again. Fig. 2 depicts the architecture. Components *Caller Client*

[6] https://projects.eclipse.org/proposals/eclipse-formal-modeling-project

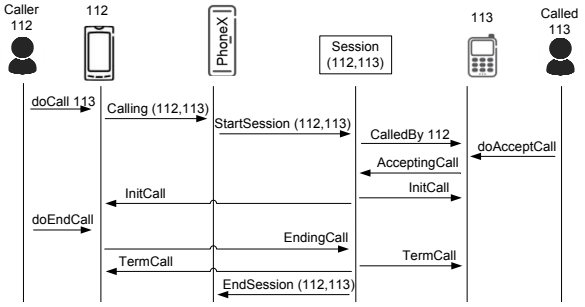

Fig. 1: Interaction scenario of a successful call operation

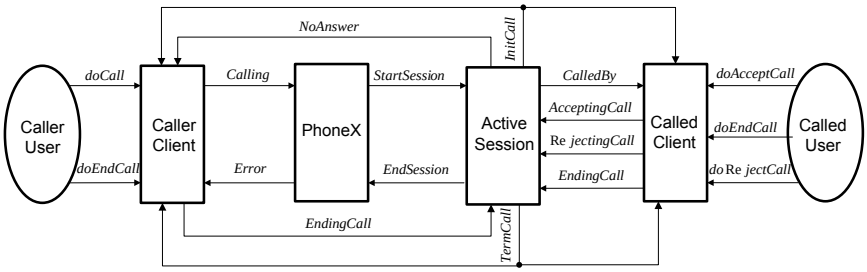

Fig. 2: The PhoneX architecture

and *Called Client* define two roles that registered phones can have. *PhoneX* is the component that plays the role of the telecommunication centre. *Active Session* is a generic representation of sessions created by the centre to manage communications between phones. Communication channels model the media used by components in Fig. 2.

Caller client behaviour (Fig 3(a)). At the *Idle* state, caller *src* receives a call from the environment (a caller) to make a call operation with called *dest*. Then it joins PhoneX central by sending to it *src* and *dest* (caller reaches *Initiating* state). Caller returns to *Idle* state when it receives an error code from PhoneX (PhoneX cannot establish a call due to violated condition of call establishment) or a signal to terminate the call from the active session (due to a call rejection by called client). At *Initiating*, *src* may reach *Established* if a call is established by active session or state *Terminating* if a no-answer (from called client) is observed during a waiting delay. When a call is established (at *Established*), *src* may return to *Idle* by receiving a terminating signal from the active session (due to an ending call by called client) or receive a signal from the environment (a caller) to end the call in progress (caller reaches $UserEndingCall$). At $UserEndingCall$, the caller notifies the active session for terminating the call (caller reaches *Terminating*). At *Terminating*, *src* returns to *Idle* by receiving a terminating signal from the active session.

Called client behaviour (Fig 3(b)). This role is symmetric (on the called client side) to the one described in Fig 3(a)).

PhoneX central behaviour (Fig 4(a)). At the *Idle* state, PhoneX may receive a call and reach *Calling* or get notified of the ending of an already

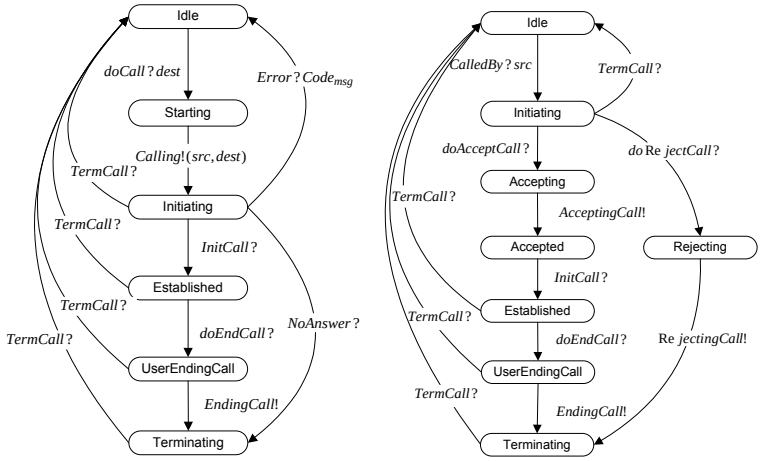

Fig. 3: TIOSTSs $\mathbb{G}_{src}$ and $\mathbb{G}_{dest}$ of Caller and Called clients

active session and return to $Idle$. At $Calling$, PhoneX may start a new session $(src, dest)$ and return to $Idle$ provided that $dest$ is a registered and allowed-to-call number in the Client database and there is no active session with called client $dest$. Otherwise, PhoneX may also return to $Idle$ when $dest$ is not registered in Client database, or calling $dest$ is not allowed, or called client $dest$ is busy.

Session behaviour (depicted in Fig 4(b)). When a new session is started, a Session TIOSTS is instantiated. At the $Idle$ state, Session receives src and $dest$ numbers, it then reaches $Starting$. It notifies $dest$ with a call operation emitted by src and reaches $Initiating$. At $Initiating$, it may reach either $Accepted$ when called client accepts the call during a waiting delay or $Terminating$ if a no-answer is observed during a waiting delay or the call get rejected. At $Accepted$, active session initiates a call between src and $dest$ and reaches state $Established$. Then, either caller or called client may end the call (session reaches $Terminating$ state). At $Terminating$ state session sends a terminating signal to both caller and called clients and reaches $Ending$. Finally, it returns to $Idle$ by notifying PhoneX central of ending the active session.

4 Testing

In [6], we modelled timed distributed systems as tuples $(\mathbb{LS}_1, \ldots, \mathbb{LS}_n)$ where each $\mathbb{LS}_i$ denotes a black box localised system under test. Then we defined a conformance relation *dtioco* to test such a distributed system with respect to a system model $(\mathbb{G}_1, \ldots, \mathbb{G}_n)$. We showed that solving the oracle problem for an observable multi-trace $(\sigma_1, \cdots, \sigma_n)$ reduces to: (a) solve the oracle problem of each σ_i with respect to *tioco* [15] and with $\mathbb{G}_i$ as reference model (unitary testing, see Section 4.1) and, (b) check whether $(\sigma_1, \cdots, \sigma_n)$ is an observable (uninitialised) multi-trace. In Section 4.1 we briefly recall the principles of a simplified[7] version of the unitary testing algorithm defined in [1]. Then in Section

[7] Due to the lack of space.

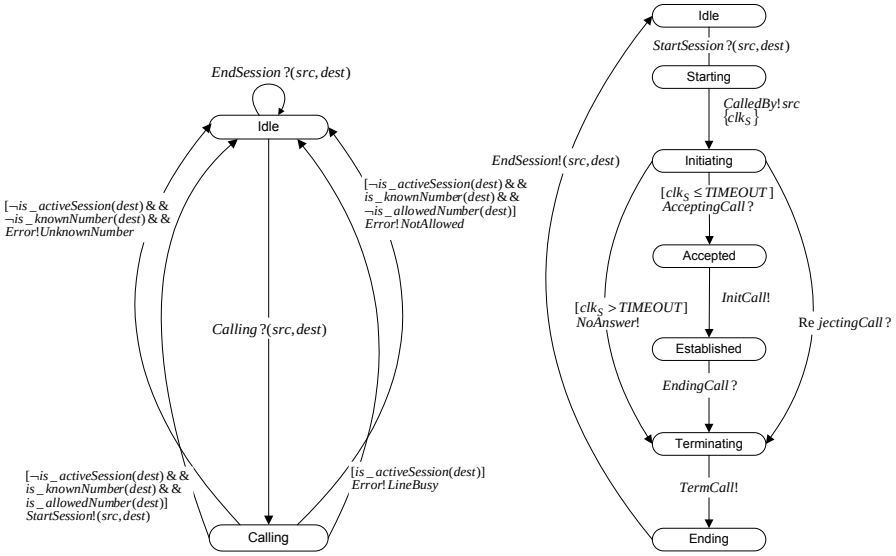

Fig. 4: TIOSTSs $\mathbb{G}_X$ and $\mathbb{G}_S$ of PhoneX Central and Active Session

4.2, we introduce the new algorithm based on constraint solving to decide if a tuple is an observational multi-trace. As compared to [1] and [6], we have slightly adapted our definitions of timed traces in order to deal with events instead of atomic observations such as inputs, outputs or durations; this adaptation has no impact on the results in [1] and [6].

4.1 Unitary testing

A Localised subsystem Under Test (LUT) is defined over a set of channels C as a non-empty subset $\mathbb{LS}$ of $UOTraces(C)$ such that:

- **Input completeness**: for any σ in $\mathbb{LS}$ of the form $\sigma'.ev'$, for any $ev \in Evt(C)$ such that $act(ev) \in I(C)$ and $delay(ev) \leq delay(ev')$, we have $\sigma'.ev \in \mathbb{LS}$.
- **Quiescence**: for all $\sigma \in \mathbb{LS}$ we have:

$$\forall ev \in Evt(C).(act(ev) \in O(C) \Rightarrow \sigma.ev \notin \mathbb{LS})$$

$$\Leftrightarrow$$

$$(\sigma \neq \varepsilon \Rightarrow (\forall d \in D^+, \sigma.(d, \delta) \in \mathbb{LS})) \wedge (\sigma = \varepsilon \Rightarrow (\_, \delta) \in \mathbb{LS})$$

  Moreover for any σ in $\mathbb{LS}$ of the form $\sigma'.ev'$ with $act(ev') = \delta$, for any $ev \in Evt(C)$ with $act(ev) \in O(C)$ we have $\sigma.ev \notin \mathbb{LS}$.
- **Reaction prefix**: for any σ in $\mathbb{LS}$, we have $Pref(\sigma) \subseteq \mathbb{LS}$.

Input completeness is required so that an LUT cannot refuse an input from the environment. **Quiescence** corresponds to situations where the LUT

will not react anymore until it receives a new stimulation. **Reaction prefix** is a realistic property stating that a prefix of an observation is an observation.

The local verdict computation algorithm is based on a symbolic structure $SE(\mathbb{G})_\delta$ computed from the reference model $\mathbb{G}$ obtained by classical symbolic execution techniques. It is a tree-like structure whose nodes are symbolic states used to capture information related to the possible executions of $\mathbb{G}$. A path p is a sequence of consecutive edges relating symbolic states and labelled by symbolic events. The set of executions (uninitialised timed traces) associated to p is characterised by the sequence $ev_1 \cdots ev_n$ of symbolic events labelling the consecutive edges and by the final symbolic state η. Each symbolic event of the sequence is of the form (d_i, act_i) (except ev_1 which is of the form $(\_, act_1)$). Each d_i is a new fresh variable (*i.e.* not used in the definition of $\mathbb{G}$) used to represent durations (they are typed as clocks) and each act_i is of the form $c?z_i$ or $c!t_i$ where z_i is a new fresh variable and t_i is a term built over the same equational logic signature Ω as terms in $\mathbb{G}$ but on a set of new fresh variables. η is of the form $(q, \pi_d, \pi_t, \varrho)$ where q is the state reached in $\mathbb{G}$, π_d is a constraint on new fresh data variables (let F_d be the set of those variables), π_t is a constraint on the set of variables of the form d_i and $\varrho : A \to T_\Omega(F_d)$ associates symbolic values to variables of $\mathbb{G}$. An uninitialised timed trace $ev'_1 \cdots ev'_n$ belongs to p iff for all $i \leq n$:

- ev'_i is of the form $(\_, act'_i)$ (resp. (d'_i, act'_i)) if ev_i is of the form $(\_, act_i)$ (resp. (d_i, act_i)) and act'_i is of the form $c?z'_i$ (resp. $c!t'_i$) if act_i is of the form $c?z_i$ (resp. $c!t_i$).
- Let x_i (resp. x'_i) stand for the variable z_i (resp. z'_i) if act_i (resp. act'_i) is an input and for the term t_i (resp. t'_i) if act_i (resp. act'_i) is an output. The formula $(\bigwedge_{i \leq n} x_i = x'_i) \wedge \pi_d \wedge \pi_t$ is satisfiable.

The verdict computation algorithm analyses successively all events of $\sigma = ev'_1 \cdots ev'_n$ and at each steps it computes the set of paths to which the already analysed prefix of σ belongs. As soon as possible a verdict is emitted[8]:

- *Fail* if $act(ev'_i)$ is an output or δ and the set of path becomes empty, or else $act(ev'_i)$ is an input (d'_i, act'_i) and there exists an event $ev''_i = (d''_i, act''_i)$ where act''_i is an output (not δ) satisfying $d''_i < d'_i$ and $ev'_1 \cdots ev'_{i-1}.ev''_i$ belongs to some path of $SE(\mathbb{G})_\delta$.
- *Inconc* if $act(ev'_i)$ is an input (d'_i, act'_i), the set of path becomes empty, and for all events $ev''_i = (d''_i, act''_i)$ where act''_i is an output (not δ), $d''_i < d'_i$ we have $ev'_1 \cdots ev'_{i-1}.ev''_i$ does not belong to any path of $SE(\mathbb{G})_\delta$.
- *Pass* if σ is fully analysed without generating any of the previous verdicts.

4.2 Communication testing

An SUT over Λ is a tuple $\mathbb{S} = (\mathbb{LS}_1, \ldots, \mathbb{LS}_n)$ where $\mathbb{LS}_i$ is an LUT defined over C_i (all $i \leq n$). The semantics of $\mathbb{S}$, denoted $Obs(\mathbb{S}) \subseteq \mathbb{LS}_1 \times \cdots \times \mathbb{LS}_n$,

[8] In accordance with the *tioco* conformance relation

contains all observations that can be made when executing $\mathbb{S}$. The goal of Algorithm 1 is to check whether those observations reveal communication errors by checking whether they are in $UOTraces(\Lambda)$. It is based on the property that an uninitialised observable multi-trace $\mu = (\sigma_1, \cdots, \sigma_n)$ is such that each σ_i is either empty or of the form $(\_, a_i).\sigma'_i$, but in the latter case μ has been obtained from an initialised observable multi-trace of the form $\mu' = (\sigma''_1, \cdots, \sigma''_n)$ where σ''_i is ε for $\sigma_i = \varepsilon$ and of the form $(d_i, a_i).\sigma'_i$ for σ_i of the form $(\_, a_i).\sigma'_i$. Thus, $(\sigma_1, \cdots, \sigma_i.ev, \cdots \sigma_n) \in UOTraces(\Lambda)$ iff there exist durations $d_1, \cdots, d_n$ where $\mu'' = (\sigma''_1, \cdots, \sigma''_i, \cdots \sigma''_n) \in IOTraces(\Lambda)$. We check whether such durations exist by considering them as n variables $d_1, \cdots, d_n$ (of type D); we construct constraints on these variables characterising the properties of observable traces. By definition, only the occurrence of an internal input might break the property. There are two reasons for allowing an initialised observable multi-trace to be extended by an internal input. The first is that a sufficient number of corresponding internal outputs have previously been emitted. The second is that at the time when the extension is performed, the trace emitting the corresponding internal output is no longer observed. If σ_i is the trace extended by internal input a, $\rho = \sigma_i.a$ and σ_j is the trace at the interface that sends $\overline{a}$, the first case correspond to situation in which $pref(\sigma_j, \overline{a}, |\rho|_a)$ exists and C: $d_i + dur(\rho) > d_j + dur\_occ(\sigma_j, \overline{a}, |\rho|_a)$ holds. The latter case corresponds to situations in which $pref(\sigma_j, \overline{a}, |\rho|_a)$ does not exist and C': $d_i + dur(\rho) > d_j + dur(\sigma_j)$ holds. However, by definition of dur_occ, when $pref(\sigma_j, \overline{a}, |\rho|_a)$ does not exist we have that $dur\_occ(\sigma_j, \overline{a}, |\rho|_a) = dur(\sigma_j)$, which means that the constraints C and C' are equivalent. Therefore both cases can be treated in the same way by requiring that C holds, as is done in Algorithm 1. Every new constraint to be considered is added to the set E (line 10). Sat is a function on sets of constraints such that $Sat(E)$ returns $True$ if all constraints in E are simultaneously satisfiable and $False$ otherwise.

Algorithm 1: $ObsMult(\mu, d, \Lambda)$

```
   Data: μ = (σ_1, ··· , σ_n) tuple of traces, d = (d_1, ··· , d_n) n variables, Λ system signature
   Result: a verdict stating whether or not μ is an observable multi-trace
 1 begin
 2     E ← ∅;
 3     for i ∈ [1 ··· n] do
 4         ρ ← ε ;
 5         foreach ev ∈ σ_i do
 6             ρ ← ρ.ev ;
 7             if act(ev) ∈ I(C^int(Λ)) then
 8                 a ← act(ev);
 9                 j ← Sender(Λ, act(ev)‾);
10                 E ← E ∪ {(d_i + dur(ρ) > d_j + dur_occ(σ_j, a‾, |ρ|_a)};
11                 if ¬Sat(E) then
12                     return Fail_com      /* It's not an observable multi-trace  */;
13 return Pass_com;
```

5 Experiments

We implemented the approach by separating the verification of local traces (Section 4.1) and the verification of the tuple of traces against the definition of observable multi-traces (Definition 6 and Section 4.2). If there are n subsystems, the global verdict $Verdict_g$ has $n+1$ verdicts $(Verdict_1, \ldots, Verdict_n, Verdict_{com})$ where for l in $[1 \ldots n]$, $Verdict_l$ is the local verdict in $\{Pass_l, Fail_l, Inconc_l\}$ associated to the lth component and where $Verdict_{com} \in \{Pass_{com}, Fail_{com}\}$ is the verdict relating to the verification of the communication policy.

In order to assess the scalability of the framework, we adopted a mutation-based approach. We first generated multi-traces that are correct by construction with respect to local analyses and communication rules. For this purpose, a global model of the system is built by simulating internal communications using one timed queue per component. Since the reception of a message can be delayed, the model specifies asynchronous communications. Then, we use the symbolic execution platform Diversity[9] to build long traces, focusing on the behaviours that complete communication scenarios as much as possible. Finally, the resulting multi-traces are directly constructed by considering a tuple made of all projections for each component. Generated multi-traces are then modified by applying some simple mutation schemas. Table 1 summaries mutation schemas we applied on a multi-trace μ to produce a set of mutated tuples of traces.

Mut. schema	Description
#1	Choose (randomly) a position in μ and insert an event ev
#2	Choose randomly an event ev in μ and delete it
#3	Choose randomly an event ev in μ and modify its data
#4	Choose randomly an event ev in μ and modify its duration
#5	Choose randomly a round-trip-commnunication (RTC) in μ and break it.

Table 1: Mutation schemas on multi-traces

Mutation schemas #1 and #3 require that added or modified events respect syntactic requirements from the system signature and concerning channels and data types. Mutation schema #5 is designed to break the key property of multi-traces, that is that time is necessarily elapsing when messages are transmitted. Let us illustrate with the observable multi-trace $\mu = (\sigma_1, \sigma_2)$ where $\sigma_1 = (\_, c_1?v_1).(1, c_2!v_2).(3, c_3?v_3)$ and $\sigma_2 = (\_, c_2?v_2).(1, c_3!v_3)$. Applying mutation schema #5 consists of breaking the so-called round-trip communication, abbreviated as the acronym RTC, $c_2!v_2 \rightarrow c_2?v_2 \rightarrow c_3!v_3 \rightarrow c_3?v_3$, for which by construction, the delay between the emission and the reception on the first component has to be strictly greater than the delay between the reception and the emission on the second component. Mutation schema #5 modifies delays between these actions so that there is an internal reception occurring before its corresponding emission is sent. A possible mutation of μ using mutation schema #5 could be $\mu' = (\sigma'_1, \sigma'_2)$ with $\sigma'_1 = (\_, c_1?v_1).(1, c_2!v_2).(1, c_3?v_3)$ and $\sigma'_2 = (\_, c_2?v_2).(2, c_3!v_3)$. While the first 4 mutation schemas do not necessarily

[9] https://projects.eclipse.org/proposals/eclipse-formal-modeling-project

create faulty multi-traces, mutation schema #5 creates by construction at least a communication fault.

The size of the PhoneX system depends on the number of clients. We consider a system with 3 caller clients, 3 called clients, 3 active sessions and a PhoneX central. In Table 2, the third column (com. checking) give the time[10] needed to solve the constraint associated to the verification of communications described in a multi-trace whose number of events is given in the first column and number of internal communications is given in the second column. The fourth column provides the time[11] needed to analyse all local traces. For each multi-trace, we generate 1000 mutated tuples of traces and we count the ratio of multi-traces that are faulty with regards to communication policy (before last column). Finally, in the last column, we give the average time to check the communication constraint of the mutated tuples. Experiments have been performed on a 3.10Ghz Intel Xeon E5-2687W working station with 64 GB of RAM on Linux Ubuntu 14.04.

Correct multi-traces generated by Diversity				1000 Mutated tuples of traces	
#events	#internal. com	com. checking	local. testing (for all traces)	#com. errors	average of com. checking
759	340	17ms	6s519ms	713	17.371ms
1587	700	28ms	21s761ms	729	27.648ms
3633	1589	49ms	1m34s178ms	800	40.934ms
6486	2830	59ms	7m5s797ms	737	60.140ms
7797	3400	69ms	10m52s378ms	722	66.315ms
9999	4357	88ms	24m14s860ms	738	80.825ms

Table 2: Experimental data for correct multitraces and their mutants

Among classical solvers, we get best results with the Yices SMT solver [5]. The efficiency of Yices for solving constraints of the form $d_i + x > d_j + y$ where x and y are concrete durations together with the fact that constructing the set of constraints from a tuple is linear explains that communication checking is more efficient than unit testing. Unit testing of subsystems is performed by the extension to unitary testing of the symbolic execution platform Diversity which is coupled with several solvers such as Yices, CVC4 or Z3. Regarding symbolic models without timed issues, functionalities (test case generation driven by test purposes, verdict computation) offered by the Diversity test extension are similar to those provided by the tool STG [4].

6 Related work

Testing timed distributed systems from models gives rise to several recent works. In [16], hypotheses are broadly the same as those adopted in this paper, namely a model for each local component, and a testing architecture constituted of independent local testers. [16] mainly focuses on the generation of test cases from a

[10] using the Yices SMT solver [5].
[11] using the CVC4 solver [2] embedded in the Diversity platform.

global model built by composing local models and queues, similar to the one that we used in Section 5. The main difference is that the correction of the system can boil down to the local correction of each component, without any verification of internal communications. In [13], testing of distributed real-time systems is based on the conformance relation tioco and considers timed automata as models. Testers can be local or global so that the testing architecture does not necessarily reflect the one of the system. The authors focus on the construction of analogue-clock and digital-clock test cases. The question of communications is supported by a compositionality result saying that correctness up to tioco is preserved by parallel composition of timed automata provided that they are input-enabled. Similarly, in [22], local testers that can exchange synchronisation messages are derived from a global timed automaton. Thus, all these works are rather interested in the issue of test case generation, assuming testing hypotheses on communications between components, while we leave aside this question to focus on the analysis of traces with almost no hypotheses on internal communications. Lastly, the use of constraint solvers is often advocated when dealing with software and system testing issues [7]. As an example of usage for the verdict computation in MBT, [12] uses SAT solvers for generating checking sequences from finite state machines.

7 Conclusion

We focus on the oracle problem for testing distributed systems against specifications. A system execution is a tuple of timed local traces, one for each location. An observation is correct iff each local trace is allowed by the corresponding specification component and the tuple of local traces defines a valid communication scenario. The oracle problem is reduced to several instances of the standard oracle problem for centralised testing plus a constraint satisfaction problem for communication. This is implemented as an orchestration coordinating several centralised verdict computations using the Diversity tool and calls to classical constraint solvers. We have carried out experiments with a central telecommunication system which have shown low computation time. Our algorithm is designed for active testing in which we run a test and then check the observation made. It would be interesting to extend it to deal with passive testing.

References

1. B. Bannour, J. P. Escobedo, C. Gaston, and P. Le Gall. Off-line test case generation for timed symbolic model-based conformance testing. In *In Proc. of Int. Conf. of Testing Software and Systems*, volume 7641 of *Lecture Notes in Computer Science*, pages 119–135. Springer, 2012.
2. C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. Cvc4. In *Proceedings of the 23rd Int. Conf. on Computer Aided Verification*, CAV'11, pages 171–177. Springer-Verlag, 2011.
3. E. Brinksma, L. Heerink, and J. Tretmans. Factorized test generation for multi-input/output transition systems. In *FIP TC6 11th International Workshop on*

Testing Communicating Systems (IWTCS), volume 131 of *IFIP Conference Proceedings*, pages 67–82. Kluwer, 1998.
4. D. Clarke, T. Jéron, V. Rusu, and E. Zinovieva. STG : a symbolic test generation tool. In *Int. Conf. on Tools and Algorithms for Construction and Analysis of Systems*, volume 2280 of *Lecture Notes in Computer Science*. Springer, 2002.
5. B. Dutertre. Yices 2.2. In *Computer-Aided Verification (CAV'2014)*, volume 8559 of *Lecture Notes in Computer Science*, pages 737–744. Springer, 2014.
6. C. Gaston, R. M. Hierons, and P. Le Gall. An implementation relation and test framework for timed distributed systems. In *In Proc. of Int. Conf. of Testing Software and Systems*, volume 8254 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2013.
7. A. Gotlieb. Constraint-based testing: An emerging trend in software testing. *Advances in Computers*, 99:67–101, 2015.
8. W. Grieskamp, N. Kicillof, K. Stobie, and Victor Braberman. Model-based quality assurance of protocol documentation: tools and methodology. *The Journal of Software Testing, Verification and Reliability*, 21(1):55–71, 2011.
9. R. M. Hierons, M. G. Merayo, and M. Núñez. *Implementation Relations for the Distributed Test Architecture*, pages 200–215. Springer Berlin Heidelberg, 2008.
10. R. M. Hierons, M. G. Merayo, and M. Núñez. *Using Time to Add Order to Distributed Testing*, pages 232–246. Springer Berlin Heidelberg, 2012.
11. Joint Technical Committee ISO/IEC JTC 1. *International Standard ISO/IEC 9646-1. Information Technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts.* ISO/IEC, 1994.
12. G.-V. Jourdan, H. Ural, H. Yenigün, and D. Zhu. Using a SAT solver to generate checking sequences. In *The 24th Int. Symposium on Computer and Information Sciences, ISCIS 2009*, pages 549–554. IEEE, 2009.
13. M. Krichen. A formal framework for black-box conformance testing of distributed real-time systems. *Int. Journal of Critical Computer-Based Systems*, 3(1/2):26–43, 2012.
14. M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *Proc. of Int. SPIN Workshop Model Checking of Software*, volume 2989 of *Lecture Notes in Computer Science*, pages 109–126. Springer, 2004.
15. M. Krichen and S. Tripakis. Conformance testing for real-time systems. *Form. Methods Syst. Des.*, 34(3):238–304, June 2009.
16. H. N. Nguyen, F. Zaïdi, and A. R. Cavalli. A framework for distributed testing of timed composite systems. In *21st Asia-Pacific Software Engineering Conference, APSEC*, pages 47–54. IEEE, 2014.
17. A. Petrenko and N. Yevtushenko. Testing from Partial Deterministic FSM Specifications. *IEEE Trans. Comput.*, 2005.
18. Ericsson Int. report. Investigation on how to integrate Diversity (MBT tool) and Titan (TTCN-3 executor) to provide an open source MBT tool chain, 2016-08-26.
19. B. Sarikaya and G. von Bochmann. Synchronization and specification issues in protocol testing. *IEEE Transactions on Communications*, 32:389–395, 1984.
20. J. Schmaltz and J. Tretmans. *On Conformance Testing for Timed Systems*, pages 250–264. Springer Berlin Heidelberg, 2008.
21. J. Tretmans. Formal methods and testing. chapter Model Based Testing with Labelled Transition Systems, pages 1–38. Springer-Verlag, 2008.
22. J. Vain, E. Halling, G. Kanter, A. Anier, and D. Pal. Automatic distribution of local testers for testing distributed systems. In *Databases and Information Systems IX - 12th Int. Baltic Conf.*, volume 291, pages 297–310. IOS Press, 2016.