An implementation relation and test framework for timed distributed systems

Christophe Gaston, Robert M. Hierons, Pascale Le Gall

An implementation relation and test framework for timed distributed systems*

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

[1] CEA LIST, Point Courrier 174, 91191, Gif-sur-Yvette, France
email: christophe.gaston@cea.fr
[2] Brunel University, Uxbridge, Middlesex, UK, UB8 3PH
email: rob.hierons@brunel.ac.uk
[3] Laboratoire MAS, Grande Voie des Vignes, 92195 Châtenay-Malabry, France
email: pascale.legall@ecp.fr

Abstract. Many systems interact with their environment at physically distributed interfaces and the distributed nature of any observations made is known to complicate testing. This paper concerns distributed testing, where a separate tester is placed at each localised interface and may only observe what happens at this interface. Most previous work on distributed model based testing has used models that are either finite state machines or input output transition systems. In this paper we define a framework for distributed testing from timed input output transition systems along with corresponding test hypotheses and a distributed conformance relation.
Keywords: distributed systems, timed systems, model based testing, symbolic input output transition systems.

1 Introduction

Most approaches to model based testing (MBT) assume that a single tester interacts with the system under test (SUT) and this tester observes all inputs and outputs. However, many systems such as web services and wireless sensor networks interact with the environment at multiple physically distributed interfaces. This has led to interest in distributed testing, where there is a separate local tester at each interface, a tester only observes events at its interface, and there is no global clock. This approach to distributed testing was formalised by ISO as the *distributed test architecture* [15]. It is known that the use of the distributed test architecture affects software testing [5, 6, 13, 18, 20, 22]. However, only recently has the effect been formalised as implementation relations for FSMs [11] and IOTSs[4] [10, 14].

* This work was partially supported by the French Program "Investissements d'Avenir" in the IRT/SystemX/FSF project and the SesamGrid project, and by the ITEA2 project openETCS.

[4] The implementation relation *mioco* [3] has also been defined for testing systems with distributed interfaces but this assumes that a single tester makes global observations.

Previous work showed that the distributed test architecture causes additional controllability and observability problems. A controllability problem is a situation where a local tester does not know when to apply an input [5]. Consider, for example, a test case where tester t_1 at interface 1 applies input $i_1?$, this should lead to output $o_1!$ at 1, and tester t_2 at interface 2 should then send input $i_2?$. Here t_2 cannot know when to send $i_2?$ since it does not observe the previous input and output. Observability problems refer to situations where one cannot distinguish between the global trace produced by the SUT and that expected despite these being different [6]. Let us suppose, for example, that the specification contains global trace σ where the response to a first $i_1?$ at interface 1 leads to output $o_1!$ at 1, and a second $i_1?$ leads to output $o_1!$ at 1 and output $o_2!$ at 2. Here, the tester at 1 expects to observe $i_1?o_1!i_1?o_1!$ and the tester at 2 expects to observe $o_2!$ and this observation is made if the SUT produces global trace σ' in which $o_2!$ is output in response to the first input rather than the second. While σ and σ' are different they have the same projections at the interfaces and so it is not possible to distinguish between them in distributed testing.

This paper explores distributed testing from systems described by means of cooperating timed input output transition systems (TIOTS): IOTS extended with time. We assume that testers have local clocks that are not synchronised but clocks progress at the same rate; it should be straightforward to adapt this to the case where the clocks can drift. As far as we are aware only two previous papers have explored the role of clocks/time in distributed testing and these consider different problems. One paper uses timestamps and bounds on clock differences to strengthen implementation relations for IOTSs [12] but did not consider timed models. A second looked at coordinating distributed testing from an FSM through the testers exchanging messages when we have bounds on message latency [16]. While this considered timed models it assumed that the model is an FSM with time and concentrated on overcoming controllability problems.

The previously defined implementation relation *dioco* for distributed testing against an IOTS compares global traces of the SUT against global traces of the specification using $\sim$ where $\sigma \sim \sigma'$ denotes σ and σ' having the same local projections. In order to ensure that the observations at the interfaces are all projections of the same global trace, it considers either global traces of the SUT that end in quiescence[5] or infinite traces of the SUT [12, 14].

This paper uses an alternative approach in which an observation is a tuple $(\sigma_1, \ldots, \sigma_n)$ where the tester at p observes σ_p and we call such an observation a *multitrace*. Instead of defining the implementation relation with some projection mechanisms, we directly use the notion of multitraces to define a new implementation relation *dtioco* for distributed testing. We also provide a compositionality result, which says that *dtioco* holds if and only if all multitraces of the SUT are such that exchanged messages respect some communication rules and also that the local projections of the SUT conform to the local components of the

[5] The SUT is quiescent if it cannot produce output without receiving input.

specification under *tioco*. This allows standard techniques for *tioco* [1, 8] to be used in distributed testing.

Having defined a new implementation relation we describe a test architecture for TIOTSs such that test cases can be denoted as multitraces. In distributed testing we have to bring together observations made by the local testers in order to determine whether the SUT has passed a test case. Solving the oracle problem mainly becomes a multitrace analysis problem and is described as a two step process, in which a test case is run and then a verdict is produced based on the set of local (timed) traces observed. We then describe how timed testing can be carried out and provide an algorithm for checking that communication rules are verified; the compositionality result tells us that we can derive a verdict using such an algorithm along with standard methods for *tioco*.

The paper is structured as follows. Section 2 defines the terminology and notation used in this paper and in Section 3 we describe TIOTSs. In Section 4, we present specifications of timed distributed systems as a collection of cooperating TIOTSs. Section 5 gives the test framework, defines the new implementation relation *dtioco* and provides an example. Section 6 then explains how distributed timed testing can be carried out. Finally, Section 7 draws conclusions and discusses possible avenues for future work.

2 Preliminaries

We will use a carrier set D (for *Duration*) isomorphic to the set of strictly positive real numbers[6]. We may use classical operations $+, - : D \times D \to D, <, \leq$: $D \times D \to bool$... on durations[7], provided by default with their usual meanings. Variables having their values in D are called *clocks*.

A *Labelled Transition Systems* (LTS) $\mathbb{G}$ over a set of labels L is a triple (Q, q_{in}, T) where Q is a set of states, $q_{in} \in Q$ is the *initial state*, and $Tr \subseteq Q \times L \times Q$ is a set of transitions. For transition $tr = (q, a, q')$, also denoted $q \xrightarrow{a} q'$, $source(tr)$ stands for q, $target(tr)$ stands for q' and $act(t)$ stands for the action a. $Paths(\mathbb{G}) \subseteq T^*$ is[8] the set of *paths of* $\mathbb{G}$ which contains the empty sequence ε and all sequences of transitions $tr_1 \cdots tr_n$ with $n \geq 1$ such that $source(tr_1) = q_{in}$ and for all i satisfying $1 < i \leq n$, we have $source(tr_i) = target(tr_{i-1})$. For any p in path $Paths(\mathbb{G})$, the *trace of* p, denoted $trace(p)$, is inductively defined as ε for $p = \varepsilon$, and $act(tr).trace(p')$ for $p = tr.p'$ with tr a transition and p' a path. $Traces(\mathbb{G})$ stands for the set of traces of all paths of $Paths(\mathbb{G})$.

3 Timed Input Output Transition Systems

In this section we define Timed Input Output Transition Systems (TIOTSs) and associated notation. We then discuss and formalise as multitraces the observa-

[6] In practice, any set of values used in a constraint solver for approaching real numbers.
[7] $d_1 - d_2$ is defined if and only if $d_1 > d_2$.
[8] Given a set A, A^* denotes the set of finite sequences of elements of A, ε denotes the empty sequence, and '.' is used for concatenation.

tions that may be made in distributed testing. TIOTSs are labelled transition systems whose labels represent either Outputs, Inputs or durations.

Definition 1 (TIOTS). *A* TIOTS-signature *is a tuple* $\Sigma = (C, I, O)$ *with* $I \cap O = \emptyset$*, where* C *is a set of* channels*,* I *is a set of* inputs *and* O *is a set of* outputs*. Moreover* C *can be partitioned as* $C_{in} \coprod C_{out}$ *where* C_{in} *and* C_{out} *are a set of* input channels *and a set of* output channels *respectively. Accordingly,* I *and* O *can be partitioned as* $\coprod_{c \in C_{in}} I_c$ *and* $\coprod_{c \in C_{out}} O_c$ *respectively, where for channel* c *we have that* I_c *is the set of inputs that can be received on* c *and* O_c *is the set of outputs that can be sent through* c*. A* TIOTS over Σ *is an LTS* (Q, q_{in}, T) *over* $I \cup O \cup D$*.*

In the sequel, for any TIOTS $\mathbb{A}$ over $\Sigma = (C, I, O)$, $Sig(\mathbb{A})$ stands for Σ, $C(\mathbb{A})$ or $C(\Sigma)$ stand for C, $I(\mathbb{A})$ or $I(\Sigma)$ stand for I, and $O(\mathbb{A})$ or $O(\Sigma)$ stands for O. Moreover, in a slight abuse of notation we use Σ^* for $(I \cup O \cup D)^*$. Inputs (respectively Outputs) occuring in I_c (respectively O_c) are sometimes denoted $c?a$ (respectively $c!a$) where a is a value received (sent) through channel c. We also use $c?$ or $c!$ for simple signals received or sent through channel c. Executions of TIOTS are called *Timed Traces*, which are defined as follows:

Definition 2 (Timed Traces). *The set* $TTraces(\mathbb{A})$ *of* timed traces of the TIOTS $\mathbb{A}$ *is the smallest set that satisfies the following:*

- *for any* $\sigma \in Traces(\mathbb{A})$ *(with* $\mathbb{A}$ *viewed as a simple LTS), if* σ *is of the form* $\nabla.\sigma'$ *where*[9] $\nabla = d_1 \ldots d_n$*,* $n \geq 0$*, for all* $i \leq n$*,* $d_i \in D$*, and* σ' *is either* ε *or of the form* $a.\sigma''$ *with* $a \in I \cup O$*, then* $\sigma' \in TTraces(\mathbb{A})$*.*
- *for any* d_1*,* d_2 *and* d_3 *in* D *satisfying* $d_1 + d_2 = d_3$*, for any* σ*,* σ' *in* Σ^**,* $\sigma.d_3.\sigma' \in TTraces(\mathbb{A})$ *iff* $\sigma.d_1.d_2.\sigma' \in TTraces(\mathbb{A})$*.*

Given timed trace σ and action a in $I \cup O$, $|\sigma|_a$ will denote the number of instances of a in σ.

The first point of the definition prevents us from having timed traces beginning with durations. This is because in black box testing we cannot differentiate between the SUT not being initialised from the SUT being initialised but not having interacted with its environment. Thus, if d is a duration then we cannot distinguish between traces $d.\sigma$ and σ. In a distributed context, this problem is even more pronounced since we cannot even expect that the tester and SUT are initially synchronised via a reset. The second point says that durations can be composed and decomposed provided that cumulative sums of consecutive durations are maintained. The definition of timed traces makes no assumption on how durations are sampled in testing (it allows all possible choices).

4 Specifications of timed distributed systems

We specify systems as a collection of localized parts (described as TIOTSs) communicating through a network. In particular, each localisation l is identified by its

[9] If $n = 0$, then $\nabla = \varepsilon$.

interface, given as a TIOTS signature Σ_l. The next definition characterises communications between a collection of interfaces, that is *a system signature*. They are given as a set of consistency conditions defining tuples of local executions (timed traces) corresponding to compatible visions of some global execution:

Definition 3 (System communications). *A* system signature Σ_{Sys} *is a tuple* $(\Sigma_1, \ldots, \Sigma_n)$ *of TIOTS signatures. The set of* multitraces *over* Σ_{Sys}*, denoted* $MTraces(\Sigma_{Sys})$*, is the subset of* $\Sigma_1^* \times \ldots \times \Sigma_n^*$ *defined as follows:*

Empty Trace: $(\varepsilon \ldots \varepsilon)$ *is in* $MTraces(\Sigma_{Sys})$*,*

Inputs from the environment: *for any* $i \leq n$*,* $c \in C(\Sigma_i) \setminus \cup_{j \neq i} C(\Sigma_j)$*,* $a \in I(\Sigma_i)_c$*, and* $(\sigma_1 \ldots \sigma_i, \ldots \sigma_n) \in MTraces(\Sigma_{Sys})$ *we have* $(\sigma_1 \ldots \sigma_i.a, \ldots \sigma_n) \in MTraces(\Sigma_{Sys})$*.*

Non Blocking Outputs: *for any* $i \leq n$*,* $a \in O(\Sigma_i)$*, and* $(\sigma_1 \ldots \sigma_i, \ldots \sigma_n) \in MTraces(\Sigma_{Sys})$*, we have* $(\sigma_1 \ldots, \sigma_i.a, \ldots \sigma_n) \in MTraces(\Sigma_{Sys})$*.*

Causality of communication: *for any* $i \leq n$*,* $a \in I(\Sigma_i)_c$ *where* c *is a channel of at least two TIOTS-signatures of* Sys*, and* $(\sigma_1 \ldots, \sigma_i, \ldots \sigma_n) \in MTraces(\Sigma_{Sys})$*, let us denote by* $O_a \subseteq \{1, \ldots, n\}$ *the set of all indexes* j *such that* $a \in O(\Sigma_j)_c$*. If* $|\sigma_i|_a < \Sigma_{j \in O_a} |\sigma_j|_a$*, then* $(\sigma_1 \ldots, \sigma_i.a, \ldots, \sigma_n) \in MTraces(\Sigma_{Sys})$*.*

Consistent Time Elapsing: *for any* $d \in D$ *and* $(\sigma_1, \ldots, \sigma_n) \in MTraces(\Sigma_{Sys})$*, we have* $(\sigma'_1, \ldots, \sigma'_n) \in MTraces(\Sigma_{Sys})$ *where for any* $i \leq n$*,* σ'_i *is equal to* ε *if* $\sigma_i = \varepsilon$ *and equal to* $\sigma_i.d$ *otherwise.*

A multitrace is a tuple, each element being a sequence of inputs, outputs or durations that is an execution that may be observed on a localised interface. The multitrace whose sequences are all ε (Item *Empty Trace*) corresponds to no interaction having occurred. A multitrace can be extended by adding to any component either an input from the environment (Item *Inputs from the environment*) or an output (Item *Non Blocking Outputs*). Outputs are non-blocking, when sent to the environment and when sent to other parts of the system. Internal communications are on shared channels. A channel may be shared by an arbitrary number of localised TIOTSs. Internal communication is multicast: a message sent can be received by several recipients (all those who listen on the channel of interest). Messages are never lost but the time to reach a recipient is not quantifiable since it travels between interfaces and there is no global clock (we cannot measure it). If we focus on a thread of execution (*i.e.* a sequence in a multitrace), a message cannot be received more often than the total number of emissions of this message in the system (Item *Causality of communication*). Finally, we require that time elapses in the same way for all interfaces whose corresponding trace is not empty (Item *Consistent Time Elapsing*).

In distributed testing there is a separate localised tester at each interface and there is no global clock. Thus, we cannot make any suppositions on the different moments at which the different testers stop observing their associated interfaces. To reflect this, we accept as admissible observations multitraces made of trace prefixes, which we call observable multitraces.

Definition 4 (Observable Multitraces). *The set of observable multitraces of* $\Sigma_{Sys} = (\Sigma_1 \ldots \Sigma_n)$*, denoted* $OTraces(\Sigma_{Sys})$*, is the smallest set containing* $MTraces(\Sigma_{Sys})$ *and such that for any* a *in* Σ_i*, we have:*

$$(\sigma_1, \ldots, \sigma_i.a, \ldots, \sigma_n) \in OTraces(\Sigma_{Sys}) \Rightarrow (\sigma_1, \ldots, \sigma_i, \ldots, \sigma_n) \in OTraces(\Sigma_{Sys})$$

On each localised subsystem, the observer only observes a prefix of the whole (local) timed trace if it does not wait long enough. Now, system specifications are defined as tuples $(\mathbb{A}_1, \ldots, \mathbb{A}_n)$ of TIOTSs whose associated observable multitraces $(\sigma_1 \ldots, \sigma_i, \ldots \sigma_n)$ are those such that σ_i is a timed trace of $\mathbb{A}_i$.

Definition 5 (System). *A* system *Sys over $\Sigma_{Sys} = (\Sigma_1, \ldots, \Sigma_n)$ is a tuple $(\mathbb{A}_1, \ldots, \mathbb{A}_n)$ of TIOTS, $\mathbb{A}_i$ being defined on Σ_i ($1 \leq i \leq n$). $OTraces(Sys)$ is the set of multitraces:*

$$(TTraces(\mathbb{A}_1) \times \ldots \times TTraces(\mathbb{A}_n)) \cap OTraces(\Sigma_{Sys})$$

Each TIOTS corresponds to a view of the system from one interface. Its timed traces denote possible observations of system executions from this interface. Observable traces denote tuples of consistent views of system executions.

5 Testing Framework

5.1 A conformance relation for timed distributed systems

In this section we define our new implementation relation *dtioco*. In MBT it is normal to assume that certain test hypotheses hold [9], the most basic hypothesis being that the SUT can be described using the same formalism as the specification. We assume that the following classical test hypotheses hold.

Definition 6 (*LUT* and *SUT*). *Let $\Sigma = (C, I, O)$ be a TIOTS signature and $C' \subseteq C$. A TIOTS $\mathbb{A} = (Q, q_{in}, Tr)$ over Σ satisfies the so-called* Input Enableness *property over C' iff $\forall q \in Q, \forall c \in C', \forall a \in I_c, \exists q' \in Q, (q, a, q') \in Tr$.*

A Localized System Under Test *(LUT) over (Σ, C') is a TIOTS over Σ, satisfying the input enableness property over C'.*

A System Under Test *(SUT) over $\Sigma_{Sys} = (\Sigma_1, \ldots, \Sigma_n)$ is a tuple $(\mathbb{LS}_1, \ldots, \mathbb{LS}_n)$ such that for each $1 \leq i \leq n$, $\mathbb{LS}_i$ is an LUT over $(\Sigma_i, I(\Sigma_i) \backslash (\cup_{j \in 1..n} O(\Sigma_j)))$.*

Input-enabledness adapts the traditional hypothesis to distributed testing by requiring that the system is input-enabled on its public interface made of channels shared with the environment. We base our new conformance relation on *tioco* [2, 8, 17, 21]. In fact, we use a slightly modified version of *tioco* since, as stated in Definition 2, our timed traces start with an input or an output (we remove durations occuring at the beginning of traces).

Definition 7 (tioco). *Let $\mathbb{LS}$ be an LUT and $\mathbb{A}$ a TIOTS both defined on the same signature (C, I, O).* $\mathbb{LS}$ conforms to $\mathbb{A}$, *denoted $\mathbb{LS}$ tioco $\mathbb{A}$, if and only if for any σ in $TTraces(\mathbb{A})$ and r in $O \cup D$, we have:*

$$\sigma.r \in TTraces(\mathbb{LS}) \Longrightarrow \sigma.r \in TTraces(\mathbb{A})$$

Our conformance relation for distributed systems is an extension of *tioco* to observable multitraces, except that observable multitraces introduce some constraints (Definition 3), typically on internal receptions that should be preceded by internal emissions. However, it may happen that an observation of an SUT $\mathbb{S}$ does not satisfy those constraints. For example, the network might create a spurious message in a channel between localised systems, a localised system $\mathbb{LS}_i$ receiving a message (input) on a channel c that connects it to another localised system $\mathbb{LS}_j$ without $\mathbb{LS}_j$ sending this message. Let us note that there exists no specification to which such systems conform according to Definition 8 since they do not meet consistency conditions of Definition 3. Although our implementation relation will consider such behaviours to be erroneous, we cannot assume that the possible executions of the SUT are in $OTraces(\Sigma_{Sys})$. In fact, we will only suppose that each local execution is a timed trace of some localised system $\mathbb{LS}_i$ under test. In the sequel any SUT $\mathbb{S} = (\mathbb{LS}_1, \ldots, \mathbb{LS}_n)$ has a set $Sem(\mathbb{S}) \subseteq TTraces(\mathbb{LS}_1) \times \ldots \times TTraces(\mathbb{LS}_n)$ that denotes the set of all observations that can be made in testing; this allows the cloud to introduce messages on channels as discussed earlier. We further suppose, as in the case of observable traces, that for any a we have that: $((\sigma_1, \ldots, \sigma_i.a, \ldots, \sigma_n) \in Sem(\mathbb{S})) \Rightarrow ((\sigma_1, \ldots, \sigma_i, \ldots, \sigma_n) \in Sem(\mathbb{S}))$.

Definition 8 (dtioco). *Let $\mathbb{S} = (\mathbb{LS}_1, \ldots, \mathbb{LS}_n)$ and $Sys = (\mathbb{A}_1, \ldots, \mathbb{A}_n)$ be resp. an SUT and a system both on the signature $\Sigma_{Sys} = (\Sigma_1, \ldots, \Sigma_n)$.*
$\mathbb{S}$ conforms to Sys, denoted $\mathbb{S}$ dtioco Sys, if and only if $Sem(\mathbb{S}) \subseteq OTrace(\Sigma_{Sys})$ and for any $(\sigma_1 \ldots, \sigma_i, \ldots, \sigma_n) \in OTraces(Sys)$ and $r \in (\cup_{i \leq n} O(\Sigma_i)) \cup D$, we have:

$$(\sigma_1 \ldots, \sigma_i.r, \ldots \sigma_n) \in Sem(\mathbb{S}) \Longrightarrow (\sigma_1 \ldots, \sigma_i.r, \ldots \sigma_n) \in OTraces(Sys)$$

The first part of the definition requires that any observation of the SUT is valid. The second part follows an approach similar to *tioco* in that it says that for any observable multitrace $(\sigma_1 \ldots \sigma_i, \ldots, \sigma_n)$ of the specification, every possible next observation of the SUT after $(\sigma_1 \ldots \sigma_i, \ldots, \sigma_n)$ is also an observation that might be made after $(\sigma_1 \ldots \sigma_i, \ldots, \sigma_n)$ in the specification.

In the sequel, we introduce a compositionality result that allows us to reuse testing algorithms dedicated for *tioco* in a distributed system testing process. We begin by introducing a definition allowing us to identify which part of an LUT is stimulated in a distributed SUT.

Definition 9. *Let $\mathbb{S} = (\mathbb{LS}_1, \ldots, \mathbb{LS}_n)$ be an SUT with $\mathbb{LS}_i = (Q_i, q_i, Tr_i)$ for i in $1..n$. The* projection of $\mathbb{LS}_i$ on $\mathbb{S}$, *denoted $\mathbb{LS}_i|_{\mathbb{S}}$ is the TIOTS (Q_i, q_i, Tr'_i) where Tr'_i is the subset of Tr_i that contains all transitions tr such that there exists a path of the form $p'.t$ in $Paths(\mathbb{LS}_i)$, a timed trace σ in $TTraces(p'.tr)$ and a tuple $(\sigma_1, \ldots, \sigma_n)$ in $Sem(\mathbb{S})$ such that $\sigma = \sigma_i$.*

The set of timed traces of $\mathbb{LS}_i|_{\mathbb{S}}$ contains all timed traces of $\mathbb{LS}_i$ that $\mathbb{LS}_i$ can produce when interacting with the other LUTs. Thus, if a tester interacts with $\mathbb{S}$ only through the channels of $\mathbb{LS}_i$, he/she interacts with a real system that may be represented by $\mathbb{LS}_i|_{\mathbb{S}}$ (except that the tester does not control inputs received

on channels between LUTs). By construction, $\mathbb{LS}_i|_{\mathbb{S}}$ need not be input enabled over all internal channels since all configurations over the localised part are not exercised in the context of $\mathbb{S}$.

Property 1. Let $\mathbb{S} = (\mathbb{LS}_1, \ldots, \mathbb{LS}_n)$ and $Sys = (\mathbb{A}_1, \ldots, \mathbb{A}_n)$ be resp. an SUT and a system with both having signature $\Sigma_{Sys} = (\Sigma_1, \ldots, \Sigma_n)$.

If for all i in $1..n$, $\mathbb{A}_i$ is input enabled over $C(\Sigma_i)$, the following result holds: $(\mathbb{S}\ dtioco\ Sys) \Leftrightarrow ((\forall i \leq n, \mathbb{LS}_i|_{\mathbb{S}}\ tioco\ \mathbb{A}_i) \wedge Sem(\mathbb{S}) \subseteq OTraces(\Sigma_{Sys}))$

Proof. First consider the left-to-right implication. The fact that $Sem(\mathbb{S}) \subseteq OTraces(\Sigma_{Sys})$ is part of the definition of *dtioco*.

Now let us suppose that there exists i in $1, \ldots, n$ such that $\neg(\mathbb{LS}_i|_{\mathbb{S}}\ tioco\ \mathbb{A}_i)$. If so, there exists $\sigma_i \in TTraces(\mathbb{A}_i)$ and $r \in O(\Sigma_i) \cup D$ such that $\sigma_i.r \in TTraces(\mathbb{LS}_i|_{\mathbb{S}})$ and $\sigma_i.r \notin TTraces(\mathbb{A}_i)$.

Since $\sigma_i.r \in TTraces(\mathbb{LS}_i|_{\mathbb{S}})$, there exists $(\sigma_1, \ldots, \sigma_i.r, \ldots, \sigma_n) \in Sem(\mathbb{S})$. As $Sem(\mathbb{S}) \subseteq OTraces(\Sigma_{Sys})$, we have that $(\sigma_1, \ldots \sigma_i.r, \ldots, \sigma_n) \in OTraces(\Sigma_{Sys})$. Assume that i, r and $(\sigma_1 \ldots, \sigma_i, \ldots \sigma_n)$ are chosen to be minimal and so $\sigma_j \in TTraces(\mathbb{A}_j)$ for all $1 \leq j \leq n$. Thus, since $(\sigma_1, \ldots \sigma_i, \ldots, \sigma_n) \in OTraces(\Sigma_{Sys})$, we have that $(\sigma_1 \ldots, \sigma_i, \ldots \sigma_n) \in OTraces(Sys)$. By Definition 8, since $\mathbb{S}\ dtioco\ Sys$ we have that $(\sigma_1 \ldots, \sigma_i.r, \ldots \sigma_n) \in OTraces(Sys)$. Thus, $(\sigma_1, \ldots \sigma_i.r, \ldots, \sigma_n) \in TTraces(\mathbb{A}_1) \times \ldots \times TTraces(\mathbb{A}_n)$, and so we can deduce that $\sigma_i.r \in TTraces(\mathbb{A}_i)$. This contradicts our hypothesis.

Now consider the right-to-left implication. The first condition of Definition 8 (*dtioco*) holds immediately from the hypotheses. We assume that we have been given $(\sigma_1 \ldots \sigma_i, \ldots, \sigma_n) \in OTraces(Sys)$, $r \in \Sigma_i \cup D$ and $(\sigma_1 \ldots, \sigma_i.r, \ldots \sigma_n) \in Sem(\mathbb{S})$ and are required to prove that $(\sigma_1 \ldots, \sigma_i.r, \ldots \sigma_n) \in OTraces(Sys)$.

Since $(\sigma_1 \ldots, \sigma_i.r, \ldots \sigma_n) \in Sem(\mathbb{S})$ we have that $\sigma_i.r \in TTraces(\mathbb{LS}_i|_{\mathbb{S}})$. Further, since $(\sigma_1 \ldots \sigma_i, \ldots, \sigma_n) \in OTraces(Sys)$ we know that $\sigma_i \in TTraces(\mathbb{A}_i)$. Thus, since $\mathbb{LS}_i|_{\mathbb{S}}\ tioco\ \mathbb{A}_i$, we have that $\sigma_i.r \in TTraces(\mathbb{A}_i)$. Since $Sem(\mathbb{S}) \subseteq OTraces(\Sigma_{Sys})$ we have that $(\sigma_1 \ldots \sigma_i.r, \ldots, \sigma_n) \in OTraces(\Sigma_{Sys})$. We thus have that $(\sigma_1 \ldots \sigma_i.r, \ldots, \sigma_n) \in OTraces(\Sigma_{Sys})$, $\sigma_i.r \in TTraces(\mathbb{A}_i)$ and $\sigma_j \in TTraces(\mathbb{A}_j)$ for all $1 \leq j \leq n$ with $j \neq i$. From the definition of $OTraces(Sys)$, we conclude that $(\sigma_1 \ldots, \sigma_i.r, \ldots \sigma_n) \in OTraces(Sys)$ as required. □

5.2 TIOSTSs as symbolic denotation of TIOSTs

Now, we briefly present symbolic versions of TIOSTs that will be used to illustrate our test framework with a reasonable example. Indeed, generally, for expressiveness sake, we do not directly use TIOTSs, which are appropriate to theoretically reason about conformance, but do not permit real time constraints. Here, we use TIOSTS (Timed Input Output Symbolic Transition Systems) [1,8] which are automata that have variables to abstractly denote system states (we call them data variables) and variables to capture timing constraints (we call them clocks) on system executions.

TIOSTS introduce transitions of the form $(q, \mathcal{T}, \phi, \psi, act, \rho, q')$ where (i) q and q' are states, ϕ and ψ are guards on time and data respectively, (ii) $\mathcal{T}$ is

a set of clocks to be reset to 0 when the action act occurs, (iii) act may be receptions of the form $c?x$ ($c?$) where c is a channel and x is a data variable or emissions of the form $c!t$ ($c!$) where t is a term and (iv) ρ is an assignment of data variables denoting updates of variable values. Such a transition can be executed from q if ψ holds and at any moment for which values of the clocks are such that ϕ holds (as for timed automata). At this moment, the action occurs (signals $c?$ or $c!$ or a reception of a value on x if the action is $c?x$ or emission of the current value of t if the action is $c!t$). Clocks of $\mathcal{T}$ are reset and data variables are updated according to ρ. Finally, by successively executing all consecutive transitions, one may form all timed traces of a TIOST.

5.3 Example

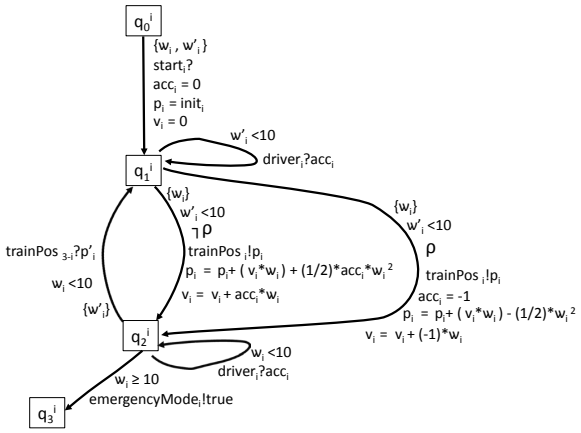

Fig. 1: Train Control System Example: TLC_i for $i = 1, 2$

Figure 1 gives a simplified model of the functional requirements of Train Local Controllers (TLCs) that forms part of the European Train Control System[10]. Similar specifications can found, e.g. in [19] and in [4]. The overall Train Control System (TCS) contains two Train Local Controllers (TLCs), one per train (say train 1 and train 2), going in the same direction on a rolling stock.

The symbol i in Figure 1 should be replaced by two possible values, 1 and 2. The system $TCS = (TLC_1, TLC_2)$ ensures that the train on the rear side automatically decreases speed as soon as the one in front of it is too close. The

[10] Interested readers can consult http://www.uic.org/spip.php?rubrique850.

relative position of trains is given by their positions, which can be accessed by consulting the value of variable p_i: if $p_1 < p_2$, then train 1 is behind train 2.

The TLC_i are automata containing 4 states (q_0^i the initial state, q_1^i, q_2^i and q_3^i), communicating through channels ($start_i$, $driver_i$, $emergencyMode_i$ for communicating with the environment, $trainPos_i$ for sending internal messages and $trainPos_{3-i}$ for receiving internal messages) and having 4 data variables (acc_i in $\{-1, 0, 1\}$ for the acceleration of the train, v_i for the speed of the train, p_i for the position value of the train, p'_i for the estimation of the position of the other train) and 2 clocks (w_i, which is reset at each emission of the position and w'_i, which is reset at each reception of the position of the second train).

TLC_i specifies the following behaviour: after an initialisation phase, the train of interest sends its position to the other train, and in return, the other train is supposed to send its position. In this loop, two consecutive communication actions are supposed to be separated by a delay of less than 10 units of time. If the remote train does not send its position on time, the local train goes into an emergency mode (not detailed here). At any moment in the loop, the driver may ask to modify the train acceleration. The new value is taken into account only if it does not affect the safety of the system (safety is threatened if condition ρ holds, that is, the distance between trains is less than the distance that can be covered by the rear train with the current acceleration). If safety is threatened, than the acceleration of the rear train is set to -1 in order to reduce its speed. Here are some examples of couples (σ_1, σ_2) with σ_i a trace of TLC_i for i in $1, 2$:

- $(\sigma_1^1 = start_1?.(2).(1).driver_1?1.(1).(2).trainPos_1!42.(2).(3).trainPos_2?300.(2).(1),$
  $\sigma_2^1 = start_2?.(1).(1).trainPos_2!300.(2).(2).trainPos_1?42.(3).(2).trainPos_2!300.(1))$
  is a multitrace with the convention that numbers between parentheses are durations, numbers without parentheses are messages and the punctuation symbol "." separates durations and actions. Note that train 1 is the rear train and train 2 does not move. This multitrace corresponds to a situation in which all local testers have observed until the end of the behaviour of TLC_i: in particular, all receptions have been preceded by an emission, and except the first duration (2) occurring in σ_1^1, all following durations (1).(1).(2).(2).(3).(2).(1) are the same in both traces (time elapsing property).
- $(\sigma_1^2 = start_1?.(2).(1).driver_1?1,$
  $\sigma_2^2 = start_2?.(1).(1).trainPos_2!300.(2).(2).trainPos_1?42.(3).(2).trainPos_2!300.(1))$
  is such that σ_1^2 is a prefix of σ_1^1 and $\sigma_2^2 = \sigma_2^1$. (σ_1^2, σ_2^2) is thus an observable multitrace. The observation of TLC_1 is stopped just after the driver asked to accelerate. Since the tester at TLC_2 observes for longer, TLC_2 receives the value 42. So, TLC_1 sent its position (42), but the observer associated with TLC_1 did not wait long enough to record this action.
- $(\sigma_1^3 = start_1?.(2).(1).driver_1?1.(1).(2).trainPos_1!42.(1).trainPos_2?300.(1).$
  $trainPos_1!60.(1).TrainPos_2?300,$
  $\sigma_2^3 = start_2?.(1).(1).trainPos_2!300.(2).(2).trainPos_1?42.(3).(2).trainPos_2!300.(1))$
  does not constitute an observable multitrace. Indeed, in σ_1^3, $trainPos_1!42$ is emitted and 3 units of time later, $trainPos_2?300$ is received a second

time. This reception corresponds to the second emission $trainPos_2!300$ in σ_2^3. Now, in σ_2^3, the second emission $trainPos_2!300$ occurs 5 units of time after the reception $trainPos_1?42$. This contradicts the time elapsing property: indeed, for TLC_1, time elapses of only 3 units of time between emission of 42 and reception of 300, but, meanwhile, for TLC_2, it elapses for at least 5 units of time since it comprises the duration separating the reception of 42 and the emission of 300.

6 Implementing Distributed Timed Testing

This section describes how testing for *dtioco* can be carried out in a manner that reflects our underlying assumptions and also utilises our compositionality result.

6.1 Architecture

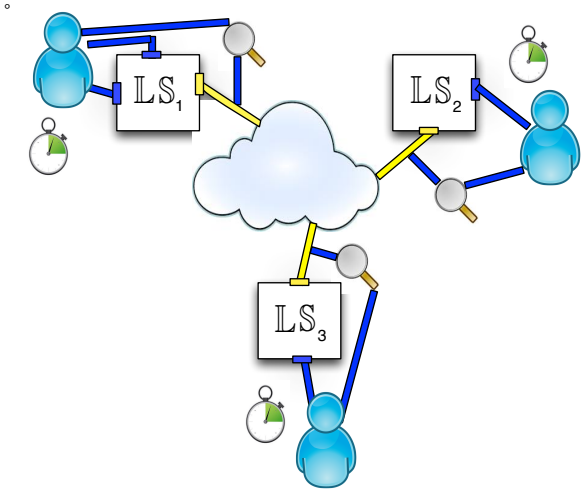

Fig. 2: Distributed testing Architecture

Figure 2 illustrates the architecture used. The SUT $\mathbb{S}$ is composed of LUTs $\mathbb{LS}_1$, $\mathbb{LS}_2$ and $\mathbb{LS}_3$. Each LUT $\mathbb{LS}_i$ has channels connected to the environment (dark connections) and internal channels to exchange values with other LUTs $\mathbb{LS}_i$ (light connections). A tester $\mathbb{T}_i$ is associated with each LUT $\mathbb{LS}_i$ and $\mathbb{T}_i$ may control inputs and observe outputs occurring on channels connected to the environment. The tester may also observe values sent through internal channels (represented by the magnifying glasses). Each $\mathbb{LS}_i$ executes in a centralised way, so that the local tester can observe the order of actions occurring on its channels and can measure durations between consecutive actions. Therefore, behaviours observed by each $\mathbb{T}_i$ can be viewed as timed traces and may be analysed with

respect to the set of timed traces of the model specifying the LUT. We cannot directly combine the timed traces observed at different LUTs since there is no global clock. Internal communications, represented by a cloud, are observed twice, at the emission and at the reception, by different testers. Recall, however, that we assume that all testers use clocks progressing at the same rate.

6.2 Process

Consider SUT $\mathbb{S} = (\mathbb{LS}_1, \ldots, \mathbb{LS}_n)$ with tester $\mathbb{T}_i$ at each $\mathbb{LS}_i$ (i in $1..n$), that we want to test against system $Sys = (\mathbb{A}_1, \ldots, \mathbb{A}_n)$. Based on Definition 8, and Property 1, we see that any fault of the whole system can be identified either at the level of one LUT, or at the level of internal communications: $(\mathbb{S}\ dtioco\ Sys) \Longleftrightarrow ((\forall i \leq n, \mathbb{LS}_i|_{\mathbb{S}}\ tioco\ \mathbb{A}_i) \wedge Sem(\mathbb{S}) \subseteq OTraces(\Sigma_{Sys}))$

So, if a local tester $\mathbb{T}_i$ exhibits timed trace σ_i contradicting $\mathbb{LS}_i|_{\mathbb{S}}\ tioco\ \mathbb{A}_i$, or if $(\sigma_1, \ldots, \sigma_n)$ (σ_i observed by tester $\mathbb{T}_i$) does not form an observational trace, then ($\mathbb{S}\ dtioco\ Sys$) does not hold. Moreover, as these conditions are sufficient to ensure that $\mathbb{S}\ dtioco\ Sys$, this suggests a two step testing process:

(1) **Timed unitary testing of each $\mathbb{LS}_i$ w.r.t. $\mathbb{A}_i$.** Tester $\mathbb{T}_i$ associated with $\mathbb{LS}_i$ checks that the timed trace it observes is allowed by $\mathbb{A}_i$ under *tioco*. If one of the testers reveals an error, then ($\mathbb{S}\ dtioco\ Sys$) does not hold;

(2) **Testing of internal communications.** All the $\mathbb{T}_i$ keep track of the localised timed traces σ_i observed. Tuple $(\sigma_1, \ldots, \sigma_n)$ of timed traces is analysed to check whether or not it constitutes an observational trace. If not, then ($\mathbb{S}\ dtioco\ Sys$) does not hold.

For the first step, the test execution process corresponds to classical unitary testing since $\mathbb{T}_i$ interacts with $\mathbb{LS}_i$ in the context of the whole system (modeled as $\mathbb{LS}_i|_{\mathbb{S}}$). The only slight difference is that inputs on internal channels are not controlled by the tester but by the remaining part of the system (the $\mathbb{LS}_j$ with $j \neq i$) and the cloud network. This kind of test architecture has already been addressed, typically in the context of *orchestrations*, a particular class of web services compositions. An orchestrator can be seen as a main localised module that orchestrates exchanges between a user and some web services. In [7], we proposed an (untimed) algorithm to test an orchestrator in the context of the system in which they are plugged. In such a test architecture, the orchestrator receives inputs from a user and web services: typically, the latter are not controlled by the tester. An orchestrator is an LUT and a whole orchestration can be seen as a special case of systems addressed in this paper. The algorithm in [7] can be adapted to be used in Step (1) to incorporate timed aspects, for example by considering the on-line test generation algorithm given in [8], or an off-line alternative [1] and by adapting them in order to ignore any duration that occurs before the first action of a timed trace. In fact, in our test framework, the off-line approach should be preferred, since for the moment, we have not addressed the question of test case generation, but focus mainly on the oracle problem. So, as we only need an algorithm for analysing trace conformance, it can be obtained for free from an off-line testing architecture.

Due to the lack of space, we do not fully present Step (1), and rather focus on the analysis of the system multitraces to check the observational trace property, and thus to achieve Step (2).

6.3 Observational Multitrace Checking

The algorithm (Figure 3) checking the observable multitrace property follows the points of Definitions 3 and 4. In the sequel we assume that any duration observed in any timed trace consists of sequences of 1 unit (which implies that we can see all delays between communication actions as integers). This is exactly what is done in practice in a timed testing process, when using a clock to measure time between actions in a testing process (the clock itself imposes the basic delay defining the unity). The main idea is to store a multitrace $(\sigma_1^c, \ldots, \sigma_n^c)$ that is observed at the end of a run, to read it from the beginning and to store the elements already read in a multitrace $ot = (\mu_1, \ldots, \mu_n)$ while keeping the elements still to be read in a multitrace $mt = (\sigma_1, \ldots, \sigma_n)$. The algorithm $check(mt, ot)$ ends with success (**return** $True$) when the current multitrace $mt = (\sigma_1, \ldots, \sigma_n)$ to be analysed is the empty multitrace $(\varepsilon, \ldots, \varepsilon)$ (line 5) and the read multitrace $ot = (\mu_1, \ldots, \mu_n)$ corresponds to the complete initial multitrace $(\sigma_1^c, \ldots, \sigma_n^c)$. There are two ways of reading a multitrace $(\sigma_1, \ldots, \sigma_n)$: either there exists a trace σ_i beginning with an action a_i (Case (1)) (line 8), or a duration d can be read on all admissible traces (Case (2)) (line 13):

Case (1) An action a_i can be read by the algorithm from σ_i (line 12), that is, removed from the trace still to be read ($replace(mt, i, tail(\sigma_i))$ and added to the trace already read ($replace(ot, i, addEnd(\mu_i, a_i))$) if one of the following conditions is fulfilled: (1) a_i is an emission towards the environment or other subsystems ($isOuptut(a_i)$), (2) a_i is a reception from the environment ($isInputEnv(a_i)$), (3) a_i is a reception of a message m on the channel c coming from one of the other subsystems ($isInputInt(a_i)$) and the number of occurrences of a_i in μ_i (elements already read by the algorithm for the subsystem $\mathbb{LS}_i$) is strictly less than the number of emissions already read by the algorithm, i.e. the number of $c!m$ occurring in $(\mu_1, \ldots, \mu_n)$ ($Nb\_I(\mu_i, a_i) < Nb\_O(ot, a_i)$), provided that none of the subsystems $\mathbb{LS}_j$ $(i \neq j)$ that can emit on the channel c has a trace fully read, i.e. $\forall j \mid \mathbb{LS}_j$ can emit on c $, \sigma_j \neq \varepsilon$, and lastly, (4) a_i is a reception of a message m on the channel c coming from one of the other subsystems and there exists a subsystem $\mathbb{LS}_j$ $(i \neq j)$ that can emit on the channel c whose trace is already fully read, i.e. $\exists j \mid \mathbb{LS}_j$ can emit on c $, \sigma_j = \varepsilon$.
In the algorithm, the predicate $FullR(Chan(a_i), mt)$ is $True$ when for all j such that $\mathbb{LS}_j$ can emit on $Chan(a_i)$, σ_j (occurring in $mt = (\sigma_1, \ldots, \sigma_n)$) is equal to ε. The predicate is used both for subcases (3) and (4).

Case (2) A duration $d = 1$ can be read by the algorithm if one of the non empty traces σ_i starts with a duration $d_i > 0$ and if for all traces σ_i starting with an action, the reading of the trace has not been started, i.e. $\mu_i = \varepsilon$ ($TimeElapsing(mt, ot)$). In this case, the duration d is subtracted from all

durations d_i occurring at the beginning of traces σ_i (d_i is simply removed if $d_i = 1$) and added to the corresponding μ_i ($time\_elapse\_of\_1(mt, ot)$).

If the reading cannot be continued until reaching the empty multitrace, then the initial multitrace does not meet the targeted observable multitrace property ($check(mt, ot)$ returns the value $False$, initialised at line 7). As the underlying principle consists of considering all possible configurations for interleaving emissions and receptions of different subsystems, its complexity is clearly high. However, as the algorithm is applied only once the local traces are completely stored (off-line algorithm), a good efficiency is not of primary necessity.

Algorithm 1: Checking of the observable multitrace property

```
1  check(mt, ot):        (* initial call : check((σ_1^c, ..., σ_n^c), (ε, ..., ε)) *)
2  (σ_1, ..., σ_n) = mt
3  (μ_1, ..., μ_n) = ot
4  if mt = (ε, ..., ε) then
5      return True
6  else
7      Cond = False
8      for i in [1, .., n] do
9          if notEmpty(σ_i) then
10             a_i = first(σ_i)
11             if isOutput(a_i) or isInputEnv(a_i) or
               (isInputInt(a_i) and Nb_I(μ_i, a_i) < Nb_O(ot, a_i) and not(FullR(Chan(a_i), mt)))
               or (isInputInt(a_i) and FullR(Chan(a_i), mt)) then
12                 Cond =
                   Cond or check(replace(mt, i, tail(σ_i)), replace(ot, i, addEnd(μ_i, a_i)))
13     if TimeElapsing(mt, ot) then
14         (mt', ot') = time_elapse_of_1(mt, ot)
15         Cond = Cond or check(mt', ot')
16     return Cond;
```

Fig. 3: Checking Observable Trace property

7 Conclusions

There has been growing interest in distributed testing where the SUT has physically distributed interfaces, there is a separate tester at each interface and a tester only observes the interactions at its interface. This paper extends previous work by investigating distributed testing from specifications based on TIOTSs that interact through an implicit distributed protocol. We assume that the SUT and specification are both composed of separate components at the interfaces and that the sending and receiving of messages between components is observed

during testing. Components themselves are described as TIOTS. We define the semantics of such systems as a sets of tuples containing a local trace for each interface. Such tuples, called observational multitraces, have to respect consistency conditions ensuring that all local traces together reflect correct interactions between components. Having defined observations, we defined an implementation relation *dtioco* for distributed testing. This implementation relation is an extension of *tioco* (timed version of *ioco*). It captures two things: tuples resulting from distributed interactions with the system under test are valid (*i.e.* are observational multitraces), and all reactions of the system under test after a specified multitrace should also be specified.

We also provide a compositionality result, which shows that an SUT conforms to a specification under *dtioco* if and only if all of the observations of the SUT that can be made are valid and the local projections of the SUT conform to the corresponding components of the specification under *tioco*. This result allows us to reuse techniques developed for *tioco*. We then describe how testing can be implemented and give an algorithm that checks that an observation is valid.

Since this is the first work to define an implementation relation for distributed timed testing, there are several lines of future work. First, there is a need to define and implement suitable test generation algorithms. In particular, it is necessary to define distributed test purposes, and to find test generation strategies to drive system executions so that they follow those test purposes. Recent work has shown that for untimed systems it is undecidable whether there is a distributed test case guaranteed to force a model M into a given state s or to distinguish two states and that this holds even if we restrict M to being a deterministic finite state machine (DFSM) [13]. Thus, general test generation problems are likely to be undecidable but we will investigate conditions under which they are decidable. Such test generation problems become tractable if we restrict attention to controllable test cases and DFSM models [11] and so it would be interesting to investigate notions of controllability for timed models. Second, we assume that local clocks progress at the same rate; it should be possible to generalise the results to the case where the clocks can drift but, for example, we have upper bounds on the rate of drift. We also intend to consider the case where the sending and receiving of internal messages are hidden.

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 *Int. Conf. ICTSS*. Springer, 2012.
2. H.C. Bohnenkamp and A. Belinfante. Timed Testing with TorX. In *Proc. of Int. Conf. Formal Methods Europe (FM)*. Springer, 2005.
3. Ed Brinksma, Lex Heerink, and Jan 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. A. Cavalli C. Andres, N. Yevtushenko. On modeling and testing the european train control system. Technical Report TechRca 14-03-2013, Telecom Sudparis, 2013.

5. R. Dssouli and G. von Bochmann. Error detection with multiple observers. In *Protocol Specification, Testing and Verification V*, pages 483–494. Elsevier Science (North Holland), 1985.
6. R. Dssouli and G. von Bochmann. Conformance testing with multiple observers. In *Protocol Specification, Testing and Verification VI*, pages 217–229. Elsevier Science (North Holland), 1986.
7. J.P. Escobedo, C. Gaston, P. Le Gall, and A. R. Cavalli. Testing web service orchestrators in context: A symbolic approach. In *Int. Conf. SEFM*. IEEE, 2010.
8. J.P. Escobedo, C. Gaston, and P. Le Gall. Timed Conformance Testing for Orchestrated Service Discovery. In *Proc. of Int. Conf. Formal Aspects of. Component Software (FACS)*. Springer, 2011.
9. M. C. Gaudel. Testing can be formal too. In *6th International Joint Conference CAAP/FASE Theory and Practice of Software Development (TAPSOFT'95)*, volume 915 of *Lecture Notes in Computer Science*, pages 82–96. Springer, 1995.
10. R. M. Hierons, M. G. Merayo, and M. Núñez. Implementation relations and test generation for systems with distributed interfaces. *Distributed Computing*, 25(1):35–62, 2012.
11. R. M. Hierons and H. Ural. The effect of the distributed test architecture on the power of testing. *The Computer Journal*, 51(4):497–510, 2008.
12. R.M. Hierons, M. G. Merayo, and M. Núñez. Using time to add order to distributed testing. In *18th International Symposium on Formal Methods (FM 2012)*, volume 7436 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 2012.
13. Robert M. Hierons. Reaching and distinguishing states of distributed systems. *SIAM Journal on Computing*, 39(8):3480–3500, 2010.
14. Robert M. Hierons, Mercedes G. Merayo, and Manuel Núñez. Implementation relations for the distributed test architecture. In *20th IFIP TC 6/WG 6.1 International Conference on the Testing of Software and Communicating Systems (TestCom/FATES 2008)*, volume 5047 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2008.
15. 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.
16. Ahmed Khoumsi. A temporal approach for testing distributed systems. *IEEE Transactions on Software Engineering*, 28(11):1085–1103, 2002.
17. M. Krichen and S. Tripakis. Black-box time systems. In *Proc. of Int. SPIN Workshop Model Checking of Software*. Springer, 2004.
18. G. Luo, R. Dssouli, and G. v. Bochmann. Generating synchronizable test sequences based on finite state machine with distributed ports. In *The 6th IFIP Workshop on Protocol Test Systems*, pages 139–153. Elsevier (North-Holland), 1993.
19. R. Meyer, J. Faber, J. Hoenicke, and A. Rybalchenko. Model checking duration calculus: a practical approach. *Formal Asp. Comput.*, 20(4-5):481–505, 2008.
20. B. Sarikaya and G. v. Bochmann. Synchronization and specification issues in protocol testing. *IEEE Transactions on Communications*, 32:389–395, April 1984.
21. J. Schmaltz and J. Tretmans. On Conformance Testing for Timed Systems. In *Proc. of Int. Conf. FORMATS*. Springer, 2008.
22. H. Ural and Z. Wang. Synchronizable test sequence generation using UIO sequences. *Computer Communications*, 16(10):653–661, 1993.