Using Time to Add Order to Distributed Testing[⋆]

Robert M. Hierons[1], Mercedes G. Merayo[2], and Manuel Núñez[2]

[1] Department of Information Systems and Computing, Brunel University
Uxbridge, Middlesex, UB8 3PH United Kingdom,
rob.hierons@brunel.ac.uk

[2] Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Madrid, Spain,
mgmerayo@fdi.ucm.es,mn@sip.ucm.es

Abstract. Many systems interact with their environment at physically distributed interfaces called ports. In testing such a system we might use a distributed approach in which there is a separate tester at each port. If the testers do not synchronise during testing then we cannot always determine the relative order of events observed at different ports and corresponding implementation relations have been developed for distributed testing. One possible method for strengthening the implementation relation is for testers to synchronise through exchanging coordination messages but this requires sufficiently fast communications channels and can increase the cost of testing. This paper explores an alternative in which each tester has a local clock and timestamps its observations. If we know nothing about how the local clocks relate then this does not help while if the local clocks agree exactly then we can reconstruct the sequence of observations made. In practice, however, we are likely to be between these extremes: the local clocks will not agree exactly but we have assumptions regarding how they can differ. This paper explores several such assumptions and derives corresponding implementation relations.

1 Introduction

Testing is the most widely used method to increase the confidence regarding the correctness of software systems. Testing has traditionally been a manual activity. This characteristic strongly increases the cost of complex software systems, where testing might take up to 50% of the project budget. As a result, there has been increasing interest in the development of techniques to automate, as much as possible, the different testing activities. One important such approach is to use formal testing methods [1, 2]. In the context of the integration of formal methods

⋆ Research partially supported by the Spanish MICINN project TESIS (TIN2009-14312-C02-01) and the UK EPSRC project Testing of Probabilistic and Stochastic Systems (EP/G032572/1). This research was carried out while the second author was visiting Brunel University supported by a grant *Fundación Caja Madrid para movilidad de profesores de universidades públicas de Madrid.*

and testing it is important to define suitable *implementation relations*, that is, formal ways to express what it means for a system to be correct with respect to a specification. Currently, the *standard* implementation relation is **ioco** [3], a well-established framework where a system under test (SUT) is correct with respect to a specification if for every sequence of actions σ that both the SUT and the specification can produce, we have that the outputs that the SUT can show after performing σ are a subset of those that the specification can show.

Many systems interact with their environment at physically distributed ports. Examples of such systems include communications protocols, web-services, cloud systems and wireless sensor networks. Users perceive these systems as black-boxes and user requirements are thus expressed at this level: users are not interested in the internal structure of a system, only in whether it delivers the services they require. In testing such systems we place a tester at each port and we are then using a distributed test architecture [4]. The use of the distributed test architecture can have a significant impact on testing and this topic has received much attention. Much of this work has concerned controllability problems, where the observations of the tester at a port p are not sufficient for it to know when to supply an input. There has also been interest in observability problems, where it is impossible to reconstruct the real order in which events were produced at different ports. A different line of work involves providing implementation relations that appropriately capture the special characteristics of the distributed test architecture. The underlying assumption in **dioco** [5], an extension of **ioco** to the distributed setting, is that we cannot compare global traces, obtained at different ports, by using equality. The idea is that if a trace is a reordering of another one where the order of events at each port has been preserved, then these two traces are indistinguishable in a distributed framework and therefore must be considered equivalent. The **dioco** framework reflects the situation in which separate agents interact with the SUT, these agents record their observations but we cannot know the causalities between events observed by different agents. However, sometimes we wish to use a framework where it is possible to establish information regarding causalities between events observed at different ports through the testers at these ports exchanging messages [6, 7]. In particular, if the testers can exchange synchronisation messages with an external agent then it is possible to use such messages to establish the exact order in which events occurred [8]. However, the assumption that the testers can synchronise (effectively, message exchange takes no time) does not seem appropriate if the testers are physically distributed.

This paper considers an alternative perspective to providing additional information regarding the causality between actions performed at different ports. We use time information: if we label actions with the time when they were observed then we can obtain additional information regarding the order in which they occurred. We can consider two possibilities to include time information in the distributed test architecture. The first one assumes the existence of a global clock. However, usually we cannot assume that there exists a global clock. We therefore consider a weaker assumption under which there is a local clock at each port. But, how do these clocks work? If we assume that the clocks are *perfect*, then **dioco** and **ioco** almost coincide. Nevertheless, this is again a very unrealistic assumption. If we make no assumptions regarding how the times given by these local clocks relate, then they add nothing and so we have **dioco**. This paper investigates different assumptions regarding how the local clocks relate and the corresponding implementation relations.

The use of timestamps to decorate actions is not new [9, 10] and the problems concerning the synchronisation of different clocks has also been studied [11]. Moreover, it has been shown that the use of timestamps has limitations since not all the causality relations can be captured [12]. Adding timestamps to actions is a common mechanism in formalisms such as process algebras to represent concurrent timed systems [13]. In this paper we investigate the use of timing information when testing from an input output transition system (IOTS).

The rest of the paper is structured as follows. Section 2 provides preliminary material. Sections 3 and 4 define implementation relations that correspond to different assumptions regarding how the clocks relate. Finally, in Section 5 we present our conclusions and some lines for future work.

2 Preliminaries

In this section we present the main concepts used in the paper. First, we define input output transition systems and notation to deal with sequences of actions that can be performed by a system. Then, we review the main differences between *classical* testing and testing in the distributed architecture.

2.1 Notation on sequences

Given a set A, we let A^* denote the set of finite sequences of elements of A; $\epsilon \in A^*$ denotes the empty sequence. Given a sequence $\sigma \in A^*$ and $1 \leq r \leq |\sigma|$, $\sigma_r \in A$ denotes the r-th element of σ. Finally, let $\sigma \in A^*$ and $a \in A$. We have that σa denotes the sequence σ followed by a and $a\sigma$ denotes the sequence σ preceded by a.

2.2 Input output transition systems

An input output transition system is a labelled transition system in which we distinguish between input and output. We use this formalism to define processes.

Definition 1. *Let $\mathcal{P} = \{1, \ldots, m\}$ be a set of ports. An* input output transition system *(IOTS) is defined by a tuple $s = (Q, I, O, T, q_{in})$ in which Q is a countable set of states, $q_{in} \in Q$ is the initial state, I is a countable set of inputs, O is a countable set of outputs, and $T \subseteq Q \times (I \cup O \cup \{\tau\}) \times Q$, where τ represents an internal (unobservable) action, is the transition relation. A transition (q, a, q'), also denoted by $q \xrightarrow{a} q'$, means that from state q it is possible to move to state q' with action $a \in I \cup O \cup \{\tau\}$.*

We say that a state $q \in Q$ is quiescent *if from q it is not possible to take a transition whose action is an output or τ without first receiving input. We extend T, the transition relation, by adding the transition (q, δ, q) for each quiescent state q. We say that s is* input-enabled *if for all $q \in Q$ and $?i \in I$ there is some $q' \in Q$ such that $(q, ?i, q') \in T$. We say that a system s is* output divergent *if it can reach a state from which there is an infinite path that contains only outputs and internal actions.*

The sets I and O are partitioned into sets $I_1, \ldots, I_m$ and $O_1, \ldots, O_m$ such that for all $p \in \mathcal{P}$, I_p and O_p are the sets of inputs and outputs at port p, respectively. We assume that $I_1, \ldots I_m, O_1, \ldots, O_m$ are pairwise disjoint.

We let $\mathcal{A}ct$ denote the set of observable actions, that is, $\mathcal{A}ct = I \cup O \cup \{\delta\}$. Given port $p \in \mathcal{P}$, $\mathcal{A}ct_p$ denotes the set of observations that can be made at p, that is, $\mathcal{A}ct_p = I_p \cup O_p \cup \{\delta\}$.

We let $\mathtt{IOTS}(I, O, \mathcal{P})$ denote the set of IOTSs with input set I, output set O and port set $\mathcal{P}$. Processes can be identified with its initial state and we can define a process corresponding to a state q of s by making q the initial state. Thus, we use states and process and their notation interchangeably. An IOTS can be represented by a diagram in which nodes represent states of the IOTS and transitions are represented by arcs between the nodes. In order to distinguish between input and output we usually precede the name of an input by ? and precede the name of an output by !.

In this paper, whenever we compare two elements of $\mathtt{IOTS}(I, O, \mathcal{P})$ we will assume that they have the same $\mathcal{A}ct_p$ for all $p \in \mathcal{P}$. Moreover, when we relate two IOTSs we will assume that they have the same port set. As usual, we assume that implementations are input-enabled.[3] We also consider that specifications are input-enabled since this assumption simplifies the analysis. However, it is possible to remove this restriction in our framework [5]. Next we introduce a system that will be used as a running example along the paper.

Example 1. The specification depicted in Figure 1 represents a simplified version of an online travel agency that sells products and services to customers on behalf of suppliers such as airlines, car rentals, hotels, etc. We focus on the functionality associated with the process that begins at the moment a client request that some services are booked and receives the confirmation. A client must ask for a flight ticket and a booking of a hotel room. Additionally, the customer can request either an airport transfer or to rent a car and, optionally, a day trip. The system presents five different ports that correspond to the different suppliers. All of them are connected to the central server where the information of the products related to each client is collected. We denote by TA the specification of the Travel Agency and the alphabets of the different ports are

- *Airlines*: $I_a = \{?data\_flight\}$, $O_a = \{!req\_flight\}$
- *Hotels*: $I_h = \{?data\_hotel\}$, $O_h = \{!req\_hotel\}$

[3] If an input cannot be applied in some state of the SUT, then we can assume that there is a response to the input that reports that this input is blocked.

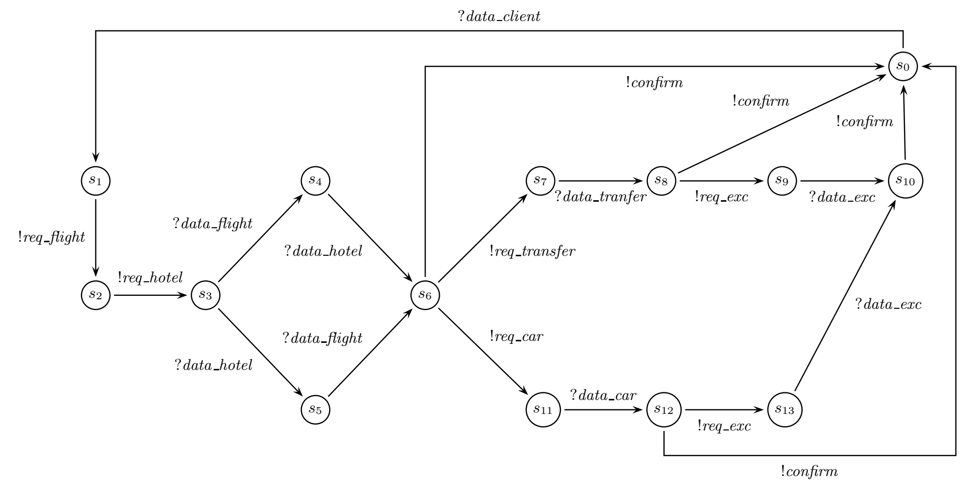

Fig. 1. Running example: travel agency.

- *Transport*: $I_t = \{?data\_transfer, ?data\_car\}$, $O_t = \{!req\_transfer, !req\_car\}$
- *Excursions*: $I_e = \{?data\_exc\}$, $O_e = \{!req\_exc\}$
- *Client*: $I_c = \{?data\_client\}$, $O_c = \{!confirm\}$

In distributed testing each tester observes only the events at its port and this corresponds to a projection of the global trace that occurred.

Definition 2. *Let $s = (Q, I, O, T, q_{in})$ be an* IOTS *with port set $\mathcal{P} = \{1, \ldots, m\}$. Let $p \in \mathcal{P}$ and $\sigma \in \mathcal{A}ct^*$ be a sequence of visible actions. We let $\pi_p(\sigma)$ denote the projection of σ onto port p and $\pi_p(\sigma)$ is called a* local trace*. Formally,*

$$\pi_p(\sigma) = \begin{cases} \epsilon & \text{if } \sigma = \epsilon \\ a\pi_p(\sigma') & \text{if } \sigma = a\sigma' \wedge a \in \mathcal{A}ct_p \\ \pi_p(\sigma') & \text{if } \sigma = a\sigma' \wedge a \in \mathcal{A}ct \setminus \mathcal{A}ct_p \end{cases}$$

Given $\sigma, \sigma' \in \mathcal{A}ct^*$ we write $\sigma \sim \sigma'$ if σ and σ' cannot be distinguished when making local observations, that is, for all $p \in \mathcal{P}(s)$ we have that $\pi_p(\sigma) = \pi_p(\sigma')$.

The equivalence relation $\sim$ among sequences is fundamental to defining our original implementation relation **dioco** [5] that we give in the next section.

In distributed testing quiescent states can be used to combine the traces observed at each port and reach a verdict. This is because we assume that quiescence can be observed and, in addition, the testers can choose to stop testing in a quiescent state. The use of distributed testers also leads to the requirement for us to compare the set of local observations made with the global traces from the specification; if we make observations in non-quiescent states then we cannot know that the observed local traces are all projections of the same global trace of the SUT and we will distinguish processes that are observationally equivalent. For example, consider the processes r and s such that r can do $!o_1!o_2$

and then can only receive input ($!o_1$ and $!o_2$ are at different ports) and s can do $!o_2!o_1$ and then can only receive input. We have that r can do $!o_1$ while s cannot. Therefore, if we consider that non quiescent traces can be used to compare processes then these two processes are not equivalent. However, in a distributed environment we cannot distinguish between these two processes if we do not have additional information (e.g. a timestamp indicating which action was performed before). Note that if a process is output-divergent then it can go through an infinite sequence of non-quiescent states, so that local traces cannot be combined. In addition, output-divergence is similar to a livelock and will generally be undesirable. We therefore restrict attention to processes that are not output divergent[4].

A *trace* is a sequence of observable actions that can be performed, possibly interspersed with τ actions, from the initial state of a process. Let s be an IOTS. Given a finite sequence of observable actions $\sigma \in \mathcal{A}ct^*$, we write $s \overset{\sigma}{\Longrightarrow} q$ if σ is a trace of s that ends in the state q. We let $\mathcal{T}r(s)$ denote the set of *traces* of s (in particular, $\epsilon \in \mathcal{T}r(s)$). Given a trace $\sigma \in \mathcal{A}ct^*$, s **after** σ denotes the set of states that can be reached from the initial state of s and after performing σ; given a state q, $\mathbf{out}(q)$ denotes the set of outputs (including quiescence) that can be performed from q possibly preceded by the performance of τ actions. The function **out** can be extended to deal with sets in the expected way, that is, $\mathbf{out}(Q') = \cup_{q \in Q'} \mathbf{out}(q)$. The interested reader is referred either to our previous work [5] or to the original **ioco** framework [3] for complete formal definitions. Next we present the standard implementation relation for testing from an IOTS [3] where information about different ports is not taken into account.

Definition 3. *Let r, s be IOTSs. We write r **ioco** s if for every $\sigma \in \mathcal{T}r(s)$ we have that $\mathbf{out}(r$ **after** $\sigma) \subseteq \mathbf{out}(s$ **after** $\sigma)$.*

2.3 Adding timestamps

We assume that there is a local clock at each port and that an event at port p is timestamped with the current time of the local clock at port p. Therefore, timed traces collected from the SUT are sequences of inputs and outputs annotated with the local time at which events were observed. We assume that actions need a minimum amount of time to be performed (this is a real assumption since we can always consider a clock cycle as this bound) and therefore it is not possible to have Zeno processes. As a consequence of this assumption, if two events are produced at the same port, then one has to be produced first and, therefore, we cannot have two events in the same port timestamped with the same value.

It is clear how a tester can timestamp inputs and outputs. In contrast, quiescence is typically observed through timeouts: the system is deemed to be quiescent if it fails to produce output for a given period of time. As a result, quiescence

[4] It is possible to consider infinite traces rather than quiescent traces [5] but this complicates the exposition.

is not observed at a particular time and so we do not include quiescence in timed traces. Naturally, corresponding untimed traces with quiescence can be produced from the timed traces. Our not including quiescence in timed traces might seem to reduce the power of testing. However, all our implementation relations will have two parts: one part considers untimed traces, which might contain quiescence, and the other part considers a set of timed traces. As a result of the first part, occurrences of δ can be safely removed from our timed traces.

Definition 4. *We consider that the time domain includes all non-negative real numbers, that is,* $\mathrm{Time} = I\!\!R_+$. *Given* $(a,t) \in \mathcal{A}ct \times \mathrm{Time}$ *we have that* $\mathrm{act}(a,t) = a$ *and* $\mathrm{time}(a,t) = t$. *Let* I *and* O *be sets of inputs and outputs, respectively. Let* $\sigma \in ((I \cup O) \times \mathrm{Time})^*$ *be a sequence of (observable action, time) pairs. We let* $\mathrm{untime}(\sigma)$ *denote the trace produced from* σ *by removing the time stamps associated with actions.*

Let $s \in \mathtt{IOTS}(I, O, \mathcal{P})$. *A* timed trace *of* s *is a sequence* $\sigma \in ((I \cup O) \times \mathrm{Time})^*$ *such that there exists* $\sigma' \in \mathcal{T}r(s)$ *such that* $\sigma' \sim \mathrm{untime}(\sigma)$, *and if there exists* $p \in \mathcal{P}$ *such that* $\mathrm{act}(\sigma_{j_1}), \mathrm{act}(\sigma_{j_2}) \in I_p \cup O_p$ *and* $j_1 < j_2$ *then* $\mathrm{time}(\sigma_{j_1}) < \mathrm{time}(\sigma_{j_2})$.

Let σ *be a timed trace of* s. *We let* $\pi_p(\sigma)$ *denote the projection of* σ *onto port* p *and* $\pi_p(\sigma)$ *is called a* timed local trace *(the formal definition of* π_p *is similar to the one given in Definition 2 for untimed traces and we therefore omit it).*

We use σ both to denote timed and untimed traces: we will state what type of sequence it represents unless this is clear from the context. We only require that timed traces can be produced by the system, that is, its untimed version is observationally equivalent to a trace of the system, and that actions at a port are sorted according to the available time information. Note that the timestamps define the exact order in which actions were produced at a given port.

2.4 Traces and event sets

A (timed or untimed) global trace defines a set of *events*, each event being either input or output (possibly with a timestamp) or the observation of quiescence. We will use information regarding timestamps to impose a partial order on the set of observed events and we therefore reason about *partially ordered sets (posets)*. In this section we only consider sequences of events that do not include quiescence since quiescence is not timestamped.

We first consider untimed traces, before generalising the definitions to timed traces. Since we wish to use a set of events we need notation to distinguish between two events with the same action and we achieve this through defining a function e from untimed traces to sets of events. We will compare traces that are equivalent under $\sim$ and so we want a representation under which ‘corresponding’ events for traces $\sigma \sim \sigma'$ have the same names; this will mean that we do not have to rename events when comparing traces. We achieve this by adding a label to each event, with the label for event a, preceded by σ', being k if this is the kth instance of a in $\sigma' a$.

Definition 5. *Let $\sigma = a_1 \dots a_n \in \mathcal{A}ct^*$ be an untimed trace. We define $e_\sigma : \mathbb{N} \longrightarrow \mathcal{A}ct \times \mathbb{N}$ as $e_\sigma(i) = (a_i, k)$ $(1 \leq i \leq n)$ if there are exactly $k-1$ occurrences of a_i in $a_1 \dots a_{i-1}$. This says that the ith element of σ is the kth instance of a_i in σ. Then we let $e(\sigma) = \{e_\sigma(1), \dots, e_\sigma(n)\}$.*

Example 2. Consider the untimed trace

$$\begin{aligned}\sigma =\, & ?data\_client!req\_flight!req\_hotel?data\_hotel?data\_flight!confirm?data\_client \\ & !req\_flight!req\_hotel?data\_flight?data\_hotel!req\_transfer?data\_transfer!confirm\end{aligned}$$

For example, the second occurrence of $?data\_client$ in σ is represented by the event $e_\sigma(7) = (?data\_client, 2)$ and the event set associated with σ is

$$e(\sigma) = \left\{ \begin{array}{l} (?data\_client, 1), (!req\_flight, 1), (!req\_hotel, 1), (?data\_hotel, 1), \\ (?data\_flight, 1), (!confirm, 1), (?data\_client, 2), (!req\_flight, 2), \\ (!req\_hotel, 2), (?data\_hotel, 2), (?data\_flight, 2), (!req\_transfer, 1), \\ (?data\_transfer, 1), (!confirm, 2) \end{array} \right\}$$

The tester at port p observes a projection of a global trace σ and so can place a total order on the events at p. We can combine these orders to obtain a partial order $<_\sigma$.

Definition 6. *Let $\sigma = a_1 \dots a_n \in \mathcal{A}ct^*$ be an untimed trace. We define the partial order $<_\sigma$ by: given $1 \leq i, j \leq n$ we have that $e_\sigma(i) <_\sigma e_\sigma(j)$ if and only if $i < j$ and there exists a port p such that $a_i, a_j \in \mathcal{A}ct_p$.*

Given a partially ordered set $(E, <)$ with $E = \{e_1, \dots, e_n\}$ we let $L(E, <)$ denote the set of linearisations *of $(E, <)$, that is, the set of sequences $e_{\rho(1)} \dots e_{\rho(n)}$ that are permutations of $e_1 \dots e_n$ and that are consistent with $<$: if $e_i < e_j$ then this ordering is preserved by the permutation $(\rho^{-1}(i) < \rho^{-1}(j))$.*

In a slight abuse of notation, given $\sigma, \sigma' \in \mathcal{A}ct^*$, with $\sigma' = a_1 \dots a_n$, we say that $\sigma' \in L(e(\sigma), <_\sigma)$ if there exists $\sigma'' \in L(e(\sigma), <_\sigma)$ and $k_1, \dots, k_n \in \mathbb{N}$ such that $\sigma'' = (a_1, k_1), \dots, (a_n, k_n)$.

Given a timed trace $\sigma = (a_1, t_1), \dots, (a_n, t_n)$, we let $e(\sigma)$ denote $e(\text{untime}(\sigma))$ and $<_\sigma$ denote $<_{\text{untime}(\sigma)}$. Given $1 \leq i \leq n$ and event $e = e_\sigma(i)$, we let $\eta_\sigma(e) = t_i$, that is, the timestamp associated with e.

Note that $(e(\sigma), <_\sigma)$ is a partially ordered set; $<_\sigma$ is irreflexive, transitive and antisymmetric. We have an interesting property, whose proof can be found in the extended version of the paper [14], that will allow us to simplify several definitions and results, since we can quantify over all traces equivalent to σ by considering the set $L(e(\sigma), <_\sigma)$.

Proposition 1. *Let $\sigma, \sigma' \in (I \cup O)^*$. We have that $\sigma \sim \sigma'$ if and only if $L(e(\sigma), <_\sigma) = L(e(\sigma'), <_{\sigma'})$.*

3 Implementation relations for clocks with imprecision bounded by a constant

In this section we study approaches to adapt our previous implementation relation **dioco** in order to take into account time information obtained from observing the behaviour of the SUT. We assume that each tester has a local clock but there is no global clock. Under **dioco** we cannot order the events at different ports but in this section we will show that more can be done if we have additional information regarding how the local clocks relate.

First, let us note that the addition of time does not modify our implementation relation **pdioco** [5]. This implementation relation assumes a framework where the agents at the ports of the SUT are entirely independent: no external agent or system can receive information regarding observations made at more than one port of the SUT. Thus, an agent at a port can observe only the local trace at that port but has no information about the observations made at the other ports. In determining whether the behaviour of the SUT is acceptable, all an agent can do is compare the observed local trace with the local traces that can be produced by the specification. Therefore, in this framework, time information cannot be used to establish causality relations between actions performed at different ports and, therefore, the addition of time does not change the implementation relation.

The implementation relation **dioco** [5] allows the set of local traces observed to be compared with the global traces of the specification. If information regarding observations made at different ports can be combined, then it is appropriate to use a stronger implementation relation.

Definition 7. *Let r, s be* IOTS*s. We write r* **dioco** *s if and only if for every quiescent trace $\sigma\delta \in \mathcal{T}r(r)$, there exists a trace $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \sim \sigma\delta$.*

As we pointed out in the previous section, the implementation relation **dioco** only considers traces that end with quiescence; we will call these *quiescent traces*. It is straightforward to prove that for the processes that we consider in this paper, input-enabled and non output divergent, r **ioco** s implies r **dioco** s but the reverse implication does not hold.

Recall that given an untimed trace σ we have the partial order $<_\sigma$ on $e(\sigma)$ and that $L(e(\sigma), <_\sigma)$ is the corresponding set of linearisations. Based on this we have the following alternative characterisation of **dioco**.

Proposition 2. *Given $r, s \in \mathtt{IOTS}(I, O, \mathcal{P})$, we have that r* **dioco** *s if and only if for every quiescent trace $\sigma\delta \in \mathcal{T}r(r)$, there exists a quiescent trace $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \in L(e(\sigma), <_\sigma)$.*

Next we present how timed traces, instead of just traces, can be used to provide a more refined implementation relation. The idea is simple: if we have timestamps then we can try to determine the order in which events were produced at different ports. Consider a specification that states that a correct system must produce output $!o_U$ at port U followed by $!o_L$ at port L. If the SUT produces $!o_L$ followed by $!o_U$ then, since these two outputs were produced at different ports, we have a correct system with respect to **dioco** because we have no means of determining that the actions were produced in the wrong order. Assume now that in addition to the actions produced at each port we are provided with timestamps. For example, let us suppose that we receive $(!o_U, 100)$ and $(!o_L, 98)$. If we have a global clock or local clocks that work perfectly, then we can claim that the SUT is not correct since $!o_L$ was performed before $!o_U$. However, if we consider a more realistic scenario where local clocks need not be synchronised, then we might consider that the difference is so small that it might be the case indeed that $!o_L$ was produced after $!o_U$ but that the clock at port U is running faster than the one placed at port L. Therefore, we need a variety of implementation relations to cope with the different alternatives.

We first assume the existence of a global clock or, equivalently, that local clocks work perfectly.

Definition 8. *Let I and O be sets of inputs and outputs, respectively. Given timed trace $\sigma = (a_1, t_1) \ldots (a_n, t_n) \in ((I \cup O) \times \mathrm{Time})^*$, $\ll_\sigma$ is the partial order on $e(\sigma)$ such that for all $1 \leq i, j, \leq n$ with $i \neq j$ we have that $e_\sigma(i) \ll_\sigma e_\sigma(j)$ if and only if $t_j > t_i$.*

*Let $r, s \in \mathcal{IOTS}(I, O, \mathcal{P})$ and $\mathcal{T} \in \mathcal{P}(((I \cup O) \times \mathrm{Time})^*)$ be a set of quiescent timed traces of r. We write r **tdioco**$(\mathcal{T})$ s if r **dioco** s and for every $\sigma \in \mathcal{T}$ there exists a quiescent trace $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \in L(e(\sigma), \ll_\sigma)$.*

Here a timed trace σ is a quiescent timed trace if untime(σ) is a quiescent trace. This new implementation relation requires **dioco** to hold since the intention is to strengthen **dioco** by including a set $\mathcal{T}$ of timed traces that have been observed in testing. As we already explained, timed traces do not contain quiescence timed traces since quiescence is not timestamped; including **dioco** ensures that the observation of quiescence is not ignored.

Example 3. Consider our running example and an SUT producing the sequence $?data\_client!req\_hotel!req\_flight$. Since $!req\_hotel$ and $!req\_flight$ were produced at different ports, the system is correct with respect to **dioco**. However, if the actions produced at each port are provided with timestamps, for example $(!req\_hotel, 40)$ and $(!req\_flight, 42)$, we can claim that the system is not correct since $!req\_hotel$ was performed before $!req\_flight$.

We now assume that there is a known value α such that the local clocks differ by at most α. We will show how this information can be used to deduce the relative ordering of events at different ports. We will express this through a partial order on the set of events observed.

Definition 9. *Let I and O be sets of inputs and outputs, respectively. Given timed trace $\sigma = (a_1, t_1) \ldots (a_n, t_n) \in ((I \cup O) \times \mathrm{Time})^*$ and $\alpha \in \mathbb{R}_+$, $\ll_\sigma^\alpha$ is the partial order on $e(\sigma)$ such that for all $1 \leq i, j, \leq n$ with $i \neq j$ we have that $e_\sigma(i) \ll_\sigma^\alpha e_\sigma(j)$ if and only if one of the following holds.*

– *There exists $p \in \mathcal{P}$ such that $a_i, a_j \in I_p \cup O_p$ and $i < j$.*

– *We have that* $t_j - t_i > \alpha$.

This says that we know that event $e_\sigma(i)$ was before event $e_\sigma(j)$ if either they were observed at the same port and $e_\sigma(i)$ was observed first or they were observed at different ports but the timestamp for $e_\sigma(i)$ was earlier than that for $e_\sigma(j)$ by more than α. In the second case, our assumption that the local clocks differ by at most α allows us to know that $e_\sigma(i)$ was observed before $e_\sigma(j)$.

This additional information, regarding the order in which events occurred, can be used to define a more refined implementation relation. This operates in the situation in which a set of timed traces has been observed, with timestamps having been produced using local clocks that can differ by at most α. As with **dioco**, we only consider quiescent traces since for these we know that the testers have observed projections of the same trace of the SUT.

Definition 10. *Let* $r, s \in \mathit{IOTS}(I, O, \mathcal{P})$, $\mathcal{T} \in \mathcal{P}(((I \cup O) \times \text{Time})^*)$ *be a set of quiescent timed traces of* r, *and* $\alpha \in \mathbb{R}_+$ *be a positive real number. We write* r $\mathbf{tdioco}_\alpha(\mathcal{T})$ s *if* r **dioco** s *and for every* $\sigma \in \mathcal{T}$ *there exists a quiescent sequence* $\sigma' \in \mathcal{T}r(s)$ *such that* $\sigma' \in L(e(\sigma), \ll_\sigma^\alpha)$.

Example 4. Let r be an SUT such that r **dioco** `TA`. Let $\mathcal{T}$ be a set of timed traces obtained from the testing process of the system and containing, in particular, the quiescent timed trace

$$\begin{aligned}\sigma =& (?data\_client, 0.5), (!req\_hotel, 1.3), (!req\_flight, 1.7), (?data\_flight, 3),\\ & (?data\_hotel, 4), (!req\_excursion, 5.8), (!req\_transfer, 6.2)\end{aligned}$$

We have that r $\mathbf{tdioco}(\mathcal{T})$ `TA` does not hold but if local clocks differ by at most $\alpha = 0.5$ units of time, then r $\mathbf{tdioco}_\alpha(\mathcal{T})$ `TA`.

We can compare implementation relations using the following relation $\sqsubseteq$.

Definition 11. *Given two implementation relations* imp_1 *and* imp_2, *we write* $imp_1 \sqsubseteq imp_2$ *if and only if for all IOTSs* r, s *we have that* r imp_2 s *implies* r imp_1 s.

Proposition 3. *Let* $\mathcal{T}$ *be a set of quiescent timed traces and* $\alpha \in \mathbb{R}_+$ *be a positive real number. We have that* **dioco** $\sqsubseteq$ $\mathbf{tdioco}_\alpha(\mathcal{T})$ *but it may be that we do not have* $\mathbf{tdioco}_\alpha(\mathcal{T}) \sqsubseteq$ **dioco**.

Proof. The first part is immediate from the definitions. For the second part it is sufficient to consider a process s that has trace $!o_1!o_2\delta$, a process r that has trace $!o_2!o_1\delta$ and a timed trace of r with timestamps that allow us to deduce that $!o_2$ was produced before $!o_1$.

If we abuse the notation slightly, to allow α to take on the value of 0, then we obtain the following result.

Proposition 4. *Given a set* $\mathcal{T}$ *of quiescent timed traces, we have that* r **ioco** s *if and only if* r $\mathbf{tdioco}_0(\mathcal{T})$ s. *Similarly, given* $\alpha \in \mathbb{R}_+$ *we have that* r **dioco** s *if and only if* r $\mathbf{tdioco}_\alpha(\emptyset)$ s.

The following gives a more general condition under which we can compare two implementation relations defined using different values for α and $\mathcal{T}$. The proof can be found in the extended version of the paper [14].

Proposition 5. *Given $\alpha_1, \alpha_2 \in \mathbb{R}_+$ and sets $\mathcal{T}_1$ and $\mathcal{T}_2$ of quiescent timed traces, we have that* $\mathbf{tdioco}_{\alpha_1}(\mathcal{T}_1) \sqsubseteq \mathbf{tdioco}_{\alpha_2}(\mathcal{T}_2)$ *if for all $\sigma_1 \in \mathcal{T}_1$ there exists $\sigma_2 \in \mathcal{T}_2$ such that $e(\sigma_2) = e(\sigma_1)$ and $\ll^{\alpha_1}_{\sigma_1} \subseteq \ll^{\alpha_2}_{\sigma_2}$.*

We now say what it means for a set of timed traces to be valid for a process r given α; these are the set of timed traces that can be produced if the clocks are within α of one another.

Definition 12. *Given $\alpha \in \mathbb{R}_+$ and timed trace σ, we say that σ is* valid for process r given α *if there exists a trace $\sigma' = a'_1 \ldots a'_n \in \mathcal{T}r(r)$ with $e(\text{untime}(\sigma)) = e(\sigma')$ such that for all $1 \le i < j \le n$ we have that $\eta_\sigma(e_{\sigma'}(i)) - \eta_\sigma(e_{\sigma'}(j)) \le \alpha$.*

This condition requires that if $e_\sigma(i)$ is before $e_\sigma(j)$ in σ' then the timestamp for $e_\sigma(i)$ is less than the timestamp for $e_\sigma(j)$ in σ or the time difference is sufficiently small for it to be possible that $e_\sigma(i)$ occurred before $e_\sigma(j)$. We now have the following result.

Proposition 6. *Given $r, s \in \mathcal{IOTS}(I, O, \mathcal{P})$, $\alpha \in \mathbb{R}_+$ and a set $\mathcal{T}$ of quiescent timed traces that are valid for r given α, then* $\mathbf{tdioco}_\alpha(\mathcal{T}) \sqsubseteq \mathbf{ioco}$.

We have seen that our new implementation relation lies between **ioco** and **dioco**: it can be more powerful than **dioco** and can be less powerful than **ioco**. In the extended version of the paper [14] we give results that explore this issue further, showing how in the limit it can approach **ioco** and also how it can be reduced to **dioco** even if we have many timed traces. Specifically, we show that if the set of timed traces parameterising our implementation relations contains appropriate instances of all the traces of the system then the stronger **ioco** implementation relation can be fully captured. We also show that *irrelevant* time values do not add distinguishing power to the collected set of traces, so that we still obtain the **dioco** relation.

4 Clocks with variable imprecision

In the previous section we defined an implementation relation based on the assumption that there is a known $\alpha \in \mathbb{R}_+$ that is an upper bound on the differences between the local clocks. However, in practice we expect there to be drift: some clocks will progress faster than others. As a result, the potential difference will grow with time. Therefore, in this section we devise implementation relations for this more general scenario.

For our next relation we assume that there is a bound α on the potential difference between the clocks at the beginning of testing and for the difference to be able to grow with time based on another value β. Even for small bounds, we think that this is a very realistic assumption. As we will see later, even though we are not able to fully capture the original ordering in which events are performed at different ports, we can still fix the occurrence of actions that were performed far apart. First we define the corresponding partial order on events.

Definition 13. *Let I and O be sets of inputs and outputs, respectively. Given timed trace $\sigma = (a_1, t_1) \ldots (a_n, t_n) \in ((I \cup O) \times \text{Time})^*$ and $\alpha, \beta \in \mathbb{R}_+$, $\ll_\sigma^{\alpha,\beta}$ is the partial order on $e(\sigma)$ such that for all $1 \leq i, j, \leq n$ with $i \neq j$ we have that $e_\sigma(i) \ll_\sigma^{\alpha,\beta} e_\sigma(j)$ if and only if one of the following holds.*

- *There exists $p \in \mathcal{P}$ such that $a_i, a_j \in I_p \cup O_p$ and $i < j$.*
- *We have that $t_j - t_i > \alpha + \beta \cdot \max(t_i, t_j)$.*

*Let $r, s \in \mathcal{IOTS}(I, O, \mathcal{P})$, $\mathcal{T} \in \mathcal{P}(((I \cup O) \times \text{Time})^*)$ be a set of quiescent timed traces of r, and $\alpha, \beta \in \mathbb{R}_+$ be positive real numbers. We write r $\mathbf{tdioco}_{\alpha,\beta}(\mathcal{T})$ s if r **dioco** s and for every $\sigma \in \mathcal{T}$ there exists a quiescent trace $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \in L(e(\sigma), \ll_\sigma^{\alpha,\beta})$.*

The next relation is a generalisation of $\mathbf{tdioco}_{\alpha,\beta}(\mathcal{T})$ where potential difference in clocks can accumulate in a non-linear way. We capture this by using an increasing function to place a bound on the relative imprecision of clocks.

Definition 14. *Let I and O be sets of inputs and outputs, respectively. Given timed trace $\sigma = (a_1, t_1) \ldots (a_n, t_n) \in ((I \cup O) \times \text{Time})^*$ and a monotonically increasing function $h : \mathbb{R}_+ \longrightarrow \mathbb{R}_+$, $\ll_\sigma^h$ is the partial order on $e(\sigma)$ such that for all $1 \leq i, j, \leq n$ with $i \neq j$ we have that $e_\sigma(i) \ll_\sigma^h e_\sigma(j)$ if and only if one of the following holds.*

- *There exists $p \in \mathcal{P}$ such that $a_i, a_j \in I_p \cup O_p$ and $i < j$.*
- *We have that $t_j - t_i > h(\max(t_i, t_j))$.*

*Let $r, s \in \mathcal{IOTS}(I, O, \mathcal{P})$, $\mathcal{T} \in \mathcal{P}(((I \cup O) \times \text{Time})^*)$ be a set of quiescent timed traces of r and $h : \mathbb{R}_+ \longrightarrow \mathbb{R}_+$ be a monotonically increasing function. We write r $\mathbf{tdioco}_h(\mathcal{T})$ s if r **dioco** s and for every $\sigma \in \mathcal{T}$ there exists a quiescent trace $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \in L(e(\sigma), \ll_\sigma^h)$.*

Example 5. Consider the specification of the travel agency depicted in Figure 1. Let r be a SUT such that its conformance to the `TA` specification with respect to **dioco** has been established. Assume that while testing r we obtained the following set $\mathcal{T}$ of timed sequences

$$tr_1 = (?data\_client, 0.5)(!req\_flight, 1)(!req\_hotel, 2)(?data\_flight, 3)(?data\_hotel, 4)(!req\_car, 5)(?data\_car, 6)$$
$$tr_2 = (?data\_client, 1.5)(!req\_hotel, 3.4)(!req\_flight, 4.4)$$
$$tr_3 = (?data\_client, 1)(!req\_hotel, 1.3)(!req\_flight, 2)(?data\_flight, 3)(?data\_hotel, 4.6)(!req\_transfer, 5.1)(!req\_excursion, 5.3)$$

The sequences tr_2 and tr_3 show that r $\mathbf{tdioco}(\mathcal{T})$ `TA` does not hold since $!req\_hotel$ was produced before $!req\_flight$. Besides, r $\mathbf{tdioco}_\alpha(\mathcal{T})$ `TA` holds if $\alpha \geq 1$, while if $\alpha < 1$ the conformance does not hold. If we assume that the difference between clocks grows with time, then for any value assigned to β when $\alpha \geq 1$ we have that r $\mathbf{tdioco}_{\alpha,\beta}(\mathcal{T})$ `TA` holds; however if, for example, $\alpha = 0.2$ and $\beta = 0.2$ then the sequence tr_3 shows that r is not correct.

Proposition 7. *Let $r, s \in \mathcal{IOTS}(I, O, \mathcal{P})$, $\mathcal{T}$ be a set of timed traces and $\alpha, \beta \in \mathbb{R}_+$ be positive real numbers. If r $\mathbf{tdioco}_\alpha(\mathcal{T})$ s then r $\mathbf{tdioco}_{\alpha,\beta}(\mathcal{T})$ s. However, we might have that r $\mathbf{tdioco}_{\alpha,\beta}(\mathcal{T})$ s but not r $\mathbf{tdioco}_\alpha(\mathcal{T})$ s.*

In order to conclude the paper, we would like to point out that if we consider single-port systems then all the timed implementation relations introduced in this paper coincide with **ioco**. The proof is easy and relies on the fact that **ioco** and **dioco** are equal for single-port, input-enabled and non output-divergent systems.

5 Conclusions and future work

Many systems interact with their environment at physically distributed interfaces, which we call ports. In distributed testing we place a separate tester at each port and the tester at port p only observes events that occur at p. As a result, it may not be possible to determine the relative order of events observed at different ports and this has led to the development of implementation relations such as **dioco** that reflect this.

This paper has explored the situation in which each tester has a local clock and adds timestamps to the observations it makes. If we have no information regarding how the local clocks relate then this does not help us. However, in practice we are likely to have some information, in the form of assumptions, regarding how much the local clocks can differ. We considered several such assumptions. In one extreme case the local clocks are known to agree and so we can reconstruct the sequence of events. However, this assumption appears to be unrealistic. We also considered the case where there is a known upper bound α on how much the clocks can differ. An alternative scenario is when there is an initial bound α and the bound on the differences between the clocks can grow linearly. We also considered the generalisation when there is a monotonically increasing function h such that at time t the differences between the clocks is at most $h(t)$. For each scenario we defined a corresponding implementation relation and we explored how these relate.

There are several possible lines of future work. First, we can integrate our approach with methods for establishing local clocks through message exchange. We might consider the case where the specification contains timing requirements. Distributed testing in the situation in which there are timing requirements is likely to be challenging, especially if there are requirements regarding the relative timing of events at different ports. In the same line, there is a need to consider the implications of time for test generation. Another direction worth investigating is to study the limits of timed implementation relations. By taking into account time we are able to distinguish processes that were undistinguishable under **dioco** but it would be interesting to explore implementation relations closer to **ioco** in the current framework. One issue that we did not consider in this paper is the computational complexity of deciding the different implementation relations, in particular, it would be important to consider the trade-off between distinguishing power and the complexity of checking whether a given relation holds.

Acknowledgements

We would like to thank the reviewers of the paper for the careful reading and for pointing out interesting lines to continue the research in this topic.

References

1. Hierons, R.M., Bogdanov, K., Bowen, J., Cleaveland, R., Derrick, J., Dick, J., Gheorghe, M., Harman, M., Kapoor, K., Krause, P., Luettgen, G., Simons, A., Vilkomir, S., Woodward, M., Zedan, H.: Using formal methods to support testing. ACM Computing Surveys **41**(2) (2009)
2. Grieskamp, W., Kicillof, N., Stobie, K., Braberman, V.: Model-based quality assurance of protocol documentation: tools and methodology. Software Testing, Verification and Reliability **21**(1) (2011) 55–71
3. Tretmans, J.: Model based testing with labelled transition systems. In: Formal Methods and Testing, LNCS 4949, Springer (2008) 1–38
4. ISO/IEC JTC 1, J.T.C.: International Standard ISO/IEC 9646-1. Information Technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts. ISO/IEC (1994)
5. Hierons, R.M., Merayo, M.G., Núñez, M.: Implementation relations and test generation for systems with distributed interfaces. Distributed Computing **25**(1) (2012) 35–62
6. Cacciari, L., Rafiq, O.: Controllability and observability in distributed testing. Information and Software Technology **41**(11–12) (1999) 767–780
7. Rafiq, O., Cacciari, L.: Coordination algorithm for distributed testing. The Journal of Supercomputing **24**(2) (2003) 203–211
8. Jard, C., Jéron, T., Kahlouche, H., Viho, C.: Towards automatic distribution of testers for distributed conformance testing. In: TC6 WG6.1 Joint Int. Conf. on Formal Description Techniques and Protocol Specification, Testing and Verification, FORTE'98, Kluwer Academic Publishers (1998) 353–368
9. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM **21**(7) (1978) 558–565
10. Jard, C., Jéron, T., Tanguy, L., Viho, C.: Remote testing can be as powerful as local testing. In: 19th Joint Int. Conf. on Protocol Specification, Testing, and Verification and Formal Description Techniques, FORTE/PSTV'99, Kluwer Academic Publishers (1999) 25–40
11. Lamport, L., Melliar-Smith, P.: Synchronizing clocks in the presence of faults. Journal of the ACM **32**(1) (1985) 52–78
12. Fidge, C.: A limitation of vector timestamps for reconstructing distributed computations. Information Processing Letters **68**(2) (1998) 87–91
13. Baeten, J., Middelburg, C.: Process algebra with timing. EATCS Monograph. Springer (2002)
14. Hierons, R.M., Merayo, M.G., Núñez, M.: Using time to add order to distributed testing (2012) Available at `http://antares.sip.ucm.es/manolo/papers/fm2012_extended.pdf`.