Noname manuscript No.
(will be inserted by the editor)

Timed Implementation Relations for the Distributed Test Architecture

Robert M. Hierons · Mercedes G. Merayo · Manuel Núñez

Received: date / Accepted: date

Abstract In order to test systems that have physically distributed interfaces, called ports, 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 this leads to new notions of correctness that have been described using corresponding implementation relations. We study the situation 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 affect the implementation relation 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 some information regarding how they can differ. We start by assuming that a local tester interacts synchronously with the corresponding port of the system under test and then extend this to the case where communications can be asynchronous, considering both the first-in-first-out (FIFO) case and the non-FIFO case. The new implementation relations are stronger than implementation relations for distributed testing that do not use timestamps but still reflect the distributed nature of observations. This paper explores these alternatives and derives corresponding implementation relations.

Keywords Model based testing · distributed systems · timed systems

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.*

Robert M. Hierons
Department of Information Systems and Computing, Brunel University
Uxbridge, Middlesex, UB8 3PH United Kingdom
E-mail: rob.hierons@brunel.ac.uk

Mercedes G. Merayo and Manuel Núñez
Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Madrid, Spain
E-mail: mgmerayo@fdi.ucm.es,mn@sip.ucm.es

1 Introduction

Complex computer systems often consist of many parts and components that interact to produce the desired result. These systems communicate 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. Due to the complexity of these systems, it is very difficult to ensure that they are correct, that is, that their behaviour is consistent with what the designers had in mind before the system was created. Therefore, it is of the utmost importance to use sound engineering techniques in order to analyse the developed systems. In this line, *testing* [33, 1] is the most widely used method to increase 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 [33]. As a result, there has been increasing interest in the development of tech-

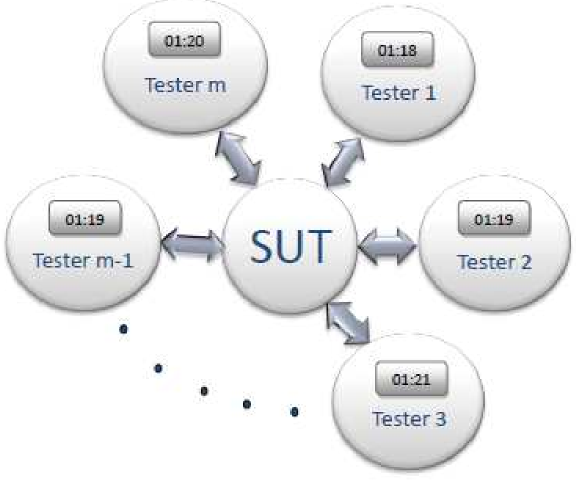

Fig. 1 Test Architecture.

niques to automate, as much as possible, the different testing activities.

One important approach to automation of testing is to use *formal testing methods* [13,30,18,17]. Formal testing methods are a type of model based testing (MBT) in which automation is based on a model of the required behaviour of the system under test (SUT) or some aspect of this required behaviour. Given such a model M there is the potential to automatically generate test cases from M, use M to direct testing, and to check that the behaviour observed in testing is consistent with M. Most MBT work focusses on testing from a finite state machine (FSM) [32,14,35] or an input output transition system (IOTS) [39,40]. While testers will often want to use richer modelling languages, MBT tools normally translate a model into either an FSM or an IOTS and use the resultant model [11,15]. In addition to the automation of the testing process [42], MBT techniques were found to be significantly more cost effective than manual testing in a recent industrial study involving hundreds of testers [15]. In the context of the integration of formal methods 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 for testing from an IOTS is **ioco** [40], a well-established framework where the 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.

In order to test systems with distributed ports we place a tester at each port and we are then using a distributed test architecture [24] (see Figure 1 for a graphical representation of this architecture). The use of a distributed test architecture can have a significant impact on testing and this topic has received much attention [37,10,5,31,38,41,23]. 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 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** [19–21], 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 [6,36]. 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 [25]. However, the assumption that the testers can synchronise (effectively, message exchange takes no time) does not seem appropriate if the testers are physically distributed. It is know that it is sometimes possible to overcome controllability and observability problems through the exchange of coordination messages between testers [8,38,6]. Almost all work in this area concerns testing from a deterministic finite state machine and it is assumed both that the input/output pair produced by a transition is atomic and that after an input/output pair the SUT waits until it receives the next input. When these assumptions hold it is possible to fully synchronise testing through the testers exchanging coordination messages. However, when these assumptions do not hold it may not be possible to synchronise testing since the exchange of coordination messages introduces delays that may not be compatible with test cases. Interestingly, it has also been suggested that the overheads introduced by the exchange of coordination messages means that approaches that use such messages do not scale to testing large-scale distributed systems and this has motivated work that aims to structure an architecture in order to reduce the number of messages required [9].

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 timestamps that give 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, typically there will not be a global clock and instead each tester has a local clock. If we know nothing about how these clocks relate then timestamps provide no additional information regarding causality. However, in practice we are likely to have some information regarding the potential differences in the times given by the local clocks. This paper investigates different assumptions regarding how the local clocks relate and the corresponding implementation relations. Initially we assume that the communication between the testers and the SUT is synchronous but we then weaken this to consider the case where the communication between the SUT and the testers can introduce a (bounded) delay. Despite the added complexity of the asynchronous approaches, these new frameworks are better suited to appropriately reflect the nature of current distributed systems. These systems usually communicate through different channels, that can transmit information at different speeds, so that an action that was produced before another one can actually be observed after the occurrence of the latter. We would like to note that, in general, timestamps are less powerful than coordination messages to reconstruct the original order in which actions where performed. However, they are an effective solution when it is not feasible to use these messages.

The use of timestamps to decorate actions is not new in formal testing, in particular, in order to generate tests in asynchronous systems [26,27]. In contrast with our work, these approaches do not consider that time information might not precisely reflect reality. Local clocks can be used to timestamp actions observed in a distributed system in order to determine information regarding the order in which actions are performed [28] and the problems concerning the synchronisation of different clocks have also been studied [29]. Moreover, it has been shown that the use of timestamps has limitations since not all the causality relations can be captured [12]. Our approach is related to the previously mentioned work on ordering of events in distributed systems. In particular, we consider both situations where the clocks allow us to determine the order between different events (a kind of *clock condition* [28]) and where this does not happen.

This paper extends our previous work [22]. In addition to providing more detailed explanations regarding the main concepts, the paper includes several new contributions. All the material concerning asynchronous communications is new and did not appear before. In fact, the new implementation relations presented in this paper for asynchronous communication are a contribution to the field of the formal testing of distributed systems, even in the absence of time information. We have included many results to compare the different implementation relations. Proofs that were omitted due to space limitations are now included in this paper. We have added a running example that helps to clarify the differences between the implementation relations presented in the paper. Finally, we have included a glossary (see Figures 2 and 3) where, in particular, all the implementation relations presented in the paper are informally described.

The rest of the paper is structured as follows. Section 2 provides preliminary material. In Section 3 we review previous work on untimed relations in the distributed architecture that will be the basis for the subsequent relations. Sections 4 and 5 define implementation relations that correspond to different assumptions regarding how the clocks relate. In Sections 4 and 5 we assume that communications are synchronous and in Sections 6 and 7 this assumption is modified to consider a framework where communication between the different entities is asynchronous. In Section 8 we briefly explain why the oracle problem is NP-complete in our setting. Finally, in Section 9 we present our conclusions and some lines for future work.

2 Preliminaries

In the following two sections we present the main concepts that we use to define the implementation relations that we study in this paper. In this section we define input output transition systems, how timestamps can be used to annotate actions performed by systems and how sequences of pairs (action, timestamp) can be used to define partially ordered sets. This later construction will simplify some of the definitions that we present in the paper.

2.1 Basic notation

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 $\sigma \in A^*$, we have that $|\sigma|$

Processes, actions and traces	
Concept	*Explanation*
IOTS Def. 1	Input Output Transition System. Labelled transition systems with a distinction between input actions (preceded by ? and grouped in a set I), output actions (preceded by ! and grouped in a set O), an internal action (denoted by τ) and an action to denote quiescence (denoted by δ). The set of visible actions, denoted by $\mathcal{A}ct$, is equal to $I \cup O \cup \{\delta\}$. Inputs and actions are usually labelled with a number denoting the port where they are performed.
trace Def. 2	Sequence of visible actions that a process can perform.
timed trace Def. 4	Sequence of pairs (visible action, time) denoting the actions performed by a process and the time when these actions were (locally) observed.

Orders between events	
Concept	*Explanation*
e_σ Def. 5	For all $1 \leq i \leq \lvert\sigma\rvert$, $e_\sigma(i)$ returns the occurrence of the symbol appearing in the position i of σ together with the number of occurrences of the symbol so far in the trace. For example, $e_{a_1 b_2 a_1 a_1 c_2 a_1}(4) = (a_1, 3)$.
$e(\sigma)$ Def. 5	Set of values generated by e_σ. For example, $e(a_1 b_2 a_1 a_1 c_2 a_1) = \{(a_1,1),(b_2,1),(a_1,2),(a_1,3),(c_2,1),(a_1,4)\}$.
$<_\sigma$ Def. 6	Two events $e_\sigma(i)$ and $e_\sigma(j)$ are related if they occur at the same port and $i < j$. For example, if $\sigma = a_1 b_2 a_1 a_1 c_2 a_1$, then $e_\sigma(2) <_\sigma e_\sigma(5)$, $e_\sigma(6) \not<_\sigma e_\sigma(4)$, $e_\sigma(2) \not<_\sigma e_\sigma(3)$.
$L(e(\sigma), <_\sigma)$ Def. 6	Linearisations. Permutations of the elements of $e(\sigma)$ consistent with $<_\sigma$. For example, if $\sigma = a_1 b_2 a_1 a_1 c_2 a_1$, then $(b_2,1)(c_2,1)(a_1,1)(a_1,2)(a_1,3)(a_1,4)$ belongs to $L(e(\sigma), <_\sigma)$ while $(a_1,2)(a_1,1)(b_2,1)(a_1,3)(c_2,1)(a_1,4)$ does not belong to $L(e(\sigma), <_\sigma)$.

Main relations between traces	
Concept	*Explanation*
$\sim$ Def. 3	Relation to compare traces in a synchronous framework. Two traces are related if all their local projections are equal. For example, $!o_1!o_2?i_1?i_2 \sim !o_2?i_2!o_1?i_1$.
$\sqsubseteq$ Def. 11	Relation to compare traces in an asynchronous FIFO framework. Intuitively, outputs in each port can be delayed, in other words, overtaken by inputs, but the order between outputs must be kept. For example, $?i_1!o_1!o'_1?i_2!o_2 \sqsubseteq !o_1!o_2!o'_1?i_1?i_2$ but $!o_2!o'_1!o_1?i_1?i_2 \not\sqsubseteq !o_1!o_2!o'_1?i_1?i_2$ and $!o_2!o'_1!o_1?i_1?i_2 \not\sqsubseteq !o_2!o'_1?i_1!o_1?i_2$.
$\sqsubseteq^N$ Def. 13	Relation to compare traces in an asynchronous non-FIFO framework. This differs from $\sqsubseteq$ because input or output can overtake one another in each port. For example, $?i_1!o_1!o'_1?i_2!o_2 \sqsubseteq^N !o_1!o_2!o'_1?i_1?i_2$, $?i_1!o_1!o'_1?i_2!o_2 \sqsubseteq^N ?i_1!o'_1!o_1?i_2!o_2$.

Fig. 2 Glossary of concepts (1/2).

denotes its length and $\sigma_r \in A$, with $1 \leq r \leq \lvert\sigma\rvert$, denotes the rth element of σ. Given a sequence $\sigma \in A^*$ 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.

Throughout this paper we let I be the set of inputs and O the set of outputs. We also let $\mathcal{P}orts = \{1, \ldots, m\}$ be the set of the names of the ports and assume that the sets I and O are partitioned into sets $I_1, \ldots, I_m$ and $O_1, \ldots, O_m$ such that for all $p \in \mathcal{P}orts$, 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. 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 addition, we will often decorate actions with a subindex indicating the port at which the action occurs. For example, $!o_1$ indicates an output at port 1. This notation should not be confused with the one used for sequences of actions: in this case, a subindex denotes, as expected, the position of the action in the sequence.

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 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 $(q, a, q') \in T$, 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 an input. We extend T to T_δ by adding 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

Implementation relations: untimed and synchronous communication	
Concept	*Explanation*
r **ioco** s Def. 8	Standard implementation relation in the non-distributed architecture [40]. An implementation r should not show behaviours that were not planned in a specification s.
r **pdioco** s Def. 9	Simple implementation relation in the distributed architecture [21]. For each port, an implementation r should not show behaviours that are not planned in a specification s. Information obtained at different ports is not combined.
r **dioco** s Def. 10	Main implementation relation in the distributed architecture [21]. An implementation r should not show behaviours that are not planned in a specification s, with the exception of the order between events performed at different ports.
r $\mathbf{tdioco}(\mathcal{T})$ s Def. 15	We assume a global clock. First, we ask for r **dioco** s. In addition, the information contained in the set of timed traces $\mathcal{T}$ is checked to ensure that the order between events at different ports is correct.
r $\mathbf{tdioco}_\alpha(\mathcal{T})$ s Def. 17	Similar to $\mathbf{tdioco}(\mathcal{T})$ but assuming that there is a local clock for each port and that local clocks can differ in at most α time units.
r $\mathbf{tdioco}_{\alpha,\beta}(\mathcal{T})$ s Def. 20	Refinement of $\mathbf{tdioco}_\alpha(\mathcal{T})$ assuming that the difference between clocks can grow but the growth is bounded by β.
r $\mathbf{tdioco}_h(\mathcal{T})$ s Def. 20	Refinement of $\mathbf{tdioco}_\alpha(\mathcal{T})$ assuming that the difference between clocks is bounded by the function h.

Implementation relations: asynchronous communication	
Concept	*Explanation*
r $\mathbf{dioco}^F s$ Def. 12	Adaption of **dioco** to deal with asynchronous FIFO communications.
r $\mathbf{tdioco}^F(\mathcal{T})$ s, r $\mathbf{tdioco}^F_\alpha(\mathcal{T})$ s, r $\mathbf{tdioco}^F_{\alpha,\beta}(\mathcal{T})$ s, r $\mathbf{tdioco}^F_h(\mathcal{T})$ s Def. 26	Adaption of the different variants of $\mathbf{tdioco}(\mathcal{T})$ to deal with asynchronous FIFO communications.
r $\mathbf{dioco}^N s$ Def. 14	Adaption of **dioco** to deal with asynchronous non-FIFO communications.
r $\mathbf{tdioco}^N(\mathcal{T})$ s, r $\mathbf{tdioco}^N_\alpha(\mathcal{T})$ s, r $\mathbf{tdioco}^N_{\alpha,\beta}(\mathcal{T})$ s, r $\mathbf{tdioco}^N_h(\mathcal{T})$ s Def. 30	Adaption of the different variants of $\mathbf{tdioco}(\mathcal{T})$ to deal with asynchronous non-FIFO communications.

Fig. 3 Glossary of concepts (2/2).

reach a state from which there is an infinite path that contains only outputs and internal actions.

We let $\mathcal{A}ct$ denote the set of *observable* actions, that is, $\mathcal{A}ct = I \cup O \cup \{\delta\}$. Given port $p \in \mathcal{P}orts$, $\mathcal{A}ct_p$ 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 $\texttt{IOTS}(I, O, \mathcal{P}orts)$ denote the set of IOTSs with input set I, output set O and port set $\mathcal{P}orts$. □

A process 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 processes 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 this paper, whenever we compare two IOTSs we will assume that they have the same set of ports and the same set of actions $\mathcal{A}ct_p$ for all $p \in \mathcal{P}orts$. Moreover, as usual, we assume that implementations are input-enabled.[1] 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 [21].

[1] 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.

A *trace* is a sequence of observable actions that can be performed, possibly interspersed with τ actions, from the initial state of a process. Since traces are composed of observable actions, they cannot contain occurrences of τ. In contrast, quiescence can appear in traces.

Definition 2 Let $s = (Q, I, O, T, q_{in})$ be an IOTS. We use the following notation.

1. If $(q, a, q') \in T_\delta$, for $a \in \mathcal{A}ct \cup \{\tau\}$, then we write $q \xrightarrow{a} q'$.
2. We write $q \overset{\epsilon}{\Longrightarrow} q'$ if there exist $q_0, \ldots, q_k \in Q$, for $k \geq 0$, such that $q = q_0$, $q' = q_k$, and we have the following transitions $q_0 \xrightarrow{\tau} q_1, \ldots, q_{k-1} \xrightarrow{\tau} q_k$.
3. We write $q \overset{a}{\Longrightarrow} q'$, for $a \in \mathcal{A}ct$, if there exist $q_0, q_0' \in Q$ such that we have the following: $q \overset{\epsilon}{\Longrightarrow} q_0, q_0 \xrightarrow{a} q_0', q_0' \overset{\epsilon}{\Longrightarrow} q'$.
4. Let $\sigma = a_1 \ldots a_k \in \mathcal{A}ct^*$ be a finite trace. We write $q \overset{\sigma}{\Longrightarrow} q'$ if there exist $q_0, \ldots, q_k \in Q$ such that $q = q_0$, $q' = q_k$ and for all $1 \leq i \leq k$ we have that $q_{i-1} \overset{a_i}{\Longrightarrow} q_i$.
5. We write $q \overset{\sigma}{\Longrightarrow}$ if there exists $q' \in Q$ such that $q \overset{\sigma}{\Longrightarrow} q'$.

6. We write $s \overset{\sigma}{\Longrightarrow}$ if $q_{in} \overset{\sigma}{\Longrightarrow}$ and we say that σ is a *trace* of s. We let $\mathcal{T}r(s)$ denote the *set of traces* of s.
7. $\sigma \in \mathcal{T}r(s)$ is a quiescent trace of s if $\sigma\delta \in \mathcal{T}r(s)$.

□

Note that for every state q we have $q \overset{\epsilon}{\Longrightarrow} q$ holds. Therefore, $\epsilon \in \mathcal{T}r(s)$ for every process s.

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 can then appear to be able to 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 use non-quiescent traces when comparing these processes then we conclude that they are not equivalent. However, in a distributed environment we have that $!o_1!o_2$ and $!o_2!o_1$ are equivalent since in each case the tester at port 1 will observe $!o_1$ and the tester at port 2 will observe output $!o_2$. Thus, we cannot distinguish between these two processes if we do not have additional information (e.g. a timestamp indicating which action was performed first). 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[2].

We introduce a system that will be used, as a running example, along the paper to illustrate the main concepts and the differences between implementation relations.

Example 1 The specification depicted in Figure 4 represents a simplified version of the user behaviour in a collaborative editor that allows multiple users to view and edit a shared document simultaneously[3]. When a user requests to open a document for editing, the server returns the corresponding file. Then, the user will have access to change, add, and delete content. Each time a user saves the changes made in the document, the rest of the users, working on the same file, receive a message and their versions are updated with the modifications. We will use this system as a running example to illustrate some of the concepts that we will introduce in the paper.

The system presents as many different ports as users accessing the document. All of them are connected to the central server where the last valid version of the document is stored. We denote by `EDShared` the specification of the collaborative editor. For the sake of clarity, we will consider only two users. □

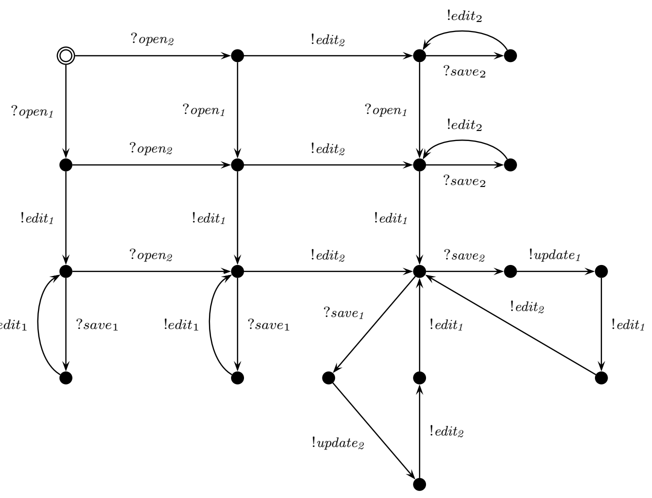

Fig. 4 Running example: collaborative Editor.

Previous implementation relations, devised for distributed testing, are based on an equivalence relation $\sim$ on traces. Essentially, the relation $\sim$ reflects the fact that 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 3 Let $p \in \mathcal{P}orts$ 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,

[2] It is possible to remove this restriction by considering infinite traces rather than quiescent traces [21] but this complicates the exposition.

[3] In the graphical representation we have omitted *irrelevant* input transitions. For each state q and input $?i \in \{?open_1, ?open_2, ?save_1, ?save_2\}$, if there does not exist an outgoing transition from q labelled by $?i$ then we assume the existence of the transition $(q, ?i, q)$.

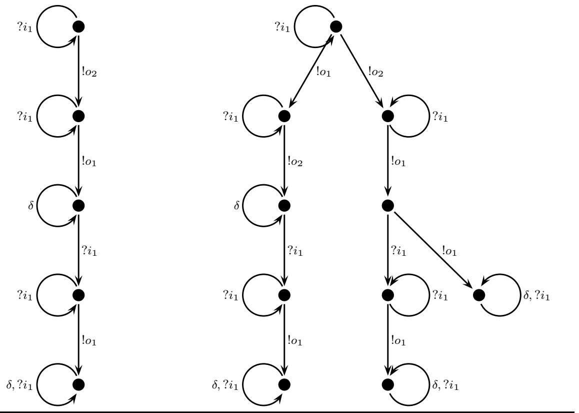

Fig. 5 r (left) and s (right); we cannot distinguish r from s without quiescence.

that is, for all $p \in \mathcal{P}orts$ we have that $\pi_p(\sigma) = \pi_p(\sigma')$. □

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 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 reasonable 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 is not observed at a particular time. In our previous work [22] we therefore did not include quiescence in timed traces. However, consider the following example shown in Figure 5 in which there are two ports and one input.

If we do not include the observation of quiescence in a timed trace then the addition of timestamps cannot help us to distinguish r from s since $!o_2!o_1?i_1!o_1$ is a trace of s. However, $!o_2!o_1\delta?i_1!o_1$ is not a trace of s and so timed traces that include quiescence can be used to distinguish r from s. Since the inclusion of quiescence in timed traces has the potential to lead to stronger implementation relations, we allow this. While the observation of quiescence does not happen at a particular time, in practice quiescence is observed using timeouts: the SUT failing to produce output before the timeout is seen as indicating that the SUT is quiescent. Naturally, the actual time used in the timeouts depends on properties of the SUT but we assume that it is chosen to be long enough for the local testers to know that the system is quiescent. While the local testers might give different local times to an occurrence of quiescence, we assume that a single time can be used in a timed trace. An alternative would be to include δ without a time in timed traces; this would not change the results but would make some of our definitions more complicated since we would have to include additional cases. Note that, it is straightforward to adapt the definitions and results in this paper to the situation in which we do not include quiescence in timed traces.

Definition 4 We consider that the time domain includes all non-negative real numbers, that is, $\text{Time} = \mathbb{R}_+$. Given $(a,t) \in \mathcal{A}ct \times \text{Time}$ we have that $\text{act}(a,t) = a$ and $\text{time}(a,t) = t$. Let $\sigma \in (\mathcal{A}ct \times \text{Time})^*$ be a sequence of (observable action, time) pairs. Then, we let $\text{untime}(\sigma)$ denote the trace produced from σ by removing the timestamps associated with actions. Formally,

$$\text{untime}(\sigma) = \begin{cases} \epsilon & \text{if } \sigma = \epsilon \\ a\ \text{untime}(\sigma') & \text{if } \sigma = (a,t)\sigma' \end{cases}$$

Let $s \in \mathtt{IOTS}(I, O, \mathcal{P}orts)$. A *timed trace* of s is a sequence $\sigma \in (\mathcal{A}ct \times \text{Time})^*$ such that there exists $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \sim \text{untime}(\sigma)$, and for all $p \in \mathcal{P}orts$ and j_1, j_2 such that $\text{act}(\sigma_{j_1}), \text{act}(\sigma_{j_2}) \in I_p \cup O_p$ and $j_1 < j_2$ we have $\text{time}(\sigma_{j_1}) < \text{time}(\sigma_{j_2})$. A timed trace σ is a *quiescent timed trace* of a process s if $\sigma(\delta, t)$ is a timed trace of s for some time t.

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 3 for untimed traces and we therefore omit it). □

We use σ both to denote timed and untimed traces: when we use σ 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* that can be observed. The idea is that 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)*. We first consider untimed traces, before generalising the definitions to timed traces. Since we wish to use a set of events we need a 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 $a_1 \dots a_{i-1}$. In other words, 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 that we observe the following (quiescent) untimed trace

$$\begin{aligned}\sigma =\, & ?open_1?open_2!edit_2!edit_1?save_2!update_1!edit_1!edit_2 \\ & ?save_2!update_1!edit_1!edit_2?save_1!update_2!edit_2!edit_1 \\ & ?save_2!update_1!edit_1!edit_2\end{aligned}$$

For example, the second occurrence of $?save_2$ in σ is represented by the event $e_\sigma(9) = (?save_2, 2)$ and the event set associated with σ is

$$e(\sigma) = \left\{\begin{array}{l} (?open_1, 1), (?open_2, 1), (!edit_2, 1), \\ (!edit_1, 1), (?save_2, 1), (!update_1, 1), \\ (!edit_1, 2), (!edit_2, 2), (?save_2, 2), \\ (!update_1, 2), (!edit_1, 3), (!edit_2, 3), \\ (?save_1, 1), (!update_2, 1), (!edit_2, 4), \\ (!edit_1, 4), (?save_2, 3), (!update_1, 3), \\ (!edit_1, 5), (!edit_2, 5) \end{array}\right\}$$

□

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 $(E, <)$, where $E = \{e_1, \dots, e_n\}$, 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.

Example 3 Consider the following untimed trace: $\sigma = ?open_1?open_2!edit_2!edit_1$. The set of linearisations of the partially ordered set $(e(\sigma), <_\sigma)$ is

$$L(e(\sigma), <_\sigma) = \left\{\begin{array}{l} ?open_1!edit_1?open_2!edit_2, \\ ?open_1?open_2!edit_1!edit_2, \\ ?open_1?open_2!edit_2!edit_1, \\ ?open_2?open_1!edit_1!edit_2, \\ ?open_2?open_1!edit_2!edit_1, \\ ?open_2!edit_2?open_1!edit_1 \end{array}\right\}$$

For the sake of clarity, we have omitted the occurrences of the events because all of the events are labelled with 1. Note that all the traces included in $L(e(\sigma), <_\sigma)$ preserve the order of the events in each port, as it is imposed by the partial order $<_\sigma$. For example, a permutation such as $?open_1!edit_2?open_2!edit_1$ is not allowed because it modifies the original order of the events at port 2. □

We have an interesting property, that will allow us to simplify several definitions and results, indicating that we can quantify over all traces equivalent to σ under $\sim$ by considering the set $L(e(\sigma), <_\sigma)$.

Proposition 1 Given $\sigma, \sigma' \in \mathcal{A}ct^*$, we have that $\sigma \sim \sigma'$ if and only if $L(e(\sigma), <_\sigma) = L(e(\sigma'), <_{\sigma'})$.

Proof First assume that $\sigma \sim \sigma'$. It is clear that $e(\sigma) = e(\sigma')$ and so it is sufficient to prove that $<_\sigma = <_{\sigma'}$. However, this must be the case since σ and σ' have the same projection at each port and so define the same total order at each port.

Now assume that $L(e(\sigma), <_\sigma) = L(e(\sigma'), <_{\sigma'})$. It is sufficient to prove that for each port p we have that $\pi_p(\sigma) = \pi_p(\sigma')$. Since $L(e(\sigma), <_\sigma) = L(e(\sigma'), <_{\sigma'})$ we must have that the same sets of events occur at p in σ and σ'. In addition, we must have that $<_\sigma = <_{\sigma'}$ and so the total orders defined by restricting $<_\sigma$ and $<_{\sigma'}$ to the events observed at p must be identical. Thus, $\pi_p(\sigma) = \pi_p(\sigma')$ as required.

□

3 Implementation relations in the (untimed) distributed test architecture

In this section we present the different implementation relations that will be used in this paper as the basis to construct our implementation relations for distributed timed systems. While some of these relations have already appeared in the literature (**ioco** [40], **pdioco** and **dioco** [21]) the implementation relations $\mathbf{dioco}^F$ and $\mathbf{dioco}^N$, where we consider that communication can be asynchronous, are original and introduced in this paper. While there is work that has investigated MBT through asynchronous channels [3,7,4,2], this has not explored distributed testing.

First, we present the standard implementation relation for testing from an IOTS where information about different ports is not taken into account: **ioco**. We initially give some auxiliary notation.

Definition 7 Let $s = (Q, I, O, T, q_{in})$ be an IOTS. Let $q \in Q$ be a state and $\sigma \in \mathcal{A}ct^*$ be a trace. We introduce the following concepts.

1. $q \textbf{ after } \sigma = \begin{cases} \{r \in Q | q \overset{\sigma}{\Longrightarrow} r\} & \text{if } q \overset{\sigma}{\Longrightarrow} \\ \emptyset & \text{otherwise} \end{cases}$
2. $\mathbf{out}(q) = \{o \in O \cup \{\delta\} | q \overset{o}{\Longrightarrow}\}$

The function **out** can be extended to deal with sets in the expected way, that is, given $Q' \subseteq Q$ we define $\mathbf{out}(Q') = \cup_{q \in Q'} \mathbf{out}(q)$. □

Intuitively, 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.

Definition 8 Given $r, s \in \mathtt{IOTS}(I, O, \mathcal{P}orts)$, we write r **ioco** s if for every sequence $\sigma \in \mathcal{T}r(s)$ we have that $\mathbf{out}(r \textbf{ after } \sigma) \subseteq \mathbf{out}(s \textbf{ after } \sigma)$. □

The idea behind the definition of **ioco** is that the SUT should not exhibit a behaviour, in terms of the observed outputs, that was not allowed in the specification. This implementation relation was adapted to the distributed test architecture [21] under different assumptions but always keeping in mind the concepts behind the **ioco** relation.

We start by defining an implementation relation for the situation in which the agents that will interact with the ports of the SUT are entirely independent in that no external agent will receive information, regarding observations made of the SUT, from more than one such agent. In such a situation, it is sufficient that each agent observes a local trace that is consistent with the specification and this leads to the **pdioco** implementation relation.

Definition 9 Given $r, s \in \mathtt{IOTS}(I, O, \mathcal{P}orts)$, we write r **pdioco** s if for every (finite) trace $\sigma \in \mathcal{T}r(r)$ and for every port $p \in \mathcal{P}orts$ there exists some trace $\sigma' \in \mathcal{T}r(s)$ such that $\pi_p(\sigma) = \pi_p(\sigma')$. □

Under **pdioco** the tester at port p compares the local trace it observes with the projections of the traces of the specification. It is therefore possible for the trace σ' in Definition 9 to vary: the testers might use different traces of the specification s. Sometimes, however, there is the potential for observations made at the different ports to be combined later. To see the effect of this, consider the situation in which the specification allows the global traces $\sigma_1 = ?i_1!o_1!o_2$ and $\sigma_2 = ?i_1!o_1'!o_2'$ and the SUT produces the trace $\sigma = ?i_1!o_1!o_2'$. Under **pdioco** this is acceptable: the local trace observed at port 1 is equal to $\pi_1(\sigma_1)$ and the local trace observed at port 2 is equal to $\pi_2(\sigma_2)$. However, if we bring together these local traces then we can deduce that the global trace produced by the SUT was not one allowed by the specification. The implementation relation **dioco** [19–21] allows the set of local traces observed to be compared with the global traces of the specification.

Definition 10 Given $r, s \in \mathtt{IOTS}(I, O, \mathcal{P}orts)$, 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 pointed out in the previous section, the implementation relation **dioco** only considers quiescent traces, that is, traces that reach a quiescent state. The motivation for this restriction is that it is necessary to compare sets of local traces with global traces, while ensuring that the local traces are all projections of the same (unknown) global trace of the implementation. If this trace ends in quiescence then it is possible to check the appropriateness of the projections and, indeed, the testers might stop testing at this point. 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

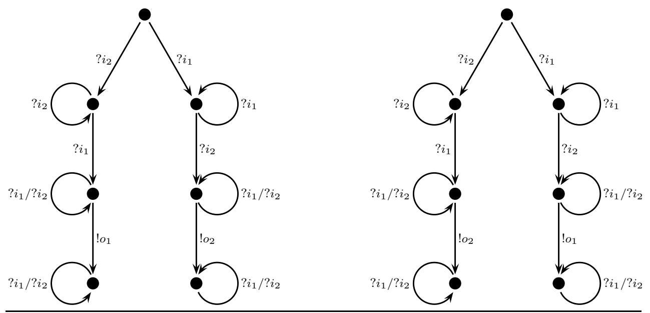

Fig. 6 r (left) and s (right) are not related under **ioco** but are related under **dioco**.

does not hold. Figure 6 shows two processes r (left) and s (right) such that r **dioco** s holds but r **ioco** s does not hold.

Even though the implementation relation **dioco** captures a general notion of conformance, it is not adequate if we consider systems where communication is asynchronous. Therefore, it is necessary to adapt **dioco** to deal with this feature. We consider two variants: FIFO and non-FIFO asynchronous communication.

The new relation $\mathbf{dioco}^F$, standing for FIFO **dioco**, differs from **dioco** in one important way: the observation of output might have been delayed.[4] As with **dioco** we will require that observations are made in quiescent states; testing might finish in such a state and the testers then know that the local traces they have observed all relate to the same global trace of the SUT. There is an asymmetry in the effect of having asynchronous communications: the observation of an output might be delayed but it cannot be early. Thus, the relation that we will use to compare traces, instead of $\sim$, is not symmetric. We first define a relation $\sqsubseteq_p$ on a sequence of observations made by tester p; we will use this to define a relation on global traces. Intuitively, given a port p and two local traces at p, σ and σ', we have that $\sigma \sqsubseteq_p \sigma'$ holds if and only if σ can be formed from σ' by delaying output.

Definition 11 Given $p \in \mathcal{P}orts$ we define $\sqsubseteq_p$ to be the smallest reflexive and transitive binary relation on $\mathcal{A}ct_p^*$ that includes all pairs of the form $(\sigma?i_p!o_p\sigma', \sigma!o_p?i_p\sigma')$ for $?i_p \in I_p$, $!o_p \in O_p$ and $\sigma, \sigma' \in \mathcal{A}ct_p^*$.

We define the relation $\sqsubseteq$ on $\mathcal{A}ct^*$ by $\sigma \sqsubseteq \sigma'$ if and only if for all $p \in \mathcal{P}orts$ we have that $\pi_p(\sigma) \sqsubseteq_p \pi_p(\sigma')$. □

It is now straightforward to define a version of **dioco** for asynchronous FIFO communications.

[4] We assume that an output cannot be delayed *past* quiescence but it is straightforward to change the definition in order to remove this assumption.

Definition 12 Let r, s be `IOTSs`. We write r $\mathbf{dioco}^F$ 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 $\sigma \sqsubseteq \sigma'$. □

Example 4 Consider our running example and an SUT in an asynchronous scenario with FIFO communications. If the SUT produces the sequence

$$\begin{aligned}\sigma =\,& ?open_1?open_2!edit_1!edit_2?save_1?save_2\\ & !update_2!edit_2!edit_1!update_1!edit_1!edit_2\end{aligned}$$

then we cannot claim that the system is incorrect according to $\mathbf{dioco}^F$ with respect to our specification since the trace

$$\begin{aligned}\sigma' =\,& ?open_1?open_2!edit_1!edit_2?save_1!update_2\\ & !edit_2!edit_1?save_2!update_1!edit_1!edit_2\end{aligned}$$

belongs to $\mathcal{T}r(\texttt{EDShared})$ and $\sigma \sqsubseteq \sigma'$. The trace σ is produced from σ' by delaying the observation of the two consecutive outputs $!update_2$ and $!edit_2$. □

We also have to define a variant of **dioco** for the case where communications are asynchronous and non-FIFO. The framework is slightly different if communications need not be FIFO since the tester at port p might observe output $!o_p$ before output $!o'_p$ in cases where $!o'_p$ was produced by the SUT before $!o_p$; $!o'_p$ might overtake $!o_p$. Similarly, inputs can overtake one another. We first define a relation $\sqsubseteq_p^N$ on a sequence of observations made by tester p and then use this to define a relation on global traces. Intuitively, $\sigma \sqsubseteq_p^N \sigma'$ holds if and only if the local trace σ might be formed from the local trace σ' by delaying output or by input or output overtaking one another.[5]

Definition 13 Given $p \in \mathcal{P}orts$, we define $\sqsubseteq_p^N$ to be the smallest reflexive and transitive binary relation on $\mathcal{A}ct_p^*$ that includes all pairs of the form

1. $(\sigma?i_p!o_p\sigma', \sigma!o_p?i_p\sigma')$ for $?i_p \in I_p$, $!o_p \in O_p$ and $\sigma, \sigma' \in \mathcal{A}ct_p^*$.
2. $(\sigma?i_p?i'_p\sigma', \sigma?i'_p?i_p\sigma')$ for $?i_p, ?i'_p \in I_p$ and $\sigma, \sigma' \in \mathcal{A}ct_p^*$.
3. $(\sigma!o_p!o'_p\sigma', \sigma!o'_p!o_p\sigma')$ for $!o_p, !o'_p \in O_p$ and $\sigma, \sigma' \in \mathcal{A}ct_p^*$.

We then define $\sqsubseteq^N$ on $\mathcal{A}ct^*$ by $\sigma \sqsubseteq^N \sigma'$ if and only if for all $p \in \mathcal{P}orts$ we have that $\pi_p(\sigma) \sqsubseteq_p^N \pi_p(\sigma')$. □

[5] Again we assume that an output cannot be delayed *past* quiescence but it is straightforward to change the definition in order to remove this assumption.

The second and third rules are a result of communications being non-FIFO. For example, if the SUT produces output sequence $!o_1!o'_1$ then the output $!o'_1$ might be observed before $!o_1$.

Next we present the adaption of **dioco** for asynchronous non-FIFO communications.

Definition 14 Let r, s be `IOTSs`. We write r $\textbf{dioco}^N$ 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 $\sigma \sqsubseteq^N \sigma'$. □

Example 5 Consider our running example and an SUT in an asynchronous scenario with non-FIFO communications. If the SUT produces the sequence

$$\sigma = ?open_1?open_2?save_1!edit_1!edit_2?save_2$$
$$!edit_2!update_2!edit_1!update_1!edit_1!edit_2$$

we have then we cannot infer that the system is incorrect with respect to $\textbf{dioco}^N$ since the trace

$$\sigma' = ?open_1?open_2!edit_1!edit_2?save_1!update_2$$
$$!edit_2!edit_1?save_2!update_1!edit_1!edit_2$$

belongs to $\mathcal{T}r(\texttt{EDShared})$ and $\sigma \sqsubseteq^N \sigma'$. The trace σ might show a delay in the first observation of $!edit_1$ and the observation of the outputs $!update_2$ and $!edit_2$. □

4 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, note that the addition of time does not modify our implementation relation **pdioco**. 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. 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, potential causalities between events at different ports are of no interest and the addition of time does not change the implementation relation. However, this is not the case for the implementation relation **dioco** and in this section we show how timestamps give raise to alternative implementation relations.

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 $r, s \in \texttt{IOTS}(I, O, \mathcal{P}orts)$, 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 $\sigma' \in L(e(\sigma), <_\sigma)$.

Proof First observe that $\sigma' \in L(e(\sigma), <_\sigma)$ holds if and only if we have that $L(e(\sigma'), <_{\sigma'}) = L(e(\sigma), <_\sigma)$. The result thus follows from Proposition 1. □

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 $!o_1$ at port 1 followed by $!o_2$ at port 2. If the SUT produces $!o_2$ followed by $!o_1$ 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 $(!o_1, 100)$ and $(!o_2, 98)$. If we have a global clock or local clocks that work perfectly, then we can claim that the SUT is not correct since $!o_2$ was performed before $!o_1$. 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 $!o_2$ was produced after $!o_1$ but that the clock at port 1 is running faster than the one placed at port 2. Therefore, we need a variety of implementation relations to cope with the alternatives.

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

Definition 15 Let $\sigma = (a_1, t_1) \ldots (a_n, t_n) \in (\mathcal{A}ct \times \text{Time})^*$ be a timed trace. We define $\ll_\sigma$ as 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 $r, s \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ and $\mathcal{T} \in \mathcal{P}((\mathcal{A}ct \times \text{Time})^*)$ be a set of quiescent timed traces of r where $\mathcal{P}$ denotes the powerset. We write r $\textbf{tdioco}(\mathcal{T})$ 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 $\sigma' \in L(e(\sigma), \ll_\sigma)$. □

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.

Example 6 Consider our running example and an SUT producing the sequence

$$?open_1?open_2!edit_1!edit_2?save_1!update_2!edit_1!edit_2$$

Even though this is not a trace of the specification, due to the second occurrence of $!edit_1$ and $!edit_2$, we have that the difference in the order between actions happens at different ports. Therefore, the system is correct with respect to **dioco**. However, if we assume that the clocks used to timestamp actions work perfectly and we observe the following sequence

$$(?open_1, 1), (?open_2, 2), (!edit_1, 3), (!edit_2, 3.8),$$
$$(?save_1, 4), (!update_2, 4.5), (!edit_1, 4.6), (!edit_2, 4.8)$$

we can claim that the system is not correct since we know that $!edit_1$ was performed before $!edit_2$. □

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

Definition 16 Let $\sigma = (a_1, t_1) \ldots (a_n, t_n) \in (\mathcal{A}ct \times \text{Time})^*$ be a timed trace and $\alpha \in \mathbb{R}_+$. We define $\ll_\sigma^\alpha$ as the partial order on $e(\sigma)$ such that for all $1 \le i, j, \le n$ with $i \neq j$ we have that $e_\sigma(i) \ll_\sigma^\alpha e_\sigma(j)$ if and only if one or more of the following holds.

– There exists $p \in \mathcal{P}orts$ such that $a_i, a_j \in \mathcal{A}ct_p$ 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 17 Let $r, s \in \texttt{IOTS}(I, O, \mathcal{P}orts)$, $\mathcal{T} \in \mathcal{P}((\mathcal{A}ct \times \text{Time})^*)$ 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 7 Let r be an SUT such that r **dioco** `EDShared`. 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

$$\begin{aligned}\sigma =& (?open_1, 0.2), (?open_2, 0.25), (!edit_1, 0.26),\\ & (!edit_2, 0.27), (?save_1, 1.3), (!update_2, 1.45),\\ & (!edit_2, 1.5), (?save_2, 1.7), (!edit_1, 1.75),\\ & (!update_1, 1.8), (!edit_2, 1.9), (!edit_1, 2.1)\end{aligned}$$

We have that r $\mathbf{tdioco}(\mathcal{T})$ `EDShared` does not hold since, for example, the second occurrence of $!edit_1$ should be observed before the occurrence of $?save_2$. However, if local clocks differ by at most $\alpha = 0.2$ units of time, then r $\mathbf{tdioco}_\alpha(\mathcal{T})$ `EDShared`. In that case, it is possible that event $(!edit_1, 1.75)$ was produced before the input $(?save_2, 1.7)$ was sent, and the outputs $(!edit_2, 1.9)$ and $(!edit_1, 2.1)$ were emitted in reverse order, due to the possible difference between the local clocks. □

We can compare implementation relations using the following relations. Note that in the following definition we refer to the concept of an implementation relation having parameters; by this we mean, for example, that α and $\mathcal{T}$ are parameters of $\mathbf{tdioco}_\alpha(\mathcal{T})$.

Definition 18 Let imp_1 and imp_2 be two implementation relations. We write $imp_1 \sqsubseteq imp_2$ if and only if for all `IOTSs` r, s and parameter values for the implementation relations we have that r imp_2 s implies r imp_1 s. Further, we write $imp_1 \equiv imp_2$ if $imp_1 \sqsubseteq imp_2$ and $imp_2 \sqsubseteq imp_1$ and we write $imp_1 \sqsubset imp_2$ if $imp_1 \sqsubseteq imp_2$ but we do not have that $imp_1 \equiv imp_2$. □

We can now compare our new implementation relation with **dioco**.

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 $\mathbf{dioco} \sqsubset \mathbf{tdioco}_\alpha(\mathcal{T})$. Similarly, for all $\alpha \in \mathbb{R}_+$ we have that $\mathbf{dioco} \equiv \mathbf{tdioco}_\alpha(\emptyset)$.

Proof First note that we have that $\mathbf{dioco} \sqsubseteq \mathbf{tdioco}_\alpha(\mathcal{T})$ follows immediately from the definitions. In order to show that $\mathbf{tdioco}_\alpha(\mathcal{T}) \sqsubseteq \mathbf{dioco}$ does not hold, it is sufficient to consider a process s that has the quiescent trace $!o_1!o_2\delta$, a process r that has the quiescent 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 $!o_1$. The proof of the second result is immediate.

□

If we abuse the notation slightly, to allow α to take on the value of 0, then we obtain the following result. Note that where we refer to a set being minimal we mean that it is minimal under set inclusion.

Proposition 4 Let r be an `IOTS`. Let $\mathcal{T}$ be a minimal set of timed traces such that for each quiescent trace $\sigma = a_1a_2 \dots a_n \in \mathcal{T}r(r)$ there exists $t_1, t_2, \dots, t_n \in$ Time such that for all $1 \leq r_1 < r_2 \leq n$ we have that $t_{r_1} < t_{r_2}$ and also that the timed trace $\sigma' = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n)$ belongs to $\mathcal{T}$. For all `IOTS` s we have that r **ioco** s if and only if r $\mathbf{tdioco}_0(\mathcal{T})$ s. In addition, $\mathbf{tdioco}_0(\mathcal{T}) \equiv \mathbf{tdioco}(\mathcal{T})$.

Proof First, we prove the left to right implication. Given `IOTSs` r, s we assume that r **ioco** s and we prove the result by contradiction. Assume that r $\mathbf{tdioco}_0(\mathcal{T})$ s does not hold. Since r **ioco** s we have that r **dioco** s [21], we must have that there exists $\sigma \in \mathcal{T}$ such that there does not exist a quiescent trace $\sigma' \in \mathcal{T}r(s)$ that belongs to $L(e(\sigma), \ll^0_\sigma)$. Due to the time values considered in the timed traces belonging to $\mathcal{T}$, we have $L(e(\sigma), \ll^0_\sigma) = \{\sigma\}$. Therefore, $\sigma \notin \mathcal{T}r(s)$. Let $\sigma' a$ be the shortest prefix of σ such that $\sigma' a \notin \mathcal{T}r(s)$. Since r and s are input-enabled we must have that a is an output and by the minimality of $\sigma' a$ we have that $\sigma' \in \mathcal{T}r(s)$. Thus, we have $a \in \mathbf{out}(r\ \mathbf{after}\ \sigma')$ but $a \notin \mathbf{out}(s\ \mathbf{after}\ \sigma')$. This fact contradicts r **ioco** s.

Now let us assume that r $\mathbf{tdioco}_0(\mathcal{T})$ s and so we are required to prove that r **ioco** s. By contradiction, assume that r **ioco** s does not hold. Let $\sigma \in \mathcal{T}r(s)$ be a trace such that $\mathbf{out}(r\ \mathbf{after}\ \sigma) \not\subseteq \mathbf{out}(s\ \mathbf{after}\ \sigma)$. Therefore, there exists $a \in O \cup \{\delta\}$ such that $\sigma a \in \mathcal{T}r(r)$ and $\sigma a \notin \mathcal{T}r(s)$. Since r is not output divergent, σa is a prefix of some quiescent trace in $\mathcal{T}r(r)$. By the hypotheses, for this quiescent trace, there exists a timed trace σ' that belongs to $\mathcal{T}$. Moreover, since r $\mathbf{tdioco}_0(\mathcal{T})$ s and $L(e(\sigma'), \ll^0_{\sigma'}) = \{\sigma'\}$, we have $\sigma' \in \mathcal{T}r(s)$. Therefore, $\sigma a \in \mathcal{T}r(s)$, and this provides a contradiction as required.

Finally, $\mathbf{tdioco}_0(\mathcal{T}) \equiv \mathbf{tdioco}(\mathcal{T})$ follows immediately from the definitions.

□

The following gives a more general condition under which we can compare two implementation relations defined using different values for α and $\mathcal{T}$.

Proposition 5 Let r be an `IOTS`. 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 sequence $\sigma_1 \in \mathcal{T}_1$ there exists $\sigma_2 \in \mathcal{T}_2$ such that $e(\sigma_2) = e(\sigma_1)$ and $\ll^{\alpha_1}_{\sigma_1} \subseteq \ll^{\alpha_2}_{\sigma_2}$.

Proof We will assume that the hypotheses hold and that for `IOTSs` r, s we have that r $\mathbf{tdioco}_{\alpha_2}(\mathcal{T}_2)$ s. Then we want to prove that r $\mathbf{tdioco}_{\alpha_1}(\mathcal{T}_1)$ s. By definition, we must have that r **dioco** s. Thus, it is sufficient to prove that for every timed trace $\sigma_1 \in \mathcal{T}_1$ there exists a trace $\sigma'_1 \in \mathcal{T}r(s)$ such that $\sigma'_1 \in L(e(\sigma_1), \ll^{\alpha_1}_{\sigma})$.

Since the hypotheses hold, there is a timed trace $\sigma_2 \in \mathcal{T}_2$ such that $e(\sigma_2) = e(\sigma_1)$ and $\ll^{\alpha_1}_{\sigma_1} \subseteq \ll^{\alpha_2}_{\sigma_2}$. Further, since r $\mathbf{tdioco}_{\alpha_2}(\mathcal{T}_2)$ s there is a trace $\sigma'_2 \in \mathcal{T}r(s)$ such that $\sigma'_2 \in L(e(\sigma_2), \ll^{\alpha_2}_{\sigma_2})$. But since $\ll^{\alpha_1}_{\sigma_1} \subseteq \ll^{\alpha_2}_{\sigma_2}$ we have that $L(e(\sigma_2), \ll^{\alpha_2}_{\sigma_2}) \subseteq L(e(\sigma_1), \ll^{\alpha_1}_{\sigma_1})$. Thus we can use $\sigma'_1 = \sigma'_2$ and so the result holds.

□

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 19 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 $\sigma' = a'_1 \dots a'_n \in \mathcal{T}r(r)$ with $e(\text{untime}(\sigma)) = e(\sigma')$ such that for all $1 \leq i < j \leq n$ we have that $\eta_\sigma(e_{\sigma'}(i)) - \eta_\sigma(e_{\sigma'}(j)) \leq \alpha$. □

This condition requires that if event $e_\sigma(i)$ is before event $e_\sigma(j)$ 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 Let $r, s \in \mathtt{IOTS}(I, O, \mathcal{P}orts)$, $\alpha \in \mathbb{R}_+$ and $\mathcal{T}$ be a set of quiescent timed traces that are valid for r given α. We have that $\mathbf{tdioco}_\alpha(\mathcal{T}) \sqsubseteq \mathbf{ioco}$.

Proof We will assume that the hypotheses hold and that for `IOTSs` r, s we have that r **ioco** s. Then we want to prove that r $\mathbf{tdioco}_\alpha(\mathcal{T})$ s. Since r **ioco** s we have that r **dioco** s [21]. Thus, it is sufficient to prove that for every quiescent timed trace $\sigma \in \mathcal{T}$ there exists a trace $\sigma_1 \in \mathcal{T}r(s)$ such that $\sigma_1 \in L(e(\sigma), \ll^\alpha_\sigma)$.

By definition, there exists a trace $\sigma' = a'_1 \dots a'_n \in \mathcal{T}r(r)$ with $e(\text{untime}(\sigma)) = e(\sigma')$ such that for all $1 \leq i < j \leq n$ we have that $\eta_\sigma(e_{\sigma'}(i)) - \eta_\sigma(e_{\sigma'}(j)) \leq \alpha$. Therefore, $\sigma' \in L(e(\sigma), \ll^\alpha_\sigma)$.

Now, assume that $\sigma' \notin \mathcal{T}r(s)$. Let $\sigma'' a$ be the shortest prefix of σ' such that $\sigma'' a \notin \mathcal{T}r(s)$. Since r and s are input-enabled we must have that a is either an output or δ and by the minimality of $\sigma'' a$ we have that $\sigma'' \in \mathcal{T}r(s)$. Thus, we have $a \in \mathbf{out}(r\ \mathbf{after}\ \sigma'')$ but $a \notin \mathbf{out}(s\ \mathbf{after}\ \sigma'')$. This fact contradicts r **ioco** s.

□

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**. We now 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. The following result states 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. The proof of the proposition is similar to the one given in Proposition 4 since the only difference lies in the fact that for each trace $\sigma = (a_1, 1)(a_2, 2) \dots (a_n, n) \in \mathcal{T}$, the sets $L(e(\sigma), \ll_\sigma)$, used in the first result, and $L(e(\sigma), \ll_\sigma^\alpha)$, used in the second result, are equal to $\{a_1 a_2 \dots a_n\}$.

Proposition 7 Let $r, s \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ and $\alpha \in \mathbb{R}_+$ be a positive real number.

Let $\mathcal{T}$ be the minimal set of quiescent timed traces such that for each quiescent trace $\sigma = a_1 a_2 \dots a_n$ in $\mathcal{T}r(r)$ we have that $\sigma' = (a_1, 1)(a_2, 2) \dots (a_n, n)$ belongs to $\mathcal{T}$. We have r **tdioco**$(\mathcal{T})$ s if and only if we have that r **ioco** s.

Let $\mathcal{T}$ be the minimal set of timed traces such that for each quiescent trace $\sigma = a_1 a_2 \dots a_n$ belonging to $\mathcal{T}r(r)$ we have that the timed trace $\sigma' = (a_1, \alpha{+}1)(a_2, 2\cdot(\alpha + 1)) \dots (a_n, n \cdot (\alpha + 1))$ belongs to $\mathcal{T}$. We have r **tdioco**$_\alpha(\mathcal{T})$ s if and only if we have that r **ioco** s. □

The results shown in the previous proposition can be generalised as follows.

Proposition 8 Let $r, s \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ and $\alpha \in \mathbb{R}_+$ be a positive real number.

Let $\mathcal{T}$ be a minimal set of quiescent timed traces such that for each quiescent trace $\sigma = a_1 a_2 \dots a_n \in \mathcal{T}r(r)$ there exists $t_1, t_2, \dots, t_n \in$ Time such that for all $1 \leq r_1 < r_2 \leq n$ we have that $t_{r_1} < t_{r_2}$ and that the timed trace $\sigma' = (a_1, t_1), (a_2, t_2), \dots, (a_n, t_n)$ belongs to $\mathcal{T}$. We have that r **tdioco**$(\mathcal{T})$ s if and only if we have that r **ioco** s.

Let $\mathcal{T}$ be a minimal set of timed traces such that for each quiescent trace $\sigma = a_1 a_2 \dots a_n \in \mathcal{T}r(r)$ there exists $t_1, t_2, \dots, t_n \in$ Time such that for all $1 \leq r_1 < r_2 \leq n$ we have that $t_{r_1} + \alpha < t_{r_2}$ and that the timed trace $\sigma' = (a_1, t_1), (a_2, t_2), \dots, (a_n, t_n)$ belongs to $\mathcal{T}$. We have that r **tdioco**$_\alpha(\mathcal{T})$ s if and only if we have that r **ioco** s. □

Finally, the following result states that *irrelevant* time values do not add distinguishing power to the collected set of traces, so that we obtain the **dioco** relation.

Proposition 9 Let $r, s \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ and $\alpha \in \mathbb{R}_+$ be a positive real number. Let $\mathcal{T}$ be the minimal set of timed traces such that for each quiescent trace $\sigma = a_1 a_2 \dots a_n \in \mathcal{T}r(r)$ we have that the timed trace $\sigma' = (a_1, \alpha)(a_2, \alpha \cdot \frac{3}{2}) \dots (a_n, \alpha \cdot \sum_{j=0}^{n-1} \frac{1}{2^j})$ belongs to $\mathcal{T}$. We have that r **tdioco**$_\alpha(\mathcal{T})$ s if and only if we have that r **dioco** s. □

The proof of this result is a particular case of the following result.

Proposition 10 Let $r, s \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ and $\alpha \in \mathbb{R}_+$ be a positive real number. Let $\mathcal{T}$ be a minimum set of timed traces such that for each quiescent trace $\sigma = a_1, a_2, \dots, a_n \in \mathcal{T}r(r)$ there exists $t_1, t_2, \dots, t_n \in$ Time such that for all $1 \leq r_1 < r_2 \leq n$ we have that $t_{r_1} + \alpha \geq t_{r_2}$ and that the timed trace $\sigma' = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n)$ belongs to $\mathcal{T}$. We have that r **tdioco**$_\alpha(\mathcal{T})$ s if and only if we have that r **dioco** s.

Proof The left to right implication is immediate. In order to prove the right to left implication we assume that r **dioco** s and we have to show that r **tdioco**$_\alpha(\mathcal{T})$ s. It is sufficient to prove that for every timed trace $\sigma = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n) \in \mathcal{T}$ there exists a trace $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \in L(e(\sigma), \ll_\sigma^\alpha)$. Since r **dioco** s there is a trace $\sigma'' \in \mathcal{T}r(s)$ such that $\sigma'' \in L(e(\sigma), <_\sigma)$. Due to the fact that for all $1 \leq r_1 < r_2 \leq n$ we have that $t_{r_1} + \alpha \geq t_{r_2}$, $L(e(\sigma), \ll_\sigma^\alpha) = L(e(\sigma), <_\sigma)$. Thus we can use $\sigma' = \sigma''$ and so the result holds.

□

5 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. Note that this is a very realistic assumption. For example, it is well known that processor clocks usually have known bounds on their rate of progress with respect to real-time [34], that is, processor clocks usually satisfy the *linear drift* assumption that we consider in this section. Even though clocks do not work at the same speeds we can assume that they are *similar* enough because if we have a strong evidence that this is not the case, then we replace them with new ones. 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 20 Let $\sigma = (a_1, t_1) \dots (a_n, t_n) \in (\mathcal{A}ct \times \mathrm{Time})^*$ be a timed trace and $\alpha, \beta \in \mathbb{R}_+$. We define $\ll_\sigma^{\alpha,\beta}$ as 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 or more of the following holds.

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

Let $r, s \in \mathtt{IOTS}(I, O, \mathcal{P}orts)$, $\mathcal{T} \in \mathcal{P}((\mathcal{A}ct \times \mathrm{Time})^*)$ 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 $\sigma' \in L(e(\sigma), \ll_\sigma^{\alpha,\beta})$. $\square$

Example 8 Consider the specification of the collaborative editor depicted in Figure 4. Let r be an SUT such that its conformance to the `EDShared` specification with respect to **dioco** has been established. Assume that while testing r we obtained the set $\mathcal{T}$ of timed sequences formed by the following quiescent timed traces:

$$\begin{aligned}
tr_1 = {} & (?open_1, 0.2), (?open_2, 0.25), (!edit_1, 0.25), \\
& (?save_1, 0.26), (!edit_1, 0.27), (!edit_2, 0.28) \\
tr_2 = {} & (?open_1, 1), (?open_2, 1.2), (!edit_1, 1.3), (?save_1, 1.4), \\
& (!edit_2, 1.45), (?save_2, 2.3), (!update_1, 2.4), \\
& (!edit_2, 2.42), (!edit_1, 2.47) \\
tr_3 = {} & (?open_1, 0.2), (?open_2, 0.25), (!edit_1, 0.25), \\
& (!edit_2, 0.27), (?save_2, 1.1), (!edit_2, 1.17), \\
& (!update_1, 1.24), (!edit_1, 1.27)
\end{aligned}$$

The sequences tr_2 and tr_3 show that the relation r $\mathbf{tdioco}(\mathcal{T})$ `EDShared` does not hold since, for example, the events $(!edit_2, 2.42), (!edit_1, 2.47)$ in tr_2 were produced in reverse order to the one specified in `EDShared`. In the same way, the event $(!edit_2, 1.17)$ of tr_3 is observed before $(!update_1, 1.24), (!edit_1, 1.27)$ which, according to the specification, must be produced before. In contrast, r $\mathbf{tdioco}_\alpha(\mathcal{T})$ `EDShared` holds if $\alpha \geq 0.1$; if $\alpha < 0.1$ the conformance does not hold. If we assume that the difference between clocks grows with time, then for any value assigned to β when $\alpha \geq 0.1$, we have that r $\mathbf{tdioco}_{\alpha,\beta}(\mathcal{T})$ `EDShared` holds. However if, for example, $\alpha = 0.05$ and $\beta < 0.04$, then the difference between the observed times for the actions $(!edit_2, 1.17)$ and $(!edit_1, 1.27)$ is greater than the established bound $\alpha + \beta \cdot \max(1.17, 1.27)$ $\square$

The next result, whose proof is easy, compares the implementation relations $\mathbf{tdioco}_{\alpha,\beta}(\mathcal{T})$ and $\mathbf{tdioco}_\alpha(\mathcal{T})$.

Proposition 11 Let $\mathcal{T}$ be a set of timed traces and $\alpha, \beta \in \mathbb{R}_+$. We have $\mathbf{tdioco}_{\alpha,\beta}(\mathcal{T}) \sqsubset \mathbf{tdioco}_\alpha(\mathcal{T})$. $\square$

The next relation is a generalisation of $\mathbf{tdioco}_{\alpha,\beta}(\mathcal{T})$ where we consider that 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 two clocks.

Definition 21 Let $\sigma = (a_1, t_1) \dots (a_n, t_n) \in (\mathcal{A}ct \times \mathrm{Time})^*$ be a timed trace and $h : \mathbb{R}_+ \longrightarrow \mathbb{R}_+$ be a monotonically increasing function. We define $\ll_\sigma^h$ as 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 or more of the following holds.

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

Let $r, s \in \mathtt{IOTS}(I, O, \mathcal{P}orts)$, $\mathcal{T} \in \mathcal{P}((\mathcal{A}ct \times \mathrm{Time})^*)$ 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 $\sigma' \in L(e(\sigma), \ll_\sigma^h)$. $\square$

We would like to conclude this section by pointing out that if we consider single-port systems then all the timed implementation relations previously introduced 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.

6 Asynchronous FIFO communications

In the previous sections we showed how timestamps could be used to add causalities between events observed at different ports. However, it was assumed that communications between the testers and the SUT is synchronous: an event is observed at the time it was produced. In some cases this is a reasonable assumption: even if the SUT is physically distributed it may be possible to have a set of distributed testers such that the tester at port p interacts with port p of the SUT in a synchronous manner. However, testers will often interact with the SUT through communications channels and so in this paper we also consider the asynchronous case. We will investigate two different types of asynchronous communications. In this section we consider FIFO communications while Section 7 considers non-FIFO communications. Note that both approaches are relevant: while the internet is non-FIFO some networks enforce FIFO communications. In both sections we assume again that quiescence can be observed by the local testers and that the testers can wait sufficiently long so that it is guaranteed that all inputs have been received and all outputs have been observed. Where this is not feasible, we can simply remove quiescence from the set of possible observations.

In order to reason about the causality between events that have been observed in asynchronous testing it is necessary to make assumptions regarding message latency. We assume that there is a known lower bound Δ_{LB} on message latency and also a known upper bound Δ_{UB}. We also assume that we have the same bound for each channel and in each direction; it is straightforward to adapt the definitions and results to the case where bounds differ.

We start by discussing the imprecision introduced by the asynchronous communications only and so analyse the case where we have a global clock; later we combine this with information regarding imprecision in the local clocks. Since communications are FIFO, we can order two inputs provided at port p and also two outputs observed at p. We have four cases to consider.

- Let us suppose that output $!o_1$ has been observed at port 1 and that $!o_2$ has been observed at port 2. Assume also that $!o_1$ was observed at (global) time t_1, $!o_2$ was observed at (global) time t_2 and $t_2 > t_1$. Given the bounds on message latency, $!o_1$ must have been sent by the SUT at a time in the region $[t_1 - \Delta_{UB}, t_1 - \Delta_{LB}]$ and $!o_2$ must have been sent by the SUT at a time in the region $[t_2 - \Delta_{UB}, t_2 - \Delta_{LB}]$. As a result, we can claim that if these two time intervals do not overlap then $!o_1$ was sent before $!o_2$ and this is the case when we have that $t_1 - \Delta_{LB} < t_2 - \Delta_{UB}$. This holds if and only if $t_2 - t_1 > \Delta_{UB} - \Delta_{LB}$. This case is shown in Figure 7, in which time progresses as we move to the right, a solid arc represents a message (an output), and the dotted arcs represent the bounds on when a message might have been sent.
- Let us suppose that input $?i_1$ is sent at port 1 at (global) time t_1 and that $?i_2$ is sent at port 2 at (global) time t_2 and we have that $t_2 > t_1$. Then $?i_1$ must arrive at the SUT at a time in the region $[t_1 + \Delta_{LB}, t_1 + \Delta_{UB}]$ and $?i_2$ must arrive at a time in the region $[t_2 + \Delta_{LB}, t_2 + \Delta_{UB}]$. Thus, we know that if these two time intervals do not overlap then $?i_1$ arrives before $?i_2$ and this is the case when $t_1 + \Delta_{UB} < t_2 + \Delta_{LB}$. This holds if and only if $t_2 - t_1 > \Delta_{UB} - \Delta_{LB}$. This case is shown in Figure 8.
- Now consider the situation in which we have an input and an output. If the output is observed by the tester before the input is sent then clearly the output was produced by the SUT before the input was received. It is therefore sufficient to consider the situation in which an input $?i_1$ is sent at time t_1 before an output $!o_2$ is observed at time t_2. Then, $?i_1$ must be received by the SUT before time $t_1 + \Delta_{UB}$ and $!o_2$ must have been sent by the SUT after time $t_2 - \Delta_{UB}$. Thus, we know that $!o_2$ was sent after $?i_1$ was received if $t_1 + \Delta_{UB} < t_2 - \Delta_{UB}$ and this is the case if and only if $t_2 - t_1 > 2\Delta_{UB}$. Similarly, we know that $!o_2$ was sent before $?i_1$ was received if $t_2 - \Delta_{LB} < t_1 + \Delta_{LB}$ and this is the case if and only if $t_2 - t_1 < 2\Delta_{LB}$. This generalises the situation in which an output is observed before an input, showing that we can potentially know that an output was produced before an input was received if the output is observed less than 2 Δ_{LB} after the in-

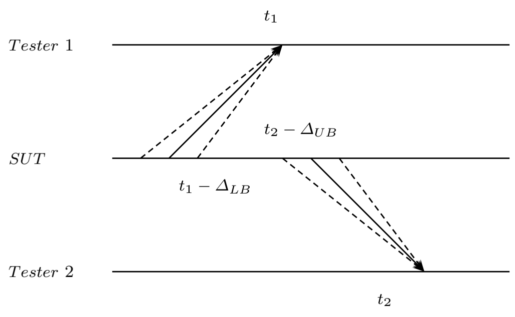

Fig. 7 Two outputs received by different testers.

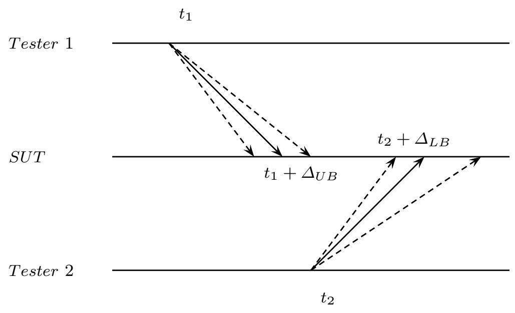

Fig. 8 Two inputs received from different testers.

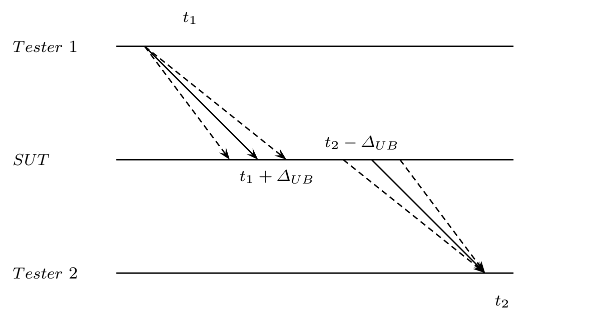

Fig. 9 An input before an output.

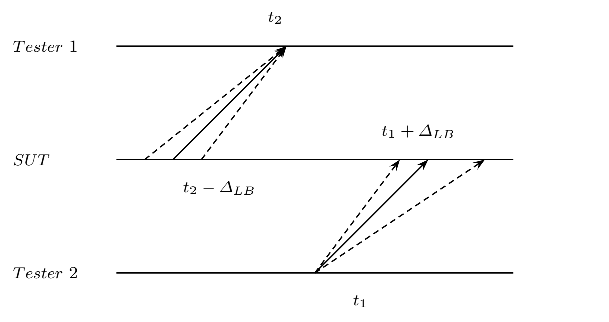

Fig. 10 An output before an input.

put was sent. These situations are shown in Figures 9 and 10.

Assuming communications are FIFO, we have to adapt our earlier relations to add this additional imprecision provided by the asynchronous nature of communications. In the rest of the paper we consider Δ_{LB} and Δ_{UB} to be *global parameters* and do not use these to decorate the notation of the different relations.

The first notion assumes the existence of a global clock or that local clocks work perfectly. It is based on the above discussion and the fact that communications are FIFO.

Definition 22 Let $\sigma = (a_1, t_1) \ldots (a_n, t_n) \in (\mathcal{A}ct \times \text{Time})^*$ be a timed trace. We define $\ll^F$ as the partial order on $e(\sigma)$ such that for all $1 \leq i, j, \leq n$ with $i \neq j$ we have that $e_\sigma(i) \ll^F e_\sigma(j)$ if and only if one or more of the following hold:

- There exists $p \in \mathcal{P}orts$ such that $a_i, a_j \in I_p \cup \{\delta\}$ and $t_i < t_j$;
- There exists $p \in \mathcal{P}orts$ such that $a_i, a_j \in O_p \cup \{\delta\}$ and $t_i < t_j$;
- We have $a_i, a_j \in I, i < j$ and $t_j - t_i > \Delta_{UB} - \Delta_{LB}$;
- We have $a_i, a_j \in O, i < j$ and $t_j - t_i > \Delta_{UB} - \Delta_{LB}$;
- We have $a_i \in O, a_j \in I$ and $t_j - t_i < 2\Delta_{LB}$;
- We have $a_i \in I, a_j \in O$ and $t_j - t_i > 2\Delta_{UB}$.

□

The first two cases are due to communications being FIFO. Cases three and four do not need to take into account the asymmetry between input and output since they are dealing with only inputs or only outputs. Case five is the situation in which an output is observed before an input is sent and we can deduce that the output was sent by the SUT before the input will be received. The final case concerns the sending of an input a_i before an output a_j is observed.

The discussion above, used to establish the partial order, was based about reasoning about the latest time event a_i might have been produced/observed by the SUT and the earliest time that a_j might have been produced/observed. We might therefore define functions *latest* and *earliest* that compute these bounds.

Definition 23 Let $\sigma = (a_1, t_1) \ldots (a_n, t_n) \in (\mathcal{A}ct \times \text{Time})^*$ be a timed trace and $1 \leq i \leq n$ such that $a_i \in I \cup O$. We define the values $earliest(a_i)$ and $latest(a_i)$ as:

- If $a_i \in I$ then $earliest(a_i) = t_i + \Delta_{LB}$ and $latest(a_i) = t_i + \Delta_{UB}$.
- If $a_i \in O$ then $earliest(a_i) = t_i - \Delta_{UB}$ and $latest(a_i) = t_i - \Delta_{LB}$.

□

The functions *earliest* and *latest* give us an alternative characterisation of the partial order $\ll^F$

Proposition 12 Let $\sigma = (a_1, t_1) \ldots (a_n, t_n) \in (\mathcal{A}ct \times \text{Time})^*$ be a timed trace. We have that $e_\sigma(i) \ll^F e_\sigma(j)$ if and only if one or more of the following hold:

- There exists $p \in \mathcal{P}orts$ such that $a_i, a_j \in I_p \cup \{\delta\}$ and $t_i < t_j$;
- There exists $p \in \mathcal{P}orts$ such that $a_i, a_j \in O_p \cup \{\delta\}$ and $t_i < t_j$;
- We have that $latest((a_i, t_i)) < earliest((a_j, t_j))$.

□

We will define the remaining implementation relations in terms of the functions *earliest* and *latest*.

First we assume that the local clocks differ by at most a known value α.

Definition 24 Let $\sigma = (a_1, t_1) \ldots (a_n, t_n) \in (\mathcal{A}ct \times \text{Time})^*$ be a timed trace and $\alpha \in \mathbb{R}_+$. We define $\ll^F_\alpha$ as the partial order on $e(\sigma)$ such that for all $1 \leq i, j, \leq n$ with $i \neq j$ we have that $e_\sigma(i) \ll^F_\alpha e_\sigma(j)$ if and only if one or more of the following hold:

- There exists $p \in \mathcal{P}orts$ such that $a_i, a_j \in I_p \cup \{\delta\}$ and $t_i < t_j$;
- There exists $p \in \mathcal{P}orts$ such that $a_i, a_j \in O_p \cup \{\delta\}$ and $t_i < t_j$;
- There exists $p \in \mathcal{P}orts$ such that $a_i, a_j \in I_p \cup O_p$ and $\mathit{latest}((a_i, t_i)) < \mathit{earliest}((a_j, t_j))$.
- We have that $\mathit{latest}((a_i, t_i)) - \mathit{earliest}((a_j, t_j)) > \alpha$.

□

The first three rules are about situations in which two events are observed at the same port; in this situation we do not have to consider the relative imprecision of the local clocks and so these rules are equivalent to rules from the definition of $\ll^F$. The remaining case involves observations at different ports.

Finally, we adapt the notions $\ll_{\alpha,\beta}$ and $\ll_h$ to the asynchronous FIFO framework; these are similar to $\ll_\alpha$ but with α replaced by the appropriate terms.

Definition 25 Let $\sigma = (a_1, t_1) \ldots (a_n, t_n) \in (\mathcal{A}ct \times \text{Time})^*$ be a timed trace. Given $\alpha, \beta \in \mathbb{R}_+$, we define $\ll^F_{\alpha,\beta}$ as the partial order on $e(\sigma)$ such that for all $1 \leq i, j, \leq n$ with $i \neq j$ we have that $e_\sigma(i) \ll^F_{\alpha,\beta} e_\sigma(j)$ if and only if one or more of the following hold:

- There exists $p \in \mathcal{P}orts$ such that $a_i, a_j \in I_p \cup \{\delta\}$ and $t_i < t_j$;
- There exists $p \in \mathcal{P}orts$ such that $a_i, a_j \in O_p \cup \{\delta\}$ and $t_i < t_j$;
- There exists $p \in \mathcal{P}orts$ such that $a_i, a_j \in I_p \cup O_p$ and $\mathit{latest}((a_i, t_i)) < \mathit{earliest}((a_j, t_j))$.
- We have that $\mathit{latest}((a_i, t_i)) - \mathit{earliest}((a_j, t_j)) > \alpha + \beta \cdot \max(t_i, t_j)$.

Given a monotonically increasing function $h : \mathbb{R}_+ \longrightarrow \mathbb{R}_+$, we define $\ll^F_h$ as the partial order on $e(\sigma)$ such that for all $1 \leq i, j, \leq n$ with $i \neq j$ we have that $e_\sigma(i) \ll^F_h e_\sigma(j)$ if and only if one or more of the following hold:

- There exists $p \in \mathcal{P}orts$ such that $a_i, a_j \in I_p \cup \{\delta\}$ and $t_i < t_j$;
- There exists $p \in \mathcal{P}orts$ such that $a_i, a_j \in O_p \cup \{\delta\}$ and $t_i < t_j$;
- There exists $p \in \mathcal{P}orts$ such that $a_i, a_j \in I_p \cup O_p$ and $\mathit{latest}((a_i, t_i)) < \mathit{earliest}((a_j, t_j))$.
- We have that $\mathit{latest}((a_i, t_i)) - \mathit{earliest}((a_j, t_j)) > h(\max(t_i, t_j))$.

□

In the next definition we introduce implementation relations for systems with asynchronous FIFO communications.

Definition 26 Let $r, s \in \mathtt{IOTS}(I, O, \mathcal{P}orts)$ and $\mathcal{T} \in \mathcal{P}((\mathcal{A}ct \times \text{Time})^*)$ be a set of quiescent timed traces of r. We write r $\mathbf{tdioco}^F(\mathcal{T})$ s if r $\mathbf{dioco}^F$ s and for every $\sigma \in \mathcal{T}$ there exists a quiescent trace $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \in L(e(\sigma), \ll^F)$.

Let $\alpha \in \mathbb{R}_+$ be a positive real number. We write r $\mathbf{tdioco}^F_\alpha(\mathcal{T})$ s if r $\mathbf{dioco}^F$ s and for every $\sigma \in \mathcal{T}$ there exists a quiescent trace $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \in L(e(\sigma), \ll^F_\alpha)$.

Let $\alpha, \beta \in \mathbb{R}_+$ be positive real numbers. We write r $\mathbf{tdioco}^F_{\alpha,\beta}(\mathcal{T})$ s if r $\mathbf{dioco}^F$ s and for every $\sigma \in \mathcal{T}$ there exists a quiescent trace $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \in L(e(\sigma), \ll^F_{\alpha,\beta})$.

Let $h : \mathbb{R}_+ \longrightarrow \mathbb{R}_+$ be a monotonically increasing function on $\mathbb{R}_+$. We write r $\mathbf{tdioco}^F_h(\mathcal{T})$ s if r $\mathbf{dioco}^F$ s and for every $\sigma \in \mathcal{T}$ there exists a quiescent trace $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \in L(e(\sigma), \ll^F_h)$. □

Example 9 Consider the specification of the collaborative editor depicted in Figure 4 and presented in Example 1. Let r be an SUT such that its conformance to the `EDShared` specification with respect to $\mathbf{dioco}^F$ has been previously established. Consider that the set of traces $\mathcal{T}$, conformed by the traces tr_1 and tr_2, has been observed while testing the system:

$$\begin{aligned}
tr_1 = {} & (?open_1, 0.2), (?open_2, 0.25), (!edit_1, 0.25),\\
& (!edit_2, 0.27), (?save_2, 1), (?save_1, 1.05),\\
& (!update_2, 1.11), (!edit_2, 1.14), (!edit_1, 1.15),\\
& (!update_1, 1.16), (!edit_1, 1.2), (!edit_2, 1.3)\\
tr_2 = {} & (?open_1, 1), (?open_2, 1.2), (!edit_1, 1.3),\\
& (!edit_2, 1.4), (?save_1, 2), (?save_2, 2.3),\\
& (!edit_1, 2.4), (!update_2, 2.5), (!update_1, 2.6),\\
& (!edit_2, 2.7), (!edit_2, 2.8), (!edit_1, 2.9)
\end{aligned}$$

If we consider *big* values for the message latency, for example $\Delta_{LB} = 0.5$ and $\Delta_{UB} = 0.9$ ($\Delta_{UB} - \Delta_{LB} = 0.4$), then these traces cannot be used to conclude that the system r is an incorrect implementation of the `EDShared` specification with respect to any of the implementation relations given in Definition 26. It is possible that the observation of the subsequence

$$?save_2?save_1!update_2!edit_2!edit_1!update_1!edit_1!edit_2$$

appearing in trace tr_1 was due to message latency, in the case of the inputs $(?save_2, 1), (?save_1, 1.1)$, and to the delay of the outputs $(!update_2, 1.11), (!edit_2, 1.14)$ and $(!edit_1, 1.15)$. In the same way in trace tr_2 it can happen that the outputs $(!edit_1, 2.4), (!update_2, 2.5)$ and $(!update_1, 2.6)$ were delayed with respect to the input $(?save_2, 2.3)$; in particular, the output $(!edit_1, 2.4)$ could have been produced after the outputs $(!update_2, 2.5)$ and $(!update_1, 2.6)$

On the contrary, if we have that message latency is small, for example $\Delta_{LB} = 0.025$ and $\Delta_{UB} = 0.05$ ($\Delta_{UB} - \Delta_{LB} = 0.025$), none of the previous implementation relations hold.

This example shows an interesting property of the asynchronous framework: when $\Delta_{UB} - \Delta_{LB}$ tends to zero we have that the asynchronous FIFO implementation relations tend to be equal to their synchronous counterparts. □

Results similar to the ones shown at the end of the previous section hold for the new relations. For the sake of brevity, we do not present all of these.

Proposition 13 Let $\mathcal{T}$ be a set of timed traces and $\alpha, \beta \in \mathbb{R}_+$. We have $\textbf{tdioco}^F_\alpha(\mathcal{T}) \sqsubset \textbf{tdioco}^F(\mathcal{T})$ and $\textbf{tdioco}^F_{\alpha,\beta}(\mathcal{T}) \sqsubset \textbf{tdioco}^F_\alpha(\mathcal{T})$. □

As a final remark, note that in this section we assumed that bounds on message latency are identical for all channels and in both directions. In practice, different channels might have different timing properties but it is straightforward to adapt the definitions and results given here to this situation.

7 Asynchronous non-FIFO communications

In this section we consider a communications framework where the FIFO hypothesis is not assumed. In order to adapt our relations we change the rules that differentiate between observations at a single port. We still need to distinguish between the cases where events are at different ports and when they are at the same port; in the latter case the timestamps are produced on the basis of the same local clock.

The following notion is suitable if we assume the existence of a global clock or that local clocks work perfectly.

Definition 27 Let $\sigma = (a_1, t_1) \dots (a_n, t_n) \in (\mathcal{A}ct \times \text{Time})^*$ be a timed trace. We define $\ll^N$ as the partial order on $e(\sigma)$ such that for all $1 \leq i, j \leq n$ with $i \neq j$ we have that $e_\sigma(i) \ll^N e_\sigma(j)$ if and only if one or more of the following hold:

- We have $a_i = \delta$ or $a_j = \delta$ and $t_j > t_i$;
- We have $a_i, a_j \in I \cup O$ and $\mathit{latest}(t_i) < \mathit{earliest}(t_j)$.

□

We no longer have separate rules for the cases, in the definition of $\ll^F$, where two inputs or two outputs are at the same port. This is because these rules were a result of using FIFO channels. We do, however, have to separately consider the observation of quiescence; an alternative would be to extend the definition of *latest* and *earliest* to deal with quiescence.

We now adapt $\ll_\alpha$ to incorporate the delay introduced by non-FIFO communications.

Definition 28 Let $\sigma = (a_1, t_1) \dots (a_n, t_n) \in (\mathcal{A}ct \times \text{Time})^*$ be a timed trace and $\alpha \in \mathbb{R}_+$. We define $\ll^N_\alpha$ as the partial order on $e(\sigma)$ such that for all $1 \leq i, j, \leq n$ with $i \neq j$ we have that $e_\sigma(i) \ll^N_\alpha e_\sigma(j)$ if and only if one or more of the following hold:

- We have $a_i = \delta$ or $a_j = \delta$ and $t_j > t_i$;
- We have $a_i, a_j \in I_p \cup O_p$ for some $p \in \mathcal{P}orts$ and $\mathit{earliest}(t_j) > \mathit{latest}(t_i)$.
- We have $a_i, a_j \in I \cup O$ and $\mathit{earliest}(t_j) - \mathit{latest}(t_i) > \alpha$.

□

Finally, we adapt $\ll_{\alpha,\beta}$ and $\ll_h$ to the new framework.

Definition 29 Let $\sigma = (a_1, t_1) \dots (a_n, t_n) \in (\mathcal{A}ct \times \text{Time})^*$ be a timed trace and $\alpha, \beta \in \mathbb{R}_+$. We define $\ll^N_{\alpha,\beta}$ as the partial order on $e(\sigma)$ such that for all $1 \leq i, j, \leq n$ with $i \neq j$ we have that $e_\sigma(i) \ll^N_{\alpha,\beta} e_\sigma(j)$ if and only if one or more of the following hold:

- We have $a_i = \delta$ or $a_j = \delta$ and $t_j > t_i$;
- We have $a_i, a_j \in I_p \cup O_p$ for some $p \in \mathcal{P}orts$ and $\mathit{earliest}(t_j) > \mathit{latest}(t_i)$.
- We have $a_i, a_j \in I \cup O$ and $\mathit{earliest}(t_j) - \mathit{latest}(t_i) > \alpha + \beta \cdot \max(t_i, t_j)$.

Given a monotonically increasing function $h : \mathbb{R}_+ \longrightarrow \mathbb{R}_+$, $\ll^N_h$ is the partial order on $e(\sigma)$ such that for all $1 \leq i, j, \leq n$ with $i \neq j$ we have that $e_\sigma(i) \ll^N_h e_\sigma(j)$ if and only if one or more of the following hold:

- We have $a_i = \delta$ or $a_j = \delta$ and $t_j > t_i$;
- We have $a_i, a_j \in I_p \cup O_p$ for some $p \in \mathcal{P}orts$ and $\mathit{earliest}(t_j) > \mathit{latest}(t_i)$.
- We have $a_i, a_j \in I \cup O$ and $\mathit{earliest}(t_j) - \mathit{latest}(t_i) > h(\max(t_i, t_j))$.

□

The combination of the previously defined partial orders and the $\textbf{dioco}^N$ relation allows us to define implementation relations for non-FIFO communications.

Definition 30 Let $r, s \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ and $\mathcal{T} \in \mathcal{P}((\mathcal{A}ct \times \text{Time})^*)$ be a set of quiescent timed traces of r. We write $r\ \textbf{tdioco}^N(\mathcal{T})\ s$ if $r\ \textbf{dioco}^N\ s$ and for every $\sigma \in \mathcal{T}$ there exists a quiescent trace $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \in L(e(\sigma), \ll^N)$.

Let $\alpha \in \mathbb{R}_+$ be a positive real number. We write $r\ \textbf{tdioco}^N_\alpha(\mathcal{T})\ s$ if $r\ \textbf{dioco}^N\ s$ and for every $\sigma \in \mathcal{T}$

there exists a quiescent trace $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \in L(e(\sigma), \ll_{\alpha}^{N})$.

Let $\alpha, \beta \in \mathbb{R}_+$ be positive real numbers. We write r $\mathbf{tdioco}_{\alpha,\beta}^{N}(\mathcal{T})$ s if r $\mathbf{dioco}^{N}$ s and for every $\sigma \in \mathcal{T}$ there exists a quiescent trace $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \in L(e(\sigma), \ll_{\alpha,\beta}^{N})$.

Let $h : \mathbb{R}_+ \longrightarrow \mathbb{R}_+$ be a monotonically increasing function on $\mathbb{R}_+$. We write r $\mathbf{tdioco}_{h}^{N}(\mathcal{T})$ s if r $\mathbf{dioco}^{N}$ s and for every $\sigma \in \mathcal{T}$ there exists a quiescent trace $\sigma' \in \mathcal{T}r(s)$ such that $\sigma' \in L(e(\sigma), \ll_{h}^{N})$. □

Results similar to the ones shown at the end of Section 5 hold again for the new relations. For the sake of brevity, we present only one of them.

Proposition 14 Let $\mathcal{T}$ be a set of timed traces and $\alpha, \beta \in \mathbb{R}_+$. We have that $\mathbf{tdioco}_{\alpha}^{N}(\mathcal{T}) \sqsubset \mathbf{tdioco}^{N}(\mathcal{T})$ and $\mathbf{tdioco}_{\alpha,\beta}^{N}(\mathcal{T}) \sqsubset \mathbf{tdioco}_{\alpha}^{N}(\mathcal{T})$. □

Finally, we compare the synchronous, FIFO, and non-FIFO cases.

Proposition 15 The following results hold.

1. $\mathbf{tdioco}^{N}(\mathcal{T}) \sqsubset \mathbf{tdioco}^{F}(\mathcal{T}) \sqsubset \mathbf{tdioco}(\mathcal{T})$
2. $\mathbf{tdioco}_{\alpha}^{N}(\mathcal{T}) \sqsubset \mathbf{tdioco}_{\alpha}^{F}(\mathcal{T}) \sqsubset \mathbf{tdioco}_{\alpha}(\mathcal{T})$
3. $\mathbf{tdioco}_{\alpha,\beta}^{N}(\mathcal{T}) \sqsubset \mathbf{tdioco}_{\alpha,\beta}^{F}(\mathcal{T}) \sqsubset \mathbf{tdioco}_{\alpha,\beta}(\mathcal{T})$
4. $\mathbf{tdioco}_{h}^{N}(\mathcal{T}) \sqsubset \mathbf{tdioco}_{h}^{F}(\mathcal{T}) \sqsubset \mathbf{tdioco}_{h}(\mathcal{T})$

□

8 The Oracle problem in timed distributed testing

This paper defined several implementation relations but we have not yet discussed the Oracle Problem: that of determining whether an observation made in testing (a timed trace) is allowed under one of these implementation relations. It is known that the Oracle Problem is NP-complete for untimed distributed testing from an FSM [16]. In this section we explain how one could show that the Oracle Problem is also NP-complete for any of the implementation relations defined in this paper as long as the membership problem can be solved in polynomial time for the specification[6].

The first observation that can be made is that it is possible to take an instance of the Oracle Problem for an FSM and convert it into an equivalent instance of the Oracle Problem for an implementation relation given in this paper: we simply add timestamps that are sufficiently close together for one not to be able to deduce any additional ordering information from the timestamps. Thus, the Oracle Problem being NP-hard for the implementation relations given in this paper is a consequence of it being NP-hard for distributed testing from an FSM.

We now explain how we can show that the Oracle Problem is in NP. A first step in solving the Oracle Problem for a given timed trace and implementation relation is to determine the partial order defined over the events in the timed trace. It is clear that this can be decided in polynomial time unless the implementation relation is parameterised by a function h that cannot be computed in polynomial time (we will assume that this is not the case). Having done this, the problem is to determine whether there is some linearisation of the partial order that is allowed by the specification (is in the language defined by the specification). A non-deterministic Turing machine can guess a linearisation and then check this against the specification, a process that can be performed in polynomial time as long as the membership problem can be solved in polynomial time. Thus, we know that the Oracle Problem is in NP (and so is NP-complete) as long as the following two conditions hold:

1. The membership problem can be solved in polynomial time for the specification.
2. If the implementation relation is parameterised by a function h then h can be computed in polynomial time.

Naturally, if either of these conditions were not to hold then we could not expect the Oracle Problem to be in NP.

9 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 regarding how much the local clocks can differ and we can represent this using assumptions. 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

[6] The membership problem is that of determining whether a given trace is a trace of the specification.

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, from the positive reals to the positive reals, 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.

The above approach assumes that each tester interacts synchronously with the corresponding port of the SUT. However, sometimes a tester will interact with the SUT through a network and this can lead to an input being received by the SUT after it was sent by the tester and an output being observed by a tester after it was produced by the SUT. This can result in the tester at port p not observing the local trace produced by the SUT at p: instead the tester observes a version of this produced by delaying output. Naturally, this leads to different implementation relations. We considered the cases where communications are FIFO and where they are non-FIFO and produced implementation relations for each of the assumptions, regarding how the local clocks relate, described above.

There are several possible lines of future work. First, we would like to explore relationships in extreme cases, that is, the consequence of assuming that the discrepancies between clocks tend to zero/infinity. We can integrate our approach with methods for establishing local clocks through message exchange. A possible way to implement this could consist in having some sort of general *synch* event that returns precision to within α. 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 cannot be distinguished 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 anonymous reviewers for the careful reading and for their constructive comments that have helped to further strengthen the paper.

References

1. Ammann, P., Offutt, J.: Introduction to Software Testing. Cambridge University Press (2008)
2. Bhateja, P., Gastin, P., Mukund, M.: A fresh look at testing for asynchronous communication. In: 4th Int. Symp. on Automated Technology for verification and Analysis, ATVA'06, LNCS 4218, pp. 369–383. Springer (2006)
3. Boreale, M., Nicola, R.d., Pugliese, R.: Asynchronous observations of processes. In: 1st Int. Conf. on the Foundations of Software Science and Computation Structure, FoSSaCS'98, LNCS 1378, pp. 95–109. Springer (1998)
4. Boreale, M., Nicola, R.d., Pugliese, R.: A theory of "may" testing for asynchronous languages. In: 2nd Int. Conf. on the Foundations of Software Science and Computation Structure, FoSSaCS'99, LNCS 1578, pp. 165–179. Springer (1999)
5. Boyd, S., Ural, H.: The synchronization problem in protocol testing and its complexity. Information Processing Letters **40**(3), 131–136 (1991)
6. Cacciari, L., Rafiq, O.: Controllability and observability in distributed testing. Information and Software Technology **41**(11–12), 767–780 (1999)
7. Castellani, I., Hennessy, M.: Testing theories for asynchronous languages. In: 18th Conf. on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'98, LNCS 1530, pp. 90–101. Springer (1998)
8. Chen, W., Ural, H.: Synchronizable checking sequences based on multiple UIO sequences. IEEE/ACM Transactions on Networking **3**, 152–157 (1995)
9. Cunha de Almeida, E., Marynowski, J., Sunyé, G., Traon, Y.L., Valduriez, P.: Efficient distributed test architecture for large-scale systems. In: 22nd Int. Conf. on Testing Software and Systems, ICTSS'10, LNCS 6435, pp. 174–187. Springer (2010)
10. Dssouli, R., Bochmann, G.v.: Conformance testing with multiple observers. In: 6th WG6.1 Int. Conf. on Protocol Specification, Testing and Verification, PSTV'86, pp. 217–229. North-Holland (1986)
11. Farchi, E., Hartman, A., Pinter, S.: Using a model-based test generator to test for standard conformance. IBM Systems Journal **41**(1), 89–110 (2002)
12. Fidge, C.: A limitation of vector timestamps for reconstructing distributed computations. Information Processing Letters **68**(2), 87–91 (1998)
13. Gaudel, M.C.: Testing can be formal, too! In: 6th Int. Joint Conf. CAAP/FASE, Theory and Practice of Software Development, TAPSOFT'95, LNCS 915, pp. 82–96. Springer (1995)
14. Gonenc, G.: A method for the design of fault detection experiments. IEEE Transactions on Computers **19**, 551–558 (1970)
15. 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), 55–71 (2011)
16. Hierons, R.M.: Oracles for distributed testing. IEEE Transactions on Software Engineering **38**(3), 629–641 (2012)

17. 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 specifications to support testing. ACM Computing Surveys **41**(2) (2009)
18. Hierons, R.M., Bowen, J., Harman, M. (eds.): Formal Methods and Testing, LNCS 4949. Springer (2008)
19. Hierons, R.M., Merayo, M.G., Núñez, M.: Controllable test cases for the distributed test architecture. In: 6th Int. Symposium on Automated Technology for Verification and Analysis, ATVA'08, LNCS 5311, pp. 201–215. Springer (2008)
20. Hierons, R.M., Merayo, M.G., Núñez, M.: Implementation relations for the distributed test architecture. In: Joint 20th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'08, and 8th Int. Workshop on Formal Approaches to Software Testing, FATES'08, LNCS 5047, pp. 200–215. Springer (2008)
21. Hierons, R.M., Merayo, M.G., Núñez, M.: Implementation relations and test generation for systems with distributed interfaces. Distributed Computing **25**(1), 35–62 (2012)
22. Hierons, R.M., Merayo, M.G., Núñez, M.: Using time to add order to distributed testing. In: 18th Symposium on Formal Methods, FM'12, LNCS 7436, pp. 232–246. Springer (2012)
23. Hierons, R.M., Ural, H.: Checking sequences for distributed test architectures. Distributed Computing **21**(3), 223–238 (2008)
24. 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)
25. 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, pp. 353–368. Kluwer Academic Publishers (1998)
26. 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, pp. 25–40. Kluwer Academic Publishers (1999)
27. Kim, M., Shin, J., Chanson, S.T., Kang, S.: An enhanced model for testing asynchronous communicating systems. In: 19th Joint Int. Conf. on Protocol Specification, Testing, and Verification and Formal Description Techniques, FORTE/PSTV'99, pp. 337–356. Kluwer Academic Publishers (1999)
28. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM **21**(7), 558–565 (1978)
29. Lamport, L., Melliar-Smith, P.: Synchronizing clocks in the presence of faults. Journal of the ACM **32**(1), 52–78 (1985)
30. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines: A survey. Proceedings of the IEEE **84**(8), 1090–1123 (1996)
31. Luo, G., Dssouli, R., Bochmann, G.v.: Generating synchronizable test sequences based on finite state machine with distributed ports. In: 6th IFIP Workshop on Protocol Test Systems, IWPTS'93, pp. 139–153. North-Holland (1993)
32. Moore, E.: Gedanken experiments on sequential machines. In: C. Shannon, J. McCarthy (eds.) Automata Studies. Princeton University Press (1956)
33. Myers, G.: The Art of Software Testing, 2nd edn. John Wiley and Sons (2004)
34. Ostrovsky, R., B.Patt-Shamir: Optimal and efficient clock synchronization under drifting clocks. In: 18th Annual ACM Symposium on Principles of Distributed Computing, PODC'99, pp. 3–12. ACM Press (1999)
35. Petrenko, A., Yevtushenko, N.: Testing from partial deterministic FSM specifications. IEEE Transactions on Computers **54**(9), 1154–1165 (2005)
36. Rafiq, O., Cacciari, L.: Coordination algorithm for distributed testing. The Journal of Supercomputing **24**(2), 203–211 (2003)
37. Sarikaya, B., Bochmann, G.v.: Synchronization and specification issues in protocol testing. IEEE Transactions on Communications **32**, 389–395 (1984)
38. Tai, K.C., Young, Y.C.: Synchronizable test sequences of finite state machines. Computer Networks and ISDN Systems **30**(12), 1111–1134 (1998)
39. Tretmans, J.: Conformance testing with labelled transition systems: Implementation relations and test generation. Computer Networks and ISDN Systems **29**, 49–79 (1996)
40. Tretmans, J.: Model based testing with labelled transition systems. In: Formal Methods and Testing, LNCS 4949, pp. 1–38. Springer (2008)
41. Ural, H., Williams, C.: Constructing checking sequences for distributed testing. Formal Aspects of Computing **18**(1), 84–101 (2006)
42. Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan-Kaufmann (2007)