SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 0000; **00**:1–28
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe

Scenarios-based testing of systems with distributed ports

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

[1]*Department of Information Systems and Computing*
Brunel University, Uxbridge, Middlesex, UB8 3PH United Kingdom
[2]*Departamento de Sistemas Informáticos y Computación*
Universidad Complutense de Madrid, Madrid, Spain

SUMMARY

Distributed systems are usually composed of several distributed components that communicate with their environment through specific ports. When testing such a system we separately observe sequences of inputs and outputs at each port rather than a global sequence and potentially cannot reconstruct the global sequence that occurred. Typically, the users of such a system cannot synchronise their actions during use or testing. However, the use of the system might correspond to a sequence of scenarios, where each scenario involves a sequence of interactions with the system that, for example, achieves a particular objective. When this is the case there is the potential for there to be a significant delay between two scenarios and this effectively allows the users of the system to synchronise between scenarios. If we represent the specification of the global system by using a state-based notation, we say that a *scenario* is any sequence of events that happens between two of these operations. We can encode scenarios in two different ways. The first approach consists of marking some of the states of the specification to denote these synchronisation points. It transpires that there are two ways to interpret such models and these lead to two implementation relations. The second approach consists of adding a set of traces to the specification to represent the traces that correspond to scenarios. We show that these two approaches have similar expressive power by providing an encoding from marked states to sets of traces. In order to assess the appropriateness of our new framework, we show that it represents a conservative extension of previous implementation relations defined in the context of the distributed test architecture: if we consider that all the states are marked then we simply obtain **ioco** (the classical relation for single-port systems) while if no state is marked then we obtain **dioco** (our previous relation for multi-port systems). Finally, we concentrate on the study of *controllable* test cases, that is, test cases such that each local tester knows exactly when to apply inputs. We give two notions of controllable test cases, define an implementation relation for each of these notions, and relate them. We also show how we can decide whether a test case satisfies these conditions. Copyright © 0000 John Wiley & Sons, Ltd.

Received ...

KEY WORDS: formal testing; systems with distributed ports; scenarios; implementation relations; controllable testing

1. INTRODUCTION

Software engineering techniques based on a formal foundation are necessary to assist in the production of reliable software. A first step to ensure that we are developing the correct system

*Correspondence to: Robert M. Hierons, rob.hierons@brunel.ac.uk

Contract/grant sponsor: Research partially supported by the Spanish MEC project TESIS (TIN2009-14312-C02-01), the UK EPSRC project Testing of Probabilistic and Stochastic Systems (EP/G032572/1), and the UCM-BSCH programme to fund research groups (GR58/08 - group number 910606).

is to have a formal specification of its behaviour. In order to make sure that this model is sound, it is necessary to *verify* the specification with respect to the requirements of the system. In this line, *model checking* [1, 2] represents an appropriate technique to decide whether the specification fulfils the desired requirements. However, a correct specification does not imply that we will obtain a correct system.

The technique most widely used to assess the correctness of software systems is *software testing* [3, 4]. Even though formal methods and testing have been seen as rivals, so that there was very little interaction between the two communities, these approaches are now seen as complementary and the last ten years have shown significant progress in the area [5, 6, 7, 8]. The main advantage of using a formal approach is that many testing processes can be automated (see [9] for a discussion on the advantages of formal testing and [10] for a survey on formal methods and testing). In order to formally state what a *correct* implementation is, we need to define an implementation relation to relate implementations and specifications. These relations can be given in terms of the test cases, possibly extracted from a specification, that are successfully passed by an implementation.

An important class of systems is the one where the system under test (SUT) has physically distributed interfaces/ports. If we apply testing techniques to these systems, then it is normal to place a tester at each port. If we consider a black-box framework, there is no global clock, and the testers cannot directly communicate with each other during testing then we are testing in the distributed test architecture, which has been standardised by the ISO [11]. It is already well known that the use of the distributed test architecture reduces test effectiveness (see, for example, [12, 13, 14, 15, 16, 17, 18, 19]).

Most previous work on testing in the distributed test architecture has considered Deterministic Finite State Machines (DFSMs). However, the Input Output Transition System ($\mathcal{IOTS}$) formalism is more general: in a DFSM input and output alternate and DFSMs have a finite state structure and are deterministic. The last restriction is particularly problematic since distributed systems are often nondeterministic. While the implementation relation **ioco** [20, 21], that is usually used in testing from an $\mathcal{IOTS}$, has been adapted in a number of ways and extended to cope with issues such as time and probabilities, only recently has the problem of testing from an $\mathcal{IOTS}$ in the distributed test architecture been investigated [22, 23]. An implementation relation **mioco** had been defined for testing from an $\mathcal{IOTS}$ that has multiple ports. This implementation relation assumes that there is a single tester that controls and observes the ports [24] and so observes the global sequence of events that occurs (a global trace). However, when events occur at physically distributed ports we may instead observe a set of projections of the global sequence of events that occurred, one projection (local trace) for each port. As a result, we may not be able to reconstruct the global trace that occurred and this should be reflected by the implementation relation used. Our previous work introduced an implementation relation **dioco** for distributed testing. Under **dioco** it is assumed that the local traces that occurred at the different ports can be brought together once testing has finished. As a result, we are only allowed to compare traces of the SUT and of the specification when we reach *quiescent* states, that is, states that are *stable* in the sense that they cannot perform any output without receiving additional input. Since it is usually assumed that quiescence can be observed, the idea is that in quiescent states the local testers can send the traces collected so far so that they can be put together and checked against the specification (a longer discussion about this issue can be found in [22]).

It is clear that the distinguishing power of our **dioco** relation is weaker than the one corresponding to the classical **ioco** relation. Let us consider Figure 1. Actions preceded by ? are inputs while the ones preceded by ! are outputs. For the sake of clarity, most examples given in this paper consider a distributed architecture with two ports but our running example considers a system with three ports. In any case, the framework is presented for the general case where there are n ports. In the examples, we will usually call the ports U and L, and subindexes will denote at which port the action is performed. We have that M_2 (right-hand side of Figure 1) is not a good implementation of M_1 (left-hand side of Figure 1) according

DOI: 10.1002/spe

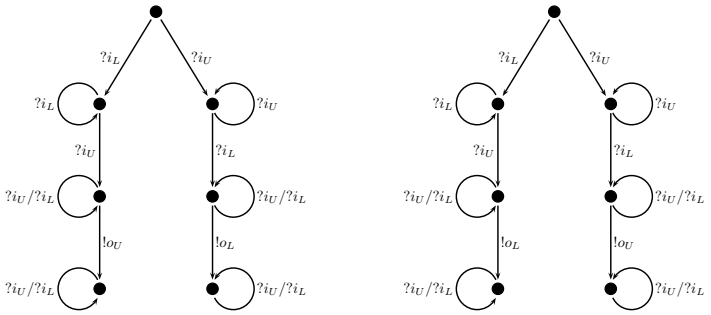


Figure 1. M_1 (left) and M_2 (right) are not related under **ioco** but are related under **dioco**.

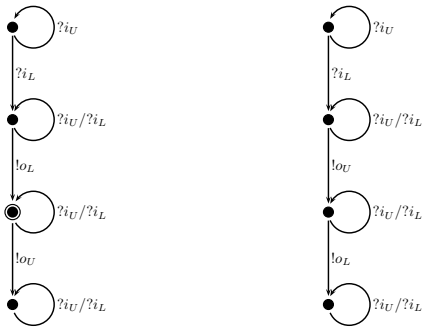


Figure 2. M_3 (left) and M_4 (right) are related under **dioco** but are not related under **sdioco**.

to **ioco** because we can find sequences of actions that can be performed by M_2 that cannot be performed by M_1. For example, $?i_U?i_L!o_U$ is such a sequence. However, M_2 is a good implementation for the distributed version of **ioco** because we do not simply compare traces, but compare them up to causality relations in the same port. For example, we consider that the trace $?i_L?i_U!o_U$ of M_1 is *equivalent* to the trace $?i_U?i_L!o_U$ of M_2 since in each case we observe $?i_U!o_U$ at port U and $?i_L$ at port L. However, a trace such as $?i_L!o_U?i_U$ would not be equivalent to the previous ones because we are changing the order in which certain actions are performed at port U.

This paper extends the study of distributed testing by allowing additional opportunities to combine local observations. Our previous work assumed that the different components have completely independent behaviours. The only way to partially *synchronise* them was by putting together the traces observed by local testers at each port when reaching quiescent states. Now let us suppose that agents A and B interact with the SUT at physically distributed ports. Under the **dioco** framework all that A and B know is that the local traces they observe are projections of the global trace that occurred. Let us suppose, however, that A observes event a on January 10th and B observes event b on February 5th of the same year. If A and B later communicate then they can deduce that a occurred before b even if they cannot reconstruct the global trace. Scenarios allow us to capture the notion of a 'complete use' of a distributed system, the idea being that different complete uses (traces) σ and σ' can occur sufficiently far apart in time for us to be able to know that all the events in σ occurred before all of the events in σ' even if we cannot construct the global trace that occurred. In this paper we investigate two ways of representing scenarios: identifying either states or traces. Let us consider, for example, Figure 2 in which the system M_3 (left-hand side of Figure 2) has a marked state. The idea is that testers (and users) can synchronise in marked states. Therefore, M_4 (right-hand side of Figure 2) is not a good implementation according to the new relation because, for example, it can perform the sequence $?i_L!o_U!o_L$ but for the specification to perform an equivalent trace

DOI: 10.1002/spe

it must perform $?i_L!o_L!o_U$ and the testers must be able to synchronise after $?i_L!o_L$; in M_3 we cannot do this (we cannot perform a trace equivalent to $?i_L!o_L$). However, M_4 conforms to M_3 if we consider **dioco**. An additional motivation is that there are situations when we need all the components of the system to have performed a certain set of tasks before we let them proceed with further computations. For example, this happens if we have a central database that has to be regularly updated: we have to make sure that all the distributed components accessing the database are not performing queries while the update takes place. If we have a state-based specification of the system we can mark some of its states so that we force the (distributed) implementation to perform such a sequence of events, up to the causality relation underlying **dioco**, that takes the specification from its initial state to one of these marked states.

The implementation relation introduced in this paper is called **sdioco**, standing for scenario-based **dioco** relation. Intuitively, a scenario is any sequence of events that takes the specification to one of its marked states. More precisely, scenarios are associated with sequences that bring the specification from one marked state to another one without traversing any marked states. Therefore, it seems natural to define an alternative framework by associating a set of traces with a specification. This set of traces contains each specific scenario that a correct implementation can show. The idea is that while **dioco**, our previous implementation relation where scenarios were not considered, allowed an implementation to produce any permutation of a trace of a specification as long as the order between actions at the same port did not vary, in the new setting the implementation will be allowed to produce only a subset of these permutations. In addition to **sdioco**, we define two stronger implementation relations **sdioco**$'$ and **sdioco**$''$ that also require states to be marked. We explore the properties of these implementation relations and demonstrate that **sdioco**$''$, while a natural extension, is too strong. We define another implementation relation, that we call **tsdioco**, for the new specification framework in which we identify traces rather than states and we provide an algorithm to transform specifications with marked states into specifications with a set of traces.

In order to show that our new relations are suitable extensions of the previously mentioned implementation relations, we prove that if no state of the specification is marked then **sdioco** and **dioco** coincide while if all the states are marked then **sdioco** and **ioco** are equal. A similar result is given for the **tsdioco** relation.

We give two notions of test case: a global test case provides a complete testing plan for the whole system while a local test case contains a specific testing plan for each port, called a local tester. We define the meaning of an implementation passing a (local or global) test case with respect to a specification. A particular class of test cases is the one including those test cases such that there does not exist a situation where a local tester has observed a trace after which either it should apply an input or wait for output, depending on what has happened at the other ports. The problem is that in such situations local testers do not know when they have to apply their input, so that the restriction to *controllable* test cases is very relevant. We provide a new notion of controllable test cases for the scenario-based framework and an algorithm to decide whether a test case is controllable. This algorithm has low-order polynomial complexity in terms of the size of the composition of the test case and the specification.

This paper represents an enhanced, revised, and extended version of [25]. We have included new additional explanations and examples to illustrate most of the concepts given in the paper. We have also added a running example. The implementation relation **sdioco**$'$ is entirely new and we show how **sdioco** and **sdioco**$'$ relate. We explore properties of **sdioco** and **sdioco**$'$ and show that **sdioco**$'$ is not compositional in that an implementation might conform to specifications s_1 and s_2 but not to specification $s_1 + s_2$ that can choose to behave like either s_1 or s_2. It transpires that this is a result of an interesting property of these implementation relations: the marking of states does not provide additional behaviour but instead provides additional opportunities to observe aspects of the behaviour. We explore an implementation relation **sdioco**$''$ that is a natural strengthening of **sdioco**$'$. We show that **sdioco**$''$ is too strong: we can have a model s such that s does not conform to itself under **sdioco**$''$. We also explore the effect of changing the set of marked states when using **sdioco** and **sdioco**$'$ and

DOI: 10.1002/spe

show how some of the possible changes can be seen as forms of refinement. The alternative view based on a set of traces is completely new and so, naturally, is the transformation from a specification with marked states to a specification with traces. Finally, the part of the paper dealing with controllable test cases has been extended and improved. In particular, we provide a new algorithm to decide whether a test case is weakly controllable[†] and this did not appear in [25].

Concerning related work, as we already mentioned, the introduction of scenarios is a novel and useful extension of our previous work on **dioco** [22, 23]. As far as we know, previous work on the distributed testing architecture did not consider any possibility similar to our scenarios: either there was no control over the actions performed at different ports (such as our **dioco** relation) or, by using synchronisation mechanisms, there was a complete control on the order in which actions were performed at different ports. Our scenarios give the freedom of determining how much control the specifier would like to have. This is the main novelty and advantage of the approach reported in this paper. As already discussed, previous work has defined an implementation relation **mioco** [24] but this implementation relation assumes that global traces are observed. There has also been work on distributing testers [26] but again this assumes that global observations are made. The notion of CSP refinement has been explored for the case where observations are distributed [27] but this work does not consider scenarios. Recent work has described models for distributed systems in which each transition has a partial order involving inputs and outputs rather than a single input or output [28, 29]. Again, this work did not consider the potential role of scenarios. Finally, there has been work on testing systems that interact with their environment through queues [30] and this is conceptually related to distributed testing since we cannot know the actual order of events that the SUT produced since outputs are observed after they are sent.

The rest of the paper is structured as follows. In Section 2 we provide preliminary definitions. Section 3 gives our formalism to define distributed systems with scenarios by marking states and three implementation relations that use marked states, two of these implementation relations being new. In Section 4 we present an alternative characterisation of the previous framework by adding a set of traces to specifications. We show that the two approaches have similar expressive power and relate them. In Section 5 we show how test cases are applied to SUTs, study the notion of controllability in the new framework, and give a new implementation relation based on controllable test cases, that is, tests cases where the order of application of inputs at different ports is completely determined. Finally, in Section 6 we present our conclusions and some lines for future work.

2. DEFINITION OF SYSTEMS AND IMPLEMENTATION RELATIONS

This section defines Input Output Transition Systems and associated notation, outlines the distributed test architecture, and introduces the new formalism to specify systems with scenarios in the distributed test architecture.

2.1. Input Output Transition Systems

We use *Input Output Transition Systems* to describe systems. These are labelled transition systems in which we distinguish between inputs and outputs [20, 21].

Definition 1
An *Input Output Transition System* ($\mathcal{IOTS}$) is defined by $M = (Q, I, O, T, q_{in})$ 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 $T \subseteq Q \times (I \cup O) \times Q$ is the transition relation. A transition

[†]A test case being weakly controllable differs from it being controllable in that we can take advantage of the marking of the states.

DOI: 10.1002/spe

(q, a, q') means that from state q it is possible to move to state q' with action $a \in I \cup O$. We let $\mathcal{IOTS}(I, O)$ denote the set of $\mathcal{IOTS}$s with input set I and output set O.

Any state $q \in Q$ induces an $\mathcal{IOTS}$ derived from M by setting the initial state to q, that is, abusing the notation we consider $q = (Q, I, O, T, q)$.

We say that state $q \in Q$ is *quiescent* if from q it is not possible to produce output without first receiving input. We can extend T to T_δ by adding (q, δ, q) for each quiescent state q. We let $\mathcal{A}ct = I \cup O \cup \{\delta\}$ denote the set of actions. We say that M is *input-enabled* if for all $q \in Q$ and $?i \in I$ there exists $q' \in Q$ such that $(q, ?i, q') \in T$. We say that M is *output-divergent* if it can reach a state in which there is an infinite path that contains outputs only. □

Let us note that processes and states are effectively the same since we can identify a process with its initial state and we can define a process corresponding to a state q of M by making q the initial state. Thus, in this paper we use states and processes and their notation interchangeably. As stated in the introduction, we use the normal notation in which we precede the name of an input by ? and the name of an output by !. We assume that all processes are input-enabled and are not output-divergent. The intuition behind the first restriction is that systems should be able to respond to any signal received from the environment. In fact, this restriction is usually imposed on implementations, while specifications are sometimes allowed to break this restriction. However, if we assume that all processes are input-enabled then some definitions are simplified, while this restriction does not lead to a significant reduction in the expressive power of specifications. Regarding the second restriction, 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. 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, the presence of a state from which we can take an infinite sequence of outputs is normally undesirable and is similar to a livelock. Let us note that formal testing approaches based on **ioco** assume that quiescence can be observed just as any regular output. This fact is better explained by using timed extensions of **ioco**: if an output is not observed *soon*, then we can consider that we have reached a quiescent state.

Traces are sequences of visible actions, possibly including quiescence, and are often called *suspension traces*. Since they are the only type of trace we consider, we call them *traces*. The following is standard notation in the context of **ioco**.

Definition 2
Let $M = (Q, I, O, T, q_{in})$ be an $\mathcal{IOTS}$.

1. If $(q, a, q') \in T_\delta$, for $a \in \mathcal{A}ct$, then we write $q \xrightarrow{a} q'$.
2. We write $q \stackrel{\sigma}{\Longrightarrow} q'$ for $\sigma = a_1 \dots a_m \in \mathcal{A}ct^*$ if there exist $q_0, \dots, q_m \in Q$, with $q = q_0$ and $q' = q_m$, such that for all $0 \leq i < m$ we have that $q_i \xrightarrow{a_{i+1}} q_{i+1}$.
3. We write $M \stackrel{\sigma}{\Longrightarrow}$ if there exists $q' \in Q$ such that $q_{in} \stackrel{\sigma}{\Longrightarrow} q'$ and we say that σ is a *trace* of M. We let $\mathcal{T}r(M)$ denote the set of traces of M.

Let $q \in Q$ be a state and $\sigma \in \mathcal{A}ct^*$ be a trace. We consider

1. q **after** $\sigma = \{q' \in Q | q \stackrel{\sigma}{\Longrightarrow} q'\}$
2. $out(q) = \{!o \in O \cup \{\delta\} | q \xrightarrow{!o}\}$
3. Given a set $Q' \subseteq Q$, we consider that Q' **after** $\sigma = \cup_{q \in Q'}\ q$ **after** σ and $out(Q') = \cup_{q \in Q'} out(q)$.

□

In testing from a single-port $\mathcal{IOTS}$ it is usual to use the **ioco** relation [20, 21] to establish what a *good* implementation is. Intuitively, an SUT correctly implements a specification if it does not *invent* behaviours that are not allowed by the specification. Since inputs alone cannot be used to differentiate two processes, implementation relations will usually depend on the set of outputs that the compared systems can perform after a given sequence.

Prepared using speauth.cls
DOI: 10.1002/spe

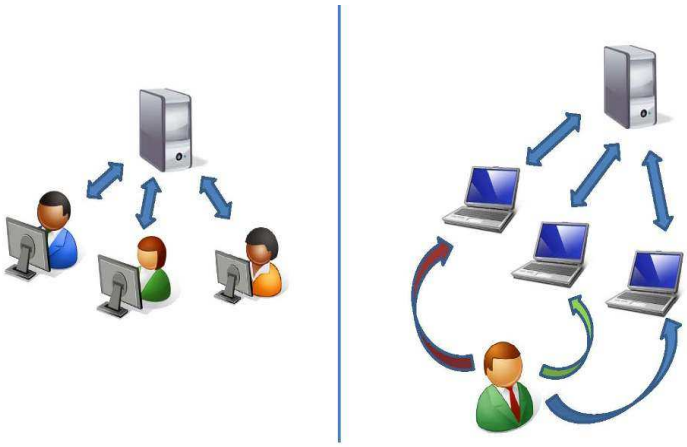


Figure 3. The local and distributed test architectures.

Definition 3
Given $M, M' \in \mathcal{IOTS}(I, O)$ we write M' **ioco** M if for every trace $\sigma \in \mathcal{T}r(M)$ we have that $out(M' \textbf{ after } \sigma) \subseteq out(M \textbf{ after } \sigma)$. □

Example 1
Let us consider the systems M_1 and M_2 (left-hand side and right-hand side of Figure 1, respectively). We have that M_2 is not a good implementation of M_1 according to **ioco**. For example, $out(M_2 \textbf{ after } ?i_L?i_U) = \{!o_L\} \not\subseteq \{!o_U\} = out(M_1 \textbf{ after } ?i_L?i_U)$. □

3. MULTI-PORT $\mathcal{IOTS}s$ WITH MARKED STATES

This section describes multi-port $\mathcal{IOTS}s$ with marked states and implementation relations **sdioco**, **sdioco**$'$ and **sdioco**$''$. It explores properties of these implementation relations and discusses the notion of refinement through changing the set of marked states.

3.1. Introduction

The two standard (ISO) test architectures are shown in Figure 3. In the local test architecture a global tester interacts with all of the ports of the SUT. In the distributed test architecture there is a local tester at each port [11]. We use the term $m\mathcal{IOTS}$ when there are multiple ports and we are considering marked states to denote scenarios; when there is only one port we use the term single-port $\mathcal{IOTS}$.

Definition 4
We will denote by $\mathcal{P}$ the set of ports. A *marked* $\mathcal{IOTS}$ ($m\mathcal{IOTS}$) is a pair $M_m = (M, \mathcal{Q})$, where $M = (Q, I, O, T, q_{in})$ is an $\mathcal{IOTS}$ and $\mathcal{Q} \subseteq Q$ is the set of marked states. We partition I into pair-wise disjoint sets I_p, for all $p \in \mathcal{P}$, containing those inputs that can be received at port p. Similarly, O is partitioned into pair-wise disjoint sets O_p, for all $p \in \mathcal{P}$, containing those outputs that can be produced at port p.

We let $m\mathcal{IOTS}(I, O)$ denote the set of $m\mathcal{IOTS}$s with input set I and output set O. □

Inputs and outputs will often be labelled in a manner that makes their port clear. For example, $?i_U$ is an input at U and $!o_L$ is an output at L. In order to avoid unnecessary definitions, we will use in the context of $m\mathcal{IOTS}$s the concepts introduced in Definitions 1 and 2 for $\mathcal{IOTS}s$. For example, if $M_m = (M, \mathcal{Q})$ then we will say that σ is a trace of M_m if σ is a trace of M.

DOI: 10.1002/spe

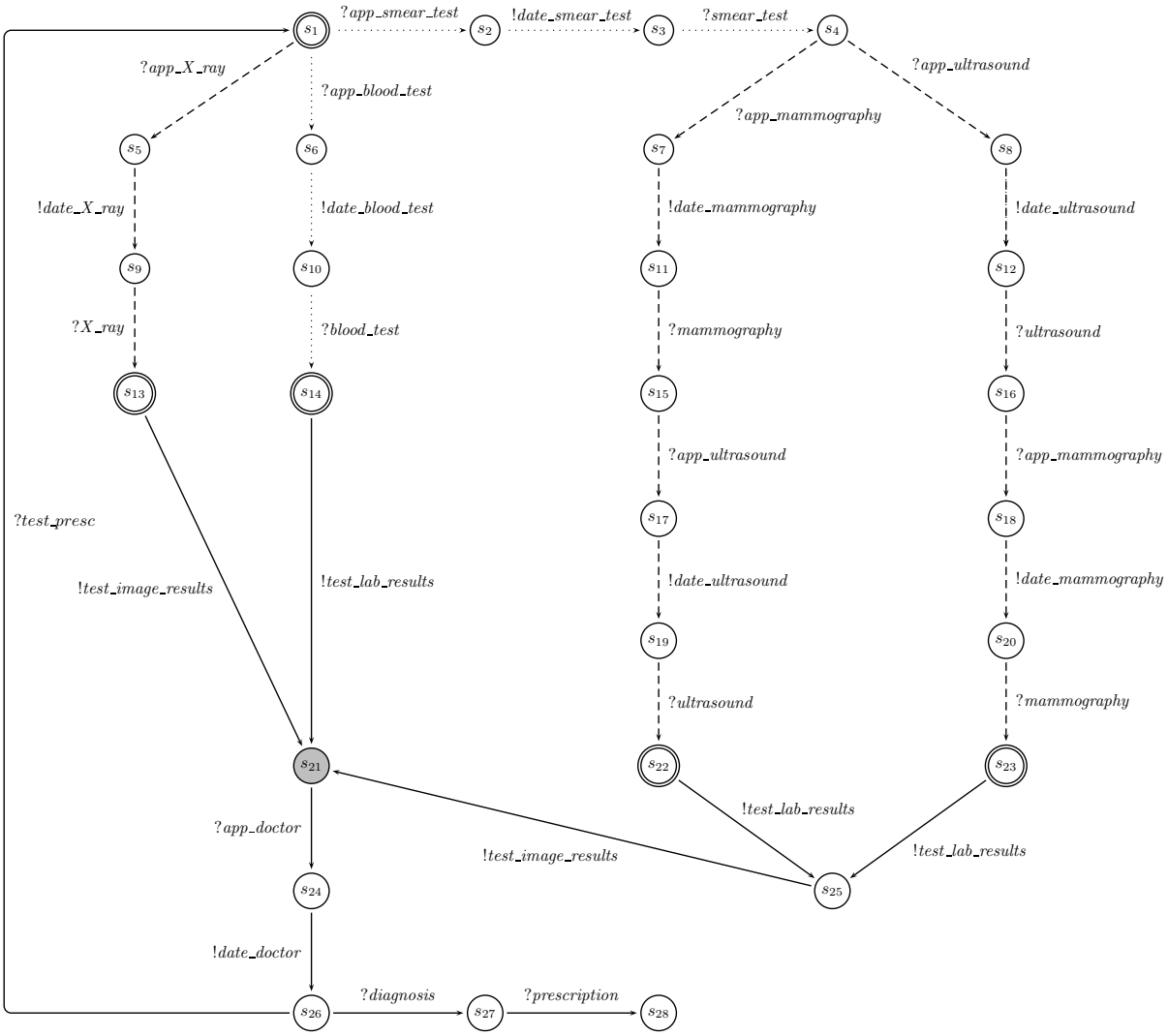


Figure 4. Specification of the appointments protocol.

Example 2

The specification depicted in Figure 4 represents a simplified version of the diagnosis protocol of a hospital management system. This protocol improves the response time to the demands of patient care because it automates the process of collecting, collating, and retrieving patient information. We focus on the functionality associated with the process that begins at the moment a patient makes a date with the doctor and receives a diagnosis. The global idea is represented in Figure 5. A patient visits the doctor that can either prescribe some tests or diagnose an illness. In the first case, the patient must go to the laboratory and/or image diagnosis section and make the corresponding appointments. Once the results of the tests are available, the patient will visit the doctor. If the results of the tests provide enough information, then the doctor will diagnose the patient and prescribe the appropriate medication. However,

DOI: 10.1002/spe

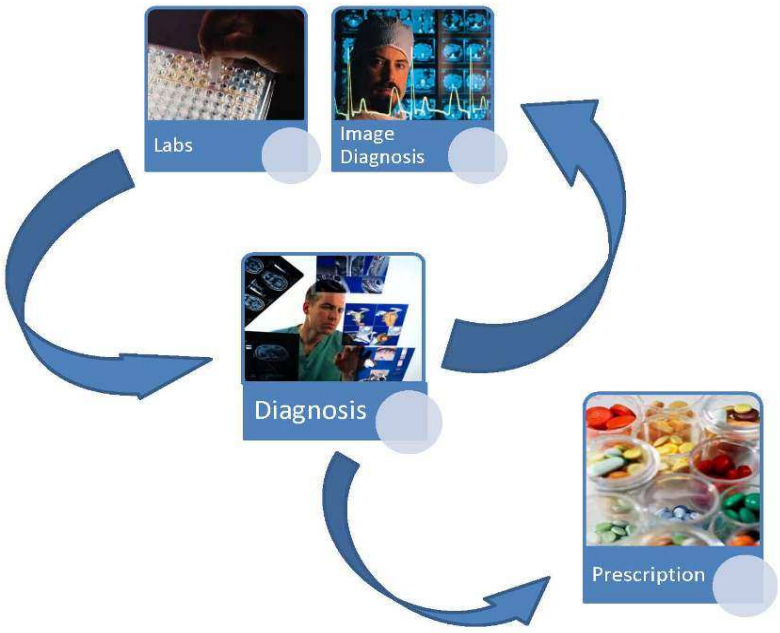


Figure 5. Overview of a hospital management system.

the doctor may need more tests to give a final diagnosis and then the patient will begin the cycle again.

The specification is in fact based on the protocol that a patient must follow in most of the hospitals of the Spanish Public Health-Care System. Thus, it is very close to a *real* system. The system presents three different ports that correspond to the laboratory, the image diagnosis section, and the surgeries. All of them are connected to the central server where the information related to each patient is stored. In order to simplify the presentation we only consider three possible batteries of tests. The first one requires only a blood test, the second one an X-ray test, and the last one consists of an ultrascan, a mammography and a smear test. The appointments for the different kind of services (laboratory tests, image tests and surgeries) are made at different offices. Once the doctor prescribes the tests, the patient must go to the corresponding office(s) for making the appointments. After the test results are received in the doctor's office and the patient makes an appointment, the patient will visit the doctor for a diagnosis.

The lines of the transitions associated to the different ports are drawn in different ways: solid for doctor's office port, dashed for image diagnosis office and dotted for laboratory office. Marked states have a double circle while the initial state s_{21} is shaded. As we previously commented, marked states restrict the possible behaviours of the system. In our example, the system is not allowed to produce and register the results of tests prescribed by the doctor for a patient, before the patient makes the corresponding appointments for each of the tests and the tests are performed. This requirement is represented by means of different scenarios. □

In order to be consistent with the **ioco** theory, we consider that specifications and implementations are defined by using the same formalism, that is, input-enabled, non output-divergent $m\mathcal{IOTS}$s. However, we will not use the set of marked states associated with

DOI: 10.1002/spe

implementations (equivalently, we can consider that it is empty). The idea is that if the implementation is treated as a black-box, we cannot know its current state. Therefore, we cannot know whether that state belongs to the set of marked ones.

A *global tester* observes all the ports and so observes a trace in $\mathcal{A}ct^*$, called a *global trace*. However, we will usually have a set of *local testers*. Therefore, we will use the *local traces* that can be obtained from a global trace.

Definition 5
Let $\sigma \in \mathcal{A}ct^*$ and $p \in \mathcal{P}$. We let $\pi_p(\sigma)$ denote the projection of σ onto p; this is called a *local trace*. The function π_p can be defined by the following rules.

1. $\pi_p(\epsilon) = \epsilon$.
2. If $z \in (I_p \cup O_p \cup \{\delta\})$ then $\pi_p(z\sigma) = z\pi_p(\sigma)$.
3. If $z \in I_q \cup O_q$, for $q \neq p$, then $\pi_p(z\sigma) = \pi_p(\sigma)$.

Given global traces $\sigma, \sigma' \in \mathcal{A}ct^*$ we write $\sigma \sim \sigma'$ if σ and σ' cannot be distinguished in the distributed test architecture. Formally, $\sigma \sim \sigma'$ if and only if for all $p \in \mathcal{P}$ we have $\pi_p(\sigma) = \pi_p(\sigma')$. □

It is trivial to prove that $\sim$ is an equivalence relation. This relation will play a crucial role in defining implementation relations for the distributed architecture: We should always compare traces up to the $\sim$ relation. Intuitively, we have $\sigma \sim \sigma'$ if the order between actions when we restrict to each of the ports is kept. For example, $?i_U!o_U?i_L \sim ?i_U?i_L!o_U$ while none of these traces is equivalent to $!o_U?i_U?i_L$. The idea is that two global traces are equivalent under $\sim$ if they look identical to all of their testers. Next we define the **dioco** implementation relation for input-enabled specification [22].

Definition 6
Let $M_m^{Spec}, M_m^{SUT} \in m\mathcal{IOTS}(I, O)$. We write M_m^{SUT} **dioco** M_m^{Spec} if for every trace σ such that $M_m^{SUT} \stackrel{\sigma\delta}{\Longrightarrow}$, there exists a trace $\sigma' \in \mathcal{T}r(M_m^{Spec})$ such that $\sigma' \sim \sigma$. □

Only traces reaching quiescent states, that is, traces ending with δ, are considered in **dioco** since these allow us to put together the local traces at a point where local testers know that the component that they are testing is stable [22].

Example 3
Let us consider the specification presented in Example 2. If we replace some of its transitions by the subgraph depicted in Figure 6 we obtain an alternative protocol. This new protocol does not conform to the original one with respect to **ioco**, since there has been a change concerning the order in which certain actions are performed. However, this modification is not relevant if we take into account that this change does not modify the original causality relations between actions at the same subsystem and, therefore, this alternative protocol does conform to the original one if we use **dioco**.

At a more abstract level, let us consider again the processes M_1 and M_2 given in the left-hand side and right-hand side of Figure 1. We have that M_2 **dioco** M_1. Even though M_2 has traces that cannot be performed by M_1, for example, $?i_L?i_U!o_L$, we overcome this problem as soon as we let M_1 simulate each (quiescent) trace of M_2 up to $\sim$. For instance, the previous *problematic* trace of M_2 can be simulated by $?i_U?i_L!o_L$, a trace of M_1, since $?i_L?i_U!o_L \sim ?i_U?i_L!o_L$. □

Let us note that we have not used marked states in the previous definition since this is a feature relevant only for our new relation. Therefore, **dioco** applies in the same way to $m\mathcal{IOTS}$ or its associated $\mathcal{IOTS}$.

3.2. An implementation relation using marked states

As we discussed in the introduction, the **dioco** relation does not capture synchronisation points since at such points we have to check that the traces that reach marked states are implemented, up to the $\sim$ relation. We can define an implementation relation for $m\mathcal{IOTS}s$ that uses marked states.

DOI: 10.1002/spe

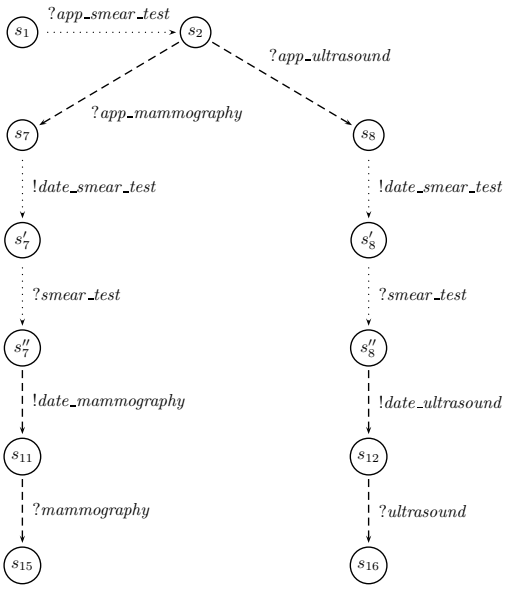


Figure 6. A variant of the appointments protocol.

Definition 7
Let $M_m^{Spec}, M_m^{SUT} \in m\mathcal{IOTS}(I,O)$, where $M_m^{Spec} = (M, \mathcal{Q})$. We write M_m^{SUT} **sdioco** M_m^{Spec} if for every trace σ such that $M_m^{SUT} \stackrel{\sigma\delta}{\Longrightarrow}$, there exists a trace $\sigma' = a_1, \ldots, a_s \in \mathcal{T}r(M_m^{Spec})$ such that the following two conditions hold:

- $\sigma' \sim \sigma$.
- There is a derivation $M_m^{Spec} \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_{s-1} \xrightarrow{a_s} q_s$ in which $J = \{j_1, \ldots, j_r\} \subseteq \{1, \ldots, s\}$ is the maximal set of indexes such that $q_{j_i} \in \mathcal{Q}$ for all $1 \leq i \leq r$ and $\sigma'_1, \ldots, \sigma'_{r+1}$ are the sequences such that $\sigma' = \sigma'_1 \cdots \sigma'_{r+1}$ and $M_m^{Spec} \stackrel{\sigma'_1}{\Longrightarrow} q_{j_1} \stackrel{\sigma'_2}{\Longrightarrow} q_{j_2} \cdots q_{j_r} \stackrel{\sigma'_{r+1}}{\Longrightarrow} q_s$. In addition, $\sigma = \sigma_1 \ldots \sigma_{r+1}$ for some sequences $\sigma_1, \ldots, \sigma_{r+1}$ such that for all $1 \leq j \leq r+1$ we have that $\sigma'_j \sim \sigma_j$.

□

In the previous definition, if the initial state of the specification is marked we assume that an additional index j_0 is added to J so that q_{j_0} corresponds to the first occurrence of the initial state, so that we have a derivation such as $q_{j_0} \stackrel{\sigma'_1}{\Longrightarrow} q_1 \stackrel{\sigma'_2}{\Longrightarrow} q_2 \cdots$. Let us note that J is the set of indexes corresponding to the marked states of the derivation. Therefore, it may happen that there exist several indexes corresponding to the same state of the specification.

Intuitively, we have that M_m^{SUT} is a good implementation of M_m^{Spec} under the **sdioco** relation if in addition to not inventing any behaviours (first condition, similar to **dioco**) we have that marked states that can be traversed in the specification while performing the analysed trace are respected in the implementation. In other words, the second condition ensures that all of the subtraces that M_m^{Spec} performs to complete the whole trace can also be performed, up to $\sim$, by M_m^{SUT}. It is sufficient for this condition to hold for one possible way in which M_m^{Spec} can perform the trace while, due to possible nondeterminism, there may be several possible ways in which M_m^{Spec} can perform the trace. Another possibility would be to consider that

DOI: 10.1002/spe

the specifier has defined a set of behaviours, that include markings, and wants *all* of them to be implemented. In this case, the *there exists* path quantification should be replaced by a *for all* path statement, and this would lead to another implementation relation. In Section 3.3 we explore two such ways of strengthening **sdioco**.

Example 4
We can use our running example to show that the implementation relation **sdioco** is stronger than **dioco**. For example, let us consider an implementation where the original transitions $s_5 \xrightarrow{!date\_X\_ray} s_9$ and $s_{13} \xrightarrow{!test\_image\_results} s_{21}$ are substituted by the transitions $s_5 \xrightarrow{!test\_image\_results} s_9$ and $s_{13} \xrightarrow{!date\_X\_ray} s_{21}$. This implementation does not conform to the specification under **sdioco**. That it is due to the fact that the marked state requires the transition labelled by $!date\_X\_ray$ to be performed before the transition labelled by $!test\_image\_results$. In contrast, if we consider the **dioco** conformance relation, then this implementation conforms to the specification because the exchange affects transitions in different ports, that is, it does not modify the order in which the actions are performed in the ports. However, this implementation would allow the system to produce the results of an X-ray test even before the patient has received the date to get the test. This example clearly shows that even though **dioco** has its merit as an implementation relation for distributed systems, it also has drawbacks if we need to consider situations where the order of the actions performed at different ports is relevant. □

The next result indicates that our new relation **sdioco** is an appropriate extension of previous relations. Specifically, if we consider that none of the states is marked we have **dioco** while if all the states are marked then we have **ioco**. This result represents a good *sanity check* to increase our confidence regarding the suitability of **sdioco** as a good implementation relation for distributed systems.

Proposition 1
Let $M_m^{Spec}, M_m^{SUT} \in m\mathcal{IOTS}(I,O)$, where $M_m^{Spec} = (M, \mathcal{Q})$ and $M = (Q, I, O, T, q_{in})$. Then,

- If $\mathcal{Q} = Q$ then M_m^{SUT} **ioco** M_m^{Spec} if and only if M_m^{SUT} **sdioco** M_m^{Spec}.
- If $\mathcal{Q} = \emptyset$ then M_m^{SUT} **dioco** M_m^{Spec} if and only if M_m^{SUT} **sdioco** M_m^{Spec}.

Proof
We start by assuming that $\mathcal{Q} = Q$ and prove that M_m^{SUT} **ioco** M_m^{Spec} if and only if M_m^{SUT} **sdioco** M_m^{Spec}.

First, we assume that M_m^{SUT} **ioco** M_m^{Spec} and prove that M_m^{SUT} **sdioco** M_m^{Spec}, but this follows immediately by noting that since M_m^{SUT} **ioco** M_m^{Spec} and M_m^{Spec} and M_m^{SUT} are input enabled we trivially have that every trace of M_m^{SUT} is also a trace of M_m^{Spec}.

Now, let us assume that M_m^{SUT} **sdioco** M_m^{Spec} and that σ is a trace of M_m^{Spec} and so we have to prove that $out(M_m^{SUT}\textbf{after } \sigma) \subseteq out(M_m^{Spec}\textbf{after } \sigma)$. Let us suppose that $a \in out(M_m^{SUT}\textbf{after } \sigma)$ and so that $M_m^{SUT} \stackrel{\sigma a}{\Longrightarrow}$. Since M_m^{SUT} **sdioco** M_m^{Spec} we must have some $\sigma' \sim \sigma a = a_1, \ldots, a_r$ such that $M_m^{Spec} \stackrel{\sigma'}{\Longrightarrow}$. In addition, since all states of M_m^{Spec} are marked, there exist sequences $\sigma'_1, \ldots, \sigma'_r$ such that $M_m^{Spec} \stackrel{\sigma'_1 \ldots \sigma'_r}{\Longrightarrow}$ and for all $1 \leq j \leq r$ we have that $\sigma'_j \sim a_j$. Therefore, for all $1 \leq j \leq r$ we have $\sigma'_j = a_j$. Thus, $M_m^{Spec} \stackrel{\sigma a}{\Longrightarrow}$ and so $a \in out(M_m^{Spec}\textbf{after } \sigma)$ as required.

The second part, which is that if $\mathcal{Q} = \emptyset$ then M_m^{SUT} **dioco** M_m^{Spec} if and only if M_m^{SUT} **sdioco** M_m^{Spec}, follows from the definitions of **dioco** and **sdioco**. □

Intuitively, quiescent states are checked in the definition of **dioco** since a quiescent state of the SUT has to be *simulated* by a quiescent state of the specification; otherwise, the SUT would be able to perform the δ output action while the specification could not. It may thus appear that if we only mark quiescent states we simply obtain **dioco** but this is not the case.

Example 5
Let us consider Figure 7 with an implementation M_m^{SUT}, given in the left-hand side, and a

DOI: 10.1002/spe

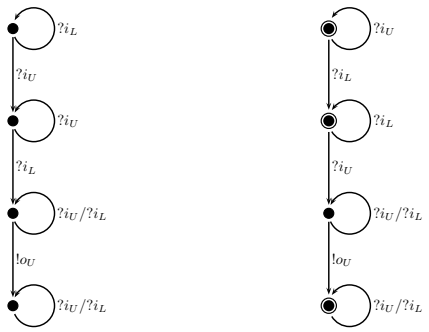


Figure 7. Quiescence alone does not capture marked states.

specification M_m^{Spec}, given in the right-hand side. The marked states of M_m^{Spec} coincide with its quiescent states. We obviously have M_m^{SUT} **dioco** M_m^{Spec}. If we consider the trace $\sigma = ?i_U?i_L!o_U$ of M_m^{SUT} we have that this trace corresponds, up to $\sim$, only to the trace $\sigma' = ?i_L?i_U!o_U$ of M_m^{Spec}. Since the state reached in M_m^{Spec} after performing $?i_L$ is marked, we have to decompose σ in such a way that $\sigma_1 \sim ?i_L$, $\sigma_2 \sim ?i_U!o_U$ and $\sigma = \sigma_1\sigma_2$. Since this is not possible, we do not have that M_m^{SUT} **sdioco** M_m^{Spec}. □

3.3. Alternative stronger implementation relations

Under **sdioco**, for each quiescent trace σ in the implementation there must be an equivalent trace σ' in the specification such that there is a derivation of the specification with trace σ' that is consistent with σ' given the synchronisation points. There may be several possible derivations of the specification with trace σ and we could instead say that each corresponds to a possible set of synchronisations that might occur and we require that σ is consistent with σ' for all of these. This results in the following implementation relation.

Definition 8
Let $M_m^{Spec}, M_m^{SUT} \in m\mathcal{IOTS}(I,O)$, where $M_m^{Spec} = (M, \mathcal{Q})$. We write M_m^{SUT} **sdioco**$'$ M_m^{Spec} if for every trace σ such that $M_m^{SUT} \stackrel{\sigma\delta}{\Longrightarrow}$, there exists a trace $\sigma' = a_1, \ldots, a_s \in \mathcal{T}r(M_m^{Spec})$ such that the following two conditions hold:

- $\sigma' \sim \sigma$.
- For every derivation $M_m^{Spec} \stackrel{a_1}{\longrightarrow} q_1 \stackrel{a_2}{\longrightarrow} q_2 \cdots q_{s-1} \stackrel{a_s}{\longrightarrow} q_s$ in which $J = \{j_1, \ldots, j_r\} \subseteq \{1, \ldots, s\}$ is the maximal set of indexes such that $q_{j_i} \in \mathcal{Q}$ for all $1 \leq i \leq r$ and $\sigma'_1, \ldots, \sigma'_{r+1}$ are the sequences such that $\sigma' = \sigma'_1 \cdots \sigma'_{r+1}$ and $M_m^{Spec} \stackrel{\sigma'_1}{\Longrightarrow} q_{j_1} \stackrel{\sigma'_2}{\Longrightarrow} q_{j_2} \cdots q_{j_r} \stackrel{\sigma'_{r+1}}{\Longrightarrow} q_s$ we have that $\sigma = \sigma_1 \ldots \sigma_{r+1}$ for some sequences $\sigma_1, \ldots, \sigma_{r+1}$ such that for all $1 \leq j \leq r+1$ we have that $\sigma'_j \sim \sigma_j$.

□

Proposition 2
Let $M_m^{Spec}, M_m^{SUT} \in m\mathcal{IOTS}(I,O)$, where $M_m^{Spec} = (M, \mathcal{Q})$ and $M = (Q, I, O, T, q_{in})$. If M_m^{SUT} **sdioco**$'$ M_m^{Spec} then we must have that M_m^{SUT} **sdioco** M_m^{Spec}. However, we might have that M_m^{SUT} **sdioco** M_m^{Spec} but that M_m^{SUT} **sdioco**$'$ M_m^{Spec} does not hold.
Proof
The first part follows immediately from the definitions since the only difference is that under **sdioco**$'$ we require all possible derivations with a trace σ' to satisfy the required condition while under **sdioco** it is sufficient for one such derivation to satisfy this condition.

For the second part, let us consider the processes in Figure 8, in which the implementation M_m^{SUT} is on the left-hand side and the specification M_m^{Spec} is on the right-hand side. Let

DOI: 10.1002/spe

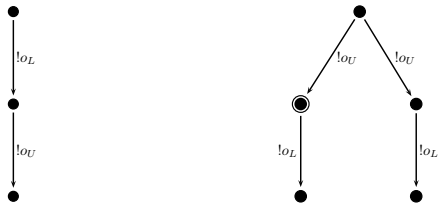


Figure 8. Processes that are not related under **sdioco**$'$ but are related under **sdioco**.

us also consider the trace $\sigma = !o_L!o_U$ of the implementation. This is allowed by M_m^{Spec} under **sdioco** since it has a trace $\sigma' = !o_U!o_L$ such that $\sigma' \sim \sigma$ and a derivation with σ' that includes no marked states. However, under **sdioco**$'$ we need to consider both possible derivations of M_m^{Spec} with trace σ' and one has a marked state after $!o_U$. We can therefore see that M_m^{SUT} does not conform to M_m^{Spec} under **sdioco**$'$ as required. □

This example illustrates an interesting point. We can see the specification as being of the form $s_1 + s_2$ in which s_1 is the left-hand branch of the specification and so can do $!o_U!o_L$ but has the state after $!o_U$ marked, while s_2 is the right-hand branch of the specification and so can do $!o_U!o_L$ and has no marked states. Clearly, under **sdioco** and **sdioco**$'$ we have that if an implementation conforms to s_1 then it must conform to s_2 but the converse is not the case. In this case, an implementation conforms to $s_1 + s_2$ under **sdioco** if and only if it conforms to s_2 under **sdioco**. In addition, an implementation conforms to $s_1 + s_2$ under **sdioco**$'$ if and only if it conforms to s_1 under **sdioco**$'$.

Naturally, we can strengthen **sdioco**$'$ further by using universal quantification over the $\sigma' \sim \sigma$ that are traces of the specification. This leads to the following implementation relation.

Definition 9
Let $M_m^{Spec}, M_m^{SUT} \in m\mathcal{IOTS}(I,O)$, where $M_m^{Spec} = (M, \mathcal{Q})$. We write M_m^{SUT} **sdioco**$''$ M_m^{Spec} if for every trace σ such that $M_m^{SUT} \stackrel{\sigma\delta}{\Longrightarrow}$, there exists a trace $\sigma' \in \mathcal{T}r(M_m^{Spec})$ such that $\sigma' \sim \sigma$ and for all $\sigma' \sim \sigma$ such that $\sigma' = a_1, \ldots, a_s$ is a trace of M_m^{Spec}, the following condition holds:

- For every derivation $M_m^{Spec} \stackrel{a_1}{\longrightarrow} q_1 \stackrel{a_2}{\longrightarrow} q_2 \cdots q_{s-1} \stackrel{a_s}{\longrightarrow} q_s$ in which $J = \{j_1, \ldots, j_r\} \subseteq \{1, \ldots, s\}$ is the maximal set of indexes such that $q_{j_i} \in \mathcal{Q}$ for all $1 \leq i \leq r$ and $\sigma'_1, \ldots, \sigma'_{r+1}$ are the sequences such that $\sigma' = \sigma'_1 \cdots \sigma'_{r+1}$ and $M_m^{Spec} \stackrel{\sigma'_1}{\Longrightarrow} q_{j_1} \stackrel{\sigma'_2}{\Longrightarrow} q_{j_2} \cdots q_{j_r} \stackrel{\sigma'_{r+1}}{\Longrightarrow} q_s$ we have that $\sigma = \sigma_1 \ldots \sigma_{r+1}$ for some sequences $\sigma_1, \ldots, \sigma_{r+1}$ such that for all $1 \leq j \leq r+1$ we have that $\sigma'_j \sim \sigma_j$.

□

Now let us consider the specification given in Figure 9 and the trace $\sigma = !o_L!o_U$ of this specification. Then, the specification also contains the trace $\sigma' = !o_U!o_L$ where the state after $!o_U$ is marked. If we consider σ and σ' in Definition 9 we see that the specification does not conform to itself under **sdioco**$''$. Thus, **sdioco**$''$ is not a suitable implementation relation.

3.4. Refinement through changing marked states

The set of marked states defines the ability of the environment to make additional observations through synchronising. Thus, we change the observational power when we change the set of marked states. This gives us a simple notion of refinement through changing the set of marked states. The following results are clear from the definitions of **sdioco** and **sdioco**$'$.

Proposition 3
Let $M_m^{Spec}, M_m^{SUT} \in m\mathcal{IOTS}(I,O)$, where $M_m^{Spec} = (M, \mathcal{Q})$. If M_m^{SUT} **sdioco** M_m^{Spec} and $\mathcal{Q}' \subseteq \mathcal{Q}$ then we have that M_m^{SUT} **sdioco** $(M, \mathcal{Q}')$. □

DOI: 10.1002/spe

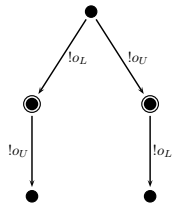


Figure 9. A specification that shows the unsuitability of **sdioco**$''$.

Proposition 4
Let $M_m^{Spec}, M_m^{SUT} \in m\mathcal{IOTS}(I,O)$, where $M_m^{Spec} = (M, \mathcal{Q})$. If M_m^{SUT} **sdioco**$'$ M_m^{Spec} and $\mathcal{Q}' \subseteq \mathcal{Q}$ then we have that M_m^{SUT} **sdioco**$'$ $(M, \mathcal{Q}')$. □

These results concern what happens when we reduce the number of marked states. At times we will want to increase the set of marked states. We now investigate the situation in which an implementation conforms to both $(M, \mathcal{Q}_1)$ and $(M, \mathcal{Q}_2)$, under **sdioco** or **sdioco**$'$ for some $\mathcal{Q}_1 \neq \mathcal{Q}_2$. The following is clear from the definitions of **sdioco** and **sdioco**$'$ and allows us to increase the set of marked states in some circumstances.

Proposition 5
Let $M_m^{SUT} \in m\mathcal{IOTS}(I,O)$ be an implementation and $(M, \mathcal{Q}_1), (M, \mathcal{Q}_2) \in m\mathcal{IOTS}(I,O)$ be specifications. Further, let us suppose that for every trace $\sigma \in (I \cup O \cup \{\delta\})^*$ we have that at least one of the following hold:

1. There does not exist $\sigma' \sim \sigma$ such that M has a derivation with trace σ' that includes a state from Q_1.
2. There does not exist $\sigma' \sim \sigma$ such that M has a derivation with trace σ' that includes a state from Q_2.

If $\mathcal{Q}_3 = \mathcal{Q}_1 \cup \mathcal{Q}_2$ then we have the following results.

1. If M_m^{SUT} **sdioco** $(M, \mathcal{Q}_1)$ and M_m^{SUT} **sdioco** $(M, \mathcal{Q}_2)$ then M_m^{SUT} **sdioco** $(M, \mathcal{Q}_3)$.
2. If M_m^{SUT} **sdioco**$'$ $(M, \mathcal{Q}_1)$ and M_m^{SUT} **sdioco**$'$ $(M, \mathcal{Q}_2)$ then M_m^{SUT} **sdioco**$'$ $(M, \mathcal{Q}_3)$.

□

Naturally, there will be other conditions under which we can extend the set of marked states but this is a topic of future work. Let us note that if we do not place restrictions on $\mathcal{Q}_1$ and $\mathcal{Q}_2$ then this result need not hold.

Proposition 6
Let $M_m^{SUT} \in m\mathcal{IOTS}(I,O)$ be an implementation and $(M, \mathcal{Q}_1), (M, \mathcal{Q}_2) \in m\mathcal{IOTS}(I,O)$ be specifications. If $\mathcal{Q}_3 = \mathcal{Q}_1 \cup \mathcal{Q}_2$ then we have the following results.

1. It is possible that M_m^{SUT} **sdioco** $(M, \mathcal{Q}_1)$ and M_m^{SUT} **sdioco** $(M, \mathcal{Q}_2)$ but that we do not have that M_m^{SUT} **sdioco** $(M, \mathcal{Q}_3)$.
2. It is possible that M_m^{SUT} **sdioco**$'$ $(M, \mathcal{Q}_1)$ and M_m^{SUT} **sdioco**$'$ $(M, \mathcal{Q}_2)$ but that we do not have that M_m^{SUT} **sdioco**$'$ $(M, \mathcal{Q}_3)$.
3. It is possible that M_m^{SUT} **sdioco**$'$ $(M, \mathcal{Q}_1)$ and M_m^{SUT} **sdioco**$'$ $(M, \mathcal{Q}_2)$ but that we do not have that M_m^{SUT} **sdioco** $(M, \mathcal{Q}_3)$.

Proof
For all the parts we can consider the implementation and two specifications s_1 and s_2 given in Figure 10. Let us consider the trace $\sigma = !o_1!o_2!o_3$ of the implementation. We have that this is acceptable for s_1 by examining the path with trace $\sigma_1 = !o_1!o_3!o_2$ and no marked states. We have that this is acceptable for s_2 by examining the path with trace $\sigma_2 = !o_2!o_1!o_3$

DOI: 10.1002/spe

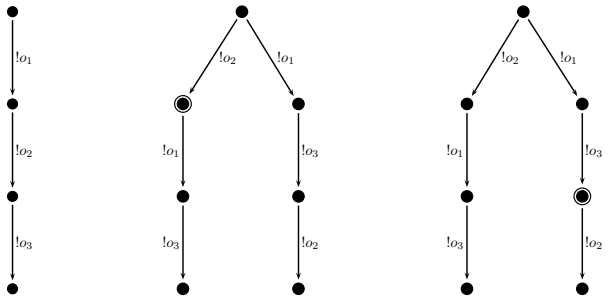


Figure 10. An implementation (left) and two specifications s_1 (center) and s_2 (right).

and no marked states. Further, we only have one derivation with trace σ_1 and only one derivation with trace σ_2. It is therefore clear that we have that the implementation conforms to both s_1 and s_2 under **sdioco** and **sdioco**$'$. However, if we take the union of the sets of marked states then we find that σ_1 and σ_2 pass through marked states and this results in the implementation failing to conform under **sdioco** and also under **sdioco**$'$. The result therefore follows. □

The proof of this result reveals an interesting feature of **sdioco**$'$: it is not compositional under choice. Specifically, we have that M_m^{SUT} **sdioco**$'$ s_1 and M_m^{SUT} **sdioco**$'$ s_2 but we do not have that M_m^{SUT} **sdioco**$'$ $s_1 + s_2$, where $s_1 + s_2$ denotes the process that can choose to behave like either s_1 or s_2. This might appear to be counter-intuitive: typically, if we add behaviours to the specification then we can allow additional implementations to conform to the new specification but do not eliminate conforming specifications. What is happening here is that the marking of states does not represent behaviours of the specification but the opportunity for the environment to make additional observations regarding the behaviour of the implementation. By adding such opportunities we can make it harder for an implementation to conform to the specification.

4. AN ALTERNATIVE CHARACTERISATION OF SCENARIOS

A scenario is any sequence of events that takes the specification to one of its marked states. More precisely, scenarios are associated with sequences that bring the specification from one marked state to another one without traversing any marked states. Therefore, it should be possible to alternatively define our new relation by associating a set of traces with a specification. In this section we give an alternative characterisation of scenarios in the distributed architecture based on a set of traces.

4.1. Multi-port $\mathcal{IOTS}$s with scenarios based on traces

We use the term *t$\mathcal{IOTS}$* when we are describing systems with multiple ports and we are considering a set of traces to denote scenarios. Intuitively, the set of traces associated with a machine represents the behaviours allowed when scenarios are considered.

Definition 10
A *traced* $\mathcal{IOTS}$ (*t$\mathcal{IOTS}$*) is a pair $M_m = (M, \mathcal{T})$, where $M = (Q, I, O, T, q_{in})$ is an $\mathcal{IOTS}$ and $\mathcal{T} \subseteq (I \cup O)^*$ is a set of traces. We partition I into pair-wise disjoint sets I_p, for all $p \in \mathcal{P}$, containing those inputs that can be received at port p. Similarly, O is partitioned into pair-wise disjoint sets O_p, for all $p \in \mathcal{P}$, containing those outputs that can be produced at port p.

We let *t$\mathcal{IOTS}$*(I, O) denote the set of *t$\mathcal{IOTS}$*s with input set I and output set O. □

DOI: 10.1002/spe

In the previous definition, the set $\mathcal{T}$ will be used to represent the traces after which the users or tests can synchronise. We will use in the context of $t\mathcal{IOTS}$s the concepts introduced in Definitions 1 and 2 for $\mathcal{IOTS}s$. In addition, we will adopt the same hypotheses over specifications and implementations, that is, they are defined using the same formalism, they are input-enabled and non output-divergent $t\mathcal{IOTS}$s, and the set of traces is empty for implementations.

As we discussed in the introduction, the **dioco** relation does not capture synchronisation points, or alternatively traces defining scenarios, since at such points we have to check that the traces that reach marked states are implemented, up to the $\sim$ relation. Therefore, a new implementation relation among $m\mathcal{IOTS}s$, that we called **sdioco**, was introduced in the previous section. Next, we define a similar implementation relation among $t\mathcal{IOTS}s$.

Definition 11
Let $M_t^{Spec}, M_t^{SUT} \in t\mathcal{IOTS}(I,O)$, where $M_t^{Spec} = (M, \mathcal{T})$. We write M_t^{SUT} **tsdioco** M_t^{Spec} if for every trace σ such that $M_t^{SUT} \stackrel{\sigma\delta}{\Longrightarrow}$, there exists a trace $\sigma' \in \mathcal{T}r(M_t^{Spec})$ such that $\sigma' \sim \sigma$ and $\sigma \in \mathcal{T}$. □

In addition to the first condition that establishes that the implementation cannot invent any behaviours, the second condition in the definition of **tsdioco** ensures that the trace performed in the implementation is accepted by the specification. There exist several ways in which a trace σ' that can be performed by M_m^{Spec} is allowed to be produced by an implementation. Those behaviours that are accepted are collected in the set of traces $\mathcal{T}$.

The next result indicates that our new relation is an appropriate extension of previous relations. Specifically, if we consider that only the traces of the specification are included in $\mathcal{T}$ then we have **ioco**, while if the set $\mathcal{T}$ contains all the traces of the specification that reach a quiescent state and all the traces related to them up to $\sim$, we have **dioco**. The proof of the result is very similar to the proof of Proposition 1 and therefore we omit it.

Proposition 7
Let $M_t^{Spec}, M_t^{SUT} \in m\mathcal{IOTS}(I,O)$, where $M_t^{Spec} = (M, \mathcal{T})$ and $M = (Q, I, O, T, q_{in})$. Then,

- If $\mathcal{T} = \mathcal{T}r(M_m^{Spec})$ then M_t^{SUT} **ioco** M_t^{Spec} if and only if M_t^{SUT} **tsdioco** M_t^{Spec}.
- If $\mathcal{T} = \{\sigma' | \exists \sigma \in (I \cup O)^* : \sigma \sim \sigma' \wedge M_t^{Spec} \stackrel{\sigma\delta}{\Longrightarrow}\}$ then M_t^{SUT} **dioco** M_t^{Spec} if and only if M_t^{SUT} **tsdioco** M_t^{Spec}.

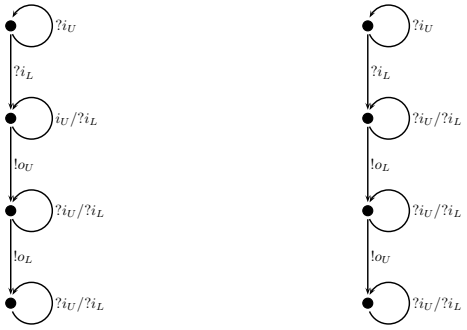
□

Example 6
Let us consider the two systems depicted in Figure 11: M_5 is given on the left-hand side while M_6 is given on the right-hand side. We have M_6 **dioco** M_5. Depending on the set of traces associated to M_5 we can have the conformance of M_6 with respect to **tsdioco** or not. If we consider that the associated set contains all the possible traces, that is, the set $\mathcal{T}_1$, then we know that **tsdioco** and **dioco** coincide and therefore we have $(M_6, \emptyset)$ **tsdioco**$(M_5, \mathcal{T}_1)$. However, if we consider the set $\mathcal{T}_2$ then we do not have the conformance of $(M_6, \emptyset)$ with respect to $(M_5, \mathcal{T}_2)$ since, for example, the trace $?i_L!o_L!o_U$ does not belong to $\mathcal{T}_2$. □

4.2. Transformation from $m\mathcal{IOTS}s$ to $t\mathcal{IOTS}s$

In this section we present an algorithm to transform an $m\mathcal{IOTS}$ into an *equivalent* $tIOTS$. Intuitively, the algorithm considers all the derivations in the $m\mathcal{IOTS}$ that reach a quiescent state. Then, the trace associated to each derivation is divided into subtraces taking into account the marked states in the derivation and all the possible traces that can be obtained by the concatenation of indistinguishable subtraces, up to the $\sim$ relation, are produced. Algorithm 1 is shown in Figure 12. We will denote by $trans(M_m)$ the $tIOTS$ obtained from the application of the algorithm to an $mIOTS$ M_m.

DOI: 10.1002/spe



$\mathcal{R}_1 = ?i_U\mathcal{R}_1|\epsilon$
$\mathcal{R}_2 = ?i_L R_{21}$
$R_{21} = ?i_U R_{21}|?i_L R_{21}|!o_U R_{22}$
$R_{22} = ?i_U R_{22}|?i_L R_{22}|\epsilon$
$\mathcal{R}_3 = !o_L R_{31}$
$R_{31} = ?i_U R_{31}|?i_L R_{31}|\epsilon$

$\mathcal{T}_1 = (I \cup O)^*$
$\mathcal{T}_2 = \{\sigma'_1\sigma'_2\sigma'_3|\exists\sigma_i \in \mathcal{R}_i : \sigma_i \sim \sigma'_i\}$

Figure 11. $(M_5, \mathcal{T}_1)$ and M_6 are related under **tsdioco** but $(M_5, \mathcal{T}_2)$ and M_6 are not related.

Input: An $m\mathcal{IOTS}$ $M_m = (M, \mathcal{Q})$.
Output: A $t\mathcal{IOTS}$ $M_t = (M, \mathcal{T})$.

{`Initialisation`}
Let Tr_δ be the set of derivations of M_m ending in quiescence and without occurrences of δ;
{`Main loop`}
For each derivation $M_m \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_{s-1} \xrightarrow{a_s} q_s \in Tr_\delta$ do

- $\sigma := a_1$;
- For $j := 1$ to $s-1$ do
  If $q_j \in \mathcal{Q}$ then $tr_j^\sim := \{\sigma'|\sigma' \sim \sigma\}$; $\sigma := a_{j+1}$
  else $tr_j^\sim := \{\epsilon\}$; $\sigma := \sigma \cdot a_{j+1}$;
- $tr_s^\sim := \{\sigma'|\sigma' \sim \sigma\}$;

$\mathcal{T} := tr_1^\sim \cdot tr_2^\sim \ldots tr_s^\sim$;
output $(M, \mathcal{T})$ and terminate.

Figure 12. Algorithm 1: translation from $m\mathcal{IOTS}$s to $t\mathcal{IOTS}$s.

Theorem 1
Let $M_m = (M, \mathcal{Q}) \in m\mathcal{IOTS}(I, O)$ and $I = (M_I, \emptyset)$ be an implementation. Then, I **sdioco** M_m if and only if I **tsdioco** $trans(M_m)$.
Proof
First, we assume that I **sdioco** M_m and prove that I **tsdioco** $trans(M_m)$. Therefore, we need to prove that for every trace σ such that $I \stackrel{\sigma\delta}{\Longrightarrow}$, there exists a trace $\sigma' \in \mathcal{T}r(M)$ such that $\sigma' \sim \sigma$ and $\sigma \in \mathcal{T}$. The first required condition follows immediately from the definitions of **sdioco** and **tsdioco**. Now, let us assume that σ is a trace that can be performed by I and so, we have to prove that $\sigma \in \mathcal{T}$. Since I **sdioco** M_m we have that there exists a

DOI: 10.1002/spe

derivation $M \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_{s-1} \xrightarrow{a_s} q_s$ in which $J = \{j_1, \ldots, j_r\} \subseteq \{1, \ldots, s\}$ is the set of indexes such that $q_{j_i} \in \mathcal{Q}$ for all $1 \leq i \leq r$ and $\sigma'_1, \ldots, \sigma'_{r+1}$ are the sequences such that $\sigma' = \sigma'_1 \cdots \sigma'_{r+1}$ and $M \stackrel{\sigma'_1}{\Longrightarrow} q_{j_1} \stackrel{\sigma'_2}{\Longrightarrow} q_{j_2} \cdots q_{j_r} \stackrel{\sigma'_{r+1}}{\Longrightarrow} q_s$. By applying the algorithm given in Figure 12 we generate, for all $1 \leq i \leq r$, the sets $tr^{\sim}_{j_i} = \{\sigma''_i | \sigma''_i \sim \sigma'_i\}$ and the additional set $tr^{\sim}_s = \{\sigma''_{r+1} | \sigma''_{r+1} \sim \sigma'_{r+1}\}$. In addition, we have that $\sigma = \sigma_1 \ldots \sigma_{r+1}$ for some sequences $\sigma_1, \ldots, \sigma_{r+1}$ such that for all $1 \leq j \leq r+1$ we have that $\sigma'_j \sim \sigma_j$. Therefore, by construction, we have that $\sigma_i \in tr^{\sim}_{j_i}$ for all $1 \leq j \leq r$ and $\sigma_{r+1} \in tr^{\sim}_s$ due to the fact that $\mathcal{T}$ is obtained by the concatenation of the traces in these sets, in particular, we have that $\sigma = \sigma_1 \ldots \sigma_{r+1}$ is a trace in $\mathcal{T}$. Therefore, I **tsdioco** $trans(M_m)$.

Now, let us assume that I **tsdioco** $trans(M_m)$ and we will prove that I **sdioco** M_m. Let us consider a trace σ such that $I \stackrel{\sigma\delta}{\Longrightarrow}$ and $\sigma \in \mathcal{T}$. Therefore, we have that $\sigma = \sigma_1 \ldots \sigma_r$ for some sequences $\sigma_1, \ldots, \sigma_r$ such that for some derivation $M \stackrel{\sigma'_1}{\Longrightarrow} q_1 \stackrel{\sigma'_2}{\Longrightarrow} q_2 \cdots q_{r-1} \stackrel{\sigma'_r}{\Longrightarrow} q_r$ where $q_j \in \mathcal{Q}$ for all $1 \leq j < r$ and $\sigma'_j \sim \sigma_j$ for all $1 \leq j \leq r$. Therefore, I **sdioco** M_m. □

5. DEFINITION AND APPLICATION OF TEST CASES: GLOBAL VS. LOCAL

A test case is a process with a finite number of states that interacts with the SUT and it usually corresponds to a test objective: it may be intended to examine some part of the behaviour of the SUT. When designing test cases it is thus simpler to consider *global test cases*, that is, test cases that can interact with all of the ports of the system. However, in the distributed test architecture we do not have a global tester that can apply a global test case: instead we place a *local tester* at each port. The local tester at port p only observes the behaviour at p and can only send input to the SUT at p. Therefore, a *local test case* is a collection of local testers, one at each port. The idea is that we will have a global test case that we will use to produce a local test case, so that each of its components can be applied by a local tester. Therefore, a global test case is an $m\mathcal{IOTS}$ that has the same input and output sets as its associated specification; a local test case is a tuple containing a test case for each of the available ports and has the inputs and outputs sets corresponding to its port.

In this section we consider only specifications with marked states. The adaption of the concepts introduced in this section to the formalism introduced in Section 4 is straightforward and we therefore omit it.

Definition 12
Let $M_m \in m\mathcal{IOTS}(I, O)$ and $\mathcal{P} = \{1, \ldots, n\}$ be the set of ports. A *global test case* t for M_m is a process from $m\mathcal{IOTS}(I, O \cup \{\delta\})$. A *local test case* for M_m is a tuple $t_l = (t_1, \ldots, t_n)$ such that for all $p \in \mathcal{P}$ we have that $t_p \in m\mathcal{IOTS}(I_p, O_p \cup \{\delta\})$. Each of the components of a local test case is called a *local tester*.

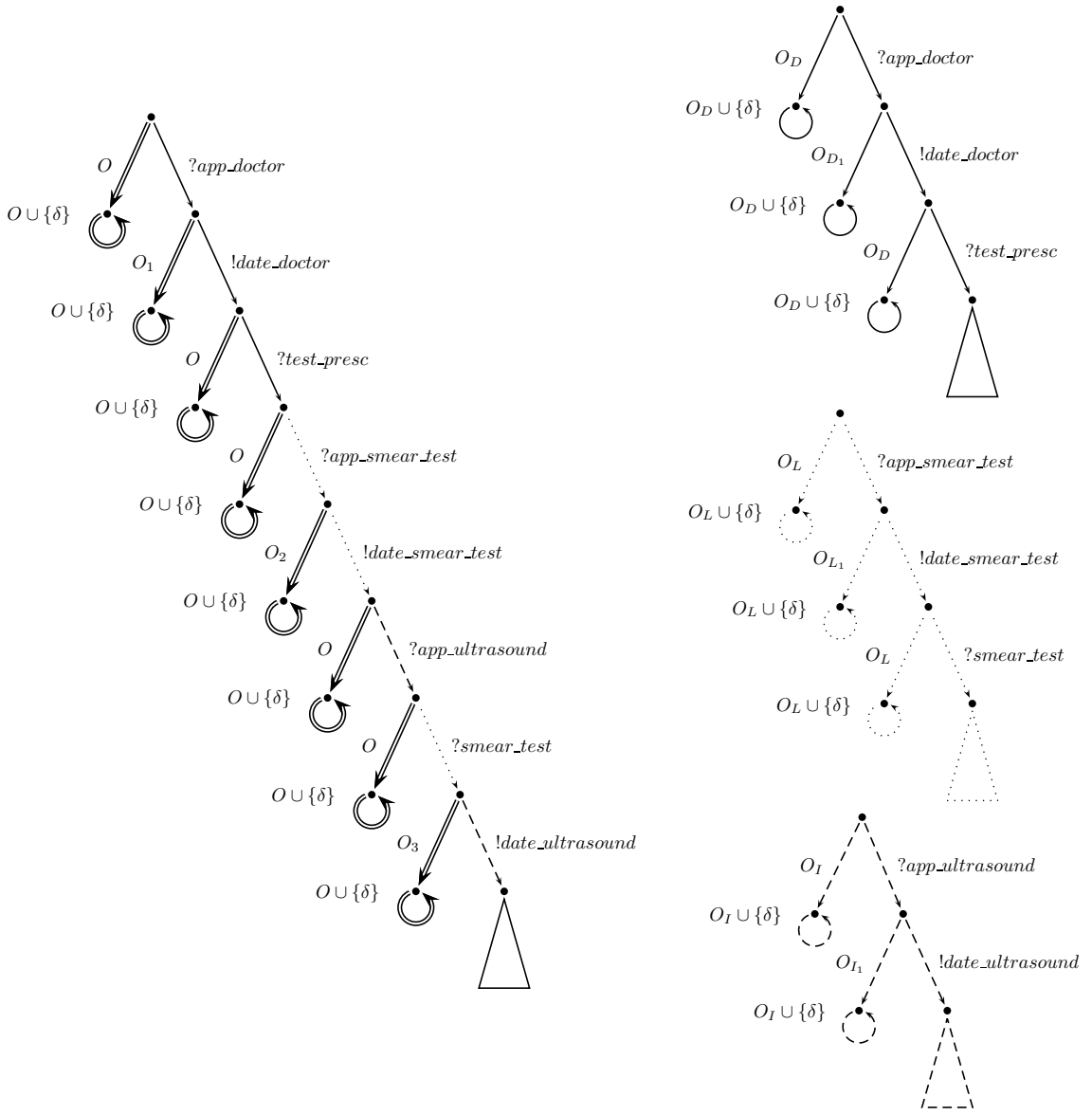
As usual, (global or local) test cases cannot block output from the SUT: if the SUT produces an output then the test case should be able to record this situation. Thus, for every state q of a global test case t (resp. local tester t_p) and output $!o \in O \cup \{\delta\}$ (resp. output $!o_p \in O_p \cup \{\delta\}$) we have that $q \xrightarrow{!o}$ (resp. $q \xrightarrow{!o_p}$).

We denote by $\perp$ the global test case that cannot send input to the SUT and thus whose traces are all elements of $(O \cup \{\delta\})^*$. We let $\perp_p$ denote the corresponding local tester for port p, whose set of traces is $(O_p \cup \{\delta\})^*$.

As usual, global test cases and local testers have a tree-like structure, that is, the induced graph is acyclic except for those loops created by occurrences of $\perp$ and $\perp_p$. □

In Figure 13 we show a global test case for our running example and its three associated local testers, one local tester for each of the three considered ports. In the graph, arrows still indicate the port where the actions are performed, with the exception of the double arrow appearing in the global test case that indicate outputs at any port.

DOI: 10.1002/spe



where O is the total set of outputs of the system, $O_1 = O \setminus \{!date\_doctor\} \cup \{\delta\}$, $O_2 = O \setminus \{!date\_smear\} \cup \{\delta\}$, $O_3 = O \setminus \{!date\_ultrasound\} \cup \{\delta\}$, O_D, O_L and O_I are the sets of outputs in the ports of the doctor office, the laboratory and the image diagnosis office, respectively, $O_{D_1} = O_D \setminus \{!date\_doctor\} \cup \{\delta\}$, $O_{L_1} = O_L \setminus \{!date\_smear\} \cup \{\delta\}$, and $O_{I_1} = O_I \setminus \{!date\_ultrasound\} \cup \{\delta\}$.

Figure 13. Global (left) and local (right) test cases.

The following function, an adaption of the one given in [22], takes a global test case and returns local testers. In this definition, for a set A we have that 2^A denotes the powerset of A. The approach used is similar to the standard method for constructing a deterministic finite automata from a non-deterministic one.

DOI: 10.1002/spe

Definition 13
Let $\mathcal{P}$ be a set of ports, $t = (Q, I, O \cup \{\delta\}, T, q_{in})$ be a global test case and $p \in \mathcal{P}$ be a port. We have that $local_p(t)$ denotes the local tester at p defined as $(2^Q, I_p, O_p \cup \{\delta\}, T', Q_{in})$, where

1. $Q_{in} = \{q \in Q | \exists \sigma \in (I \cup O \cup \{\delta\})^*.q_{in} \stackrel{\sigma}{\Longrightarrow} q \wedge \pi_p(\sigma) = \epsilon\}$.
2. For $a \in I_p \cup O_p \cup \{\delta\}$, $(Q_1, a, Q_2) \in T'$ if and only if Q_2 is the set of states $q_2 \in Q$ such that there exists $q_1 \in Q_1$ and $\sigma \in (I \cup O \cup \{\delta\})^*$ such that $\pi_p(\sigma) = a$ and $q_1 \stackrel{\sigma}{\Longrightarrow} q_2$.

□

The first rule says that the initial state of $local_p(t)$ is the set of states reachable from the initial state of t without observations at p. The second rule says that if Q_1 is a set of states of $local_p(t)$ then action $a \in I \cup O \cup \{\delta\}$ takes Q_1 to the set of states that are reachable from states of Q_1 using sequences in which the only observation at p is the event a.

The previous method can produce local testers with many states. Therefore, if we need to actually construct local test cases we can use an adaption of the algorithm given in [31], an extended version of the original framework [23, 22], to construct local test cases from controllable global test cases that works in low-order polynomial time.

Next we introduce a notion of parallel composition between a system and a (global or local) test case.

Definition 14
Let $\mathcal{P} = \{1, \ldots, n\}$ be a set of ports, $M_m \in m\mathcal{IOTS}(I, O)$, t be a global test case for M_m and $t_l = (t_1, \ldots, t_n)$ be a local test case for M_m. We introduce the following notation.

1. $M_m||t$ denotes the application of t to M_m. The system $M_m||t$ belongs to $m\mathcal{IOTS}(I, O \cup \{\delta\})$ and is formed by M_m and t synchronising on all actions (including quiescence).
2. $M_m||t_l$ denotes the application of t_l to M_m. The system $M_m||t_l$ belongs to $m\mathcal{IOTS}(I, O \cup \{\delta\})$ and it is formed from M_m and t_l by M_m and t_p synchronising on actions in $I_p \cup O_p$, for all $p \in \mathcal{P}$. In addition, M_m, $t_1, \ldots, t_n$ synchronise on δ.

Since $M_m||t$ and $M_m||t_l$ are $m\mathcal{IOTS}$s, the notation already introduced can be applied to them. In particular, we let $\mathcal{T}r(M_m, t)$ (resp. $\mathcal{T}r(M_m, t_l)$) denote the set of traces that can result from $M_m||t$ (resp. $M_m||t_l$) and their prefixes. □

The following notation is used in order to reason about the application of test cases to systems. Let us note that we have two notions of *passing* a test: taking into account marked states or not. These definitions are given for the **sdioco** implementation relation but it is straightforward to adapt them to **sdioco**$'$ and **tsdioco**.

Definition 15
Let $M_m^{Spec}, M_m^{SUT} \in m\mathcal{IOTS}(I, O)$ and t be a global test case for M_m^{Spec}.

1. A trace σ is a *test run* for M_m^{SUT} with t if $M_m^{SUT}||t \stackrel{\sigma\delta}{\Longrightarrow}$ (and so at the end of this test run the SUT is quiescent).
2. Implementation M_m^{SUT} **passes** test run σ with t for M_m^{Spec} if there exists $\sigma' \in \mathcal{T}r(M_m^{Spec})$ such that $\sigma' \sim \sigma$. Otherwise M_m^{SUT} **fails** σ with t for M_m^{Spec}.
3. Implementation M_m^{SUT} **passes** test run σ with t for the scenarios given by M_m^{Spec} if there exists a quiescent trace $\sigma' = a_1, \ldots, a_m$ such that the following hold:
   (a) $M_m^{Spec} \stackrel{\sigma'}{\Longrightarrow}$ and $\sigma' \sim \sigma$.
   (b) There is a derivation $M_m^{Spec} \stackrel{a_1}{\longrightarrow} q_1 \stackrel{a_2}{\longrightarrow} q_2 \ldots q_{m-1} \stackrel{a_m}{\longrightarrow} q_m$ in which $J = \{j_1, \ldots, j_r\} \subseteq \{1, \ldots, m\}$ is the maximal set of indexes such that $q_{j_i} \in \mathcal{Q}$ for all $1 \leq i \leq m$. Let $\sigma'_1, \ldots, \sigma'_{r+1}$ be sequences such that $\sigma' = \sigma'_1 \cdots \sigma'_{r+1}$ and $M_m^{Spec} \stackrel{\sigma'_1}{\Longrightarrow} q_{j_1} \stackrel{\sigma'_2}{\Longrightarrow} q_{j_2} \ldots q_{j_r} \stackrel{\sigma'_{r+1}}{\Longrightarrow} q_m$. Then, there exist sequences $\sigma_1, \ldots, \sigma_{r+1}$ such that for all $1 \leq j \leq r+1$ we have that $\sigma'_j \sim \sigma_j$ and $\sigma = \sigma_1 \cdots \sigma_{r+1}$.

DOI: 10.1002/spe

Otherwise M_m^{SUT} **fails** σ with t for the scenarios given by M_m^{Spec}.

4. Implementation M_m^{SUT} **passes** test case t for M_m^{Spec} if M_m^{SUT} **passes** every possible test run of M_m^{SUT} with t for M_m^{Spec} and otherwise M_m^{SUT} **fails** t for M_m^{Spec}.
5. Implementation M_m^{SUT} **passes** test case t for the scenarios given by M_m^{Spec} if M_m^{SUT} **passes** every possible test run of M_m^{SUT} with t for the scenarios given by M_m^{Spec} and otherwise M_m^{SUT} **fails** t for the scenarios given by M_m^{Spec}.

□

Let us note that our way of defining how to pass test cases is not standard since our test cases are not equipped with pass/fail states. Therefore, we need the specification to decide whether a test run is expected by the specification. Naturally, we are just using the specification as an *oracle* as commonly done in model-based testing.

5.1. Deterministic and controllable test cases

When applying test cases to SUTs, it is important to restrict ourselves to deterministic test cases. A local test case t is said to be *deterministic* for a specification s if the interaction between s and t cannot reach a situation in which more than one input can be sent [22]. In particular, there cannot be situations in which more than one local tester is capable of sending input since, in such a situation, the order in which these inputs are received by the SUT is unknown.

Definition 16
Let $M_m^{Spec} \in m\mathcal{IOTS}(I,O)$ be a specification. We say that the local test case t_l is *deterministic for* M_m^{Spec} if there do not exist traces σ_1 and σ_2, with $\sigma_2 \sim \sigma_1$, and $?i_1, ?i_2 \in I$, with $?i_1 \neq ?i_2$, such that $M_m^{Spec}||t_l \xRightarrow{\sigma_1 ?i_1}$ and $M_m^{Spec}||t_l \xRightarrow{\sigma_2 ?i_2}$. □

It is easy to show that the local testers being deterministic does not guarantee that the corresponding local test case is deterministic. For example, two or more deterministic local testers could start by sending input to the SUT.

But even restricting to deterministic test cases is not enough in the distributed test architecture to have a *controllable* testing framework. Let us consider a specification M_m^{Spec} such that $\mathcal{T}r(M_m^{Spec})$ is given by the set of prefixes of $?i_U!o_L!o_U?i_L$ plus the traces obtained by completing this to make it input-enabled. We could have a local test case (t_U, t_L) in which t_U sends $?i_U$ and expects to observe $!o_U$ and t_L sends $?i_L$ after observing $!o_L$. Then t_L does not know when to send $?i_L$ and this is a form of nondeterminism. We obtain the same problem with the corresponding global test case if we wish to apply it in the distributed test architecture.

The following is based on the definition of a test case being controllable, which is taken from [23], and is a necessary and sufficient condition under which we avoid this form of nondeterminism (when states are not marked). This essentially corresponds to the testers not taking the opportunity to synchronise in marked states and so we use the term strongly controllable.

Definition 17
A global test case t is *strongly controllable* for $M_m \in m\mathcal{IOTS}(I,O)$ if there does not exist port $p \in \mathcal{P}$, $\sigma_1, \sigma_2 \in \mathcal{T}r(M_m, t)$ and $?i_p \in I_p$ with $\sigma_1 ?i_p \in \mathcal{T}r(M_m, t)$, $\sigma_2 ?i_p \notin \mathcal{T}r(M_m, t)$ and $\pi_p(\sigma_1) = \pi_p(\sigma_2)$. □

If there are marked states then the local testers can synchronise in these states and in effect this adds additional observational power that can be used to make test cases controllable. Thus, a test case t is *weakly controllable* for M_m if when a global trace $\sigma \in \mathcal{T}r(M_m, t)$ has been produced, when synchronising in marked states, then at each point every local tester always knows what to do next (apply an input or wait for output).

Definition 18
A global test case t is *weakly controllable* for $M_m \in m\mathcal{IOTS}(I,O)$ if there do not exist port

DOI: 10.1002/spe

$p \in \mathcal{P}$ and $?i_p \in I_p$ such that there is a derivation $M_m||t \stackrel{\sigma_1}{\Longrightarrow} q_1 \stackrel{\sigma_2}{\Longrightarrow} \ldots \stackrel{\sigma_r}{\Longrightarrow} q_r \stackrel{\sigma_{r+1}}{\Longrightarrow} q_{r+1}$ in which $q_1, \ldots, q_r$ are the only traversed marked states and a derivation $M_m||t \stackrel{\sigma'_1}{\Longrightarrow} q'_1 \stackrel{\sigma'_2}{\Longrightarrow} \ldots \stackrel{\sigma'_r}{\Longrightarrow} q'_r \stackrel{\sigma'_{r+1}}{\Longrightarrow} q'_{r+1}$ in which $q'_1, \ldots, q'_r$ are the only marked states such that $\sigma_j \sim \sigma'_j$ for all $1 \leq j \leq r$, $\pi_p(\sigma_{r+1}) = \pi_p(\sigma'_{r+1})$, $q_{r+1} \xrightarrow{?i_p}$ and there is no q' such that $q'_{r+1} \xrightarrow{?i_p} q'$. In such a situation we will usually say that we are *synchronising in marked states.* □

The main difference between weak and strong controllability is that we can compare the global traces between marked states using $\sim$ and so, in effect, the local tester at p can be aware of the global traces that occurred between the marked states (up to $\sim$). After the last marked state q_r, the tester at p can only observe the projection at p of the global trace that occurred after q_r. In some situations this requirement is not realistic since it effectively requires that synchronisation always happens when it is allowed but where this is the case we can instead use strong controllability. Throughout the rest of this section we investigate the situation in which the local testers know when they can synchronise and so can take advantage of such synchronisation. Later we discuss conditions under which this is the case.

It has been shown that without scenarios, if a test case is controllable then, as long as no failures occur in testing, each input is supplied by a local tester at the point specified in the test case [23]. A similar result holds in the current framework, but with the advantage that scenarios reduce the set of traces that can occur.

Proposition 8
Let $M_m^{SUT}, M_m^{Spec} \in m\mathcal{IOTS}(I, O)$ and t be a weakly controllable test case for the specification M_m^{Spec} so that synchronising in marked states occurs when applying t. If an input $?i$ is sent after $\sigma \in \mathcal{T}r(M_m^{Spec}, t)$ then $\sigma?i \in \mathcal{T}r(t)$.
Proof
We prove the result by contradiction: we assume that $?i$ is sent after $\sigma \in \mathcal{T}r(M_m^{Spec}, t)$ but $\sigma?i \notin \mathcal{T}r(t)$. Further, let us suppose that $?i$ is supplied at port p and so there exists a trace $\sigma'?i \in \mathcal{T}r(M_m^{Spec}, t)$ such that σ and σ' are indistinguishable to the tester at p even when synchronising at marked states. We therefore must have that the following hold:

1. $\sigma = \sigma_1 \ldots \sigma_{r+1}$, where $\sigma_1 \ldots \sigma_r$ are the prefixes of σ that reach marked states in M_m^{Spec}, and
2. $\sigma' = \sigma'_1 \ldots \sigma'_{r+1}$, where $\sigma'_j \sim \sigma_j$, for $1 \leq j \leq r$, and $\pi_p(\sigma'_{r+1}) = \pi_p(\sigma_{r+1})$.

But, since t is weakly controllable, if $?i$ can be sent after σ' then we must have that $?i$ can be sent after σ, providing a contradiction as required. □

This result proves that using weakly controllable test cases and synchronising in marked states is sufficient to ensure that inputs are sent at the expected/specified time. Thus, we know that we do not require a test case to be strongly controllable: it is sufficient for it to be weakly controllable.

Even though the notion of controllability is restrictive, it is important since it appropriately captures the idea of the local testers knowing when to apply input. However, there is another issue to consider that is illustrated by the following situation.

1. A specification M_m in which the only possible transition is from the initial state to a different state through output $!o_U$. In addition, all inputs are available in both states, producing a loop.
2. A global test case t that after $!o_U$ can apply input $?i_L$ at port L and that after $!o'_U$ can apply input $?i'_L$ at port L.

When t and M_m synchronise, t cannot send input $?i'_L$ since it cannot receive output $!o'_U$ from M_m. However, if we take the projections of t then the resultant local tester at port L can send input $?i'_L$. This situation is caused by there being parts of t that cannot be 'reached' when synchronising with M_m and so we define a (desirable) condition under which this cannot happen.

DOI: 10.1002/spe

Definition 19
Let M_m be an $m\mathcal{IOTS}(I,O)$. A global test case t is said to be *reduced* for M_m if there do not exist a trace $\sigma \in \mathcal{A}ct^*$ and an input $?i$ such that $\sigma?i \in \mathcal{T}r(t) \setminus \mathcal{T}r(M_m)$.

Clearly, if a global test case t is not reduced for M_m, when testing from M_m we can remove parts of it in order to produce a reduced global test case t' such that $\mathcal{T}r(M_m,t) = \mathcal{T}r(M_m,t')$ and so we lose nothing in restricting attention to reduced global test cases.

Let us note that it is not enough to have that specifications are input-enabled to ensure that global test cases are reduced. In fact, we can consider a trace σ that cannot be performed by M_m (therefore, it must contain at least one output) and such that $\sigma?i_p \in Tr(t)$. In addition, controllability does not imply that the test case will be reduced. For example, let us consider a global test case t in which input $?i_U$ only occurs after $!o_L$ but M_m cannot produce $!o_L$ in its initial state. The test case t is controllable for M_m. We have that $?i_U$ is irrelevant when we apply t to M_m but when we take projections suddenly the local tester can apply $?i_U$ to M_m (when it is in its initial state).

The next result answers the question of whether we can always implement a controllable global test case by using a weakly controllable test case.

Proposition 9
Let $\mathcal{P} = \{1,\ldots,n\}$ be the set of ports and $M_m \in m\mathcal{IOTS}(I,O)$. If t is a reduced global test case for M_m and $t_l = (local_1(t),\ldots,local_n(t))$ then:

1. $\mathcal{T}r(M_m,t) \subseteq \mathcal{T}r(M_m,t_l)$.
2. $\mathcal{T}r(M_m,t_l) \subseteq \mathcal{T}r(M_m,t)$ if and only if t is weakly controllable for M_m.

Proof
Let $\mathcal{T}r(t_l)$ denote the set of traces formed from interleavings of traces from $\mathcal{T}r(local_1(t)),\ldots,\mathcal{T}r(local_n(t))$. It is straightforward to prove that for all $\sigma \in \mathcal{T}r(t)$ and $p \in \mathcal{P}$ there exists $\sigma_p \in \mathcal{T}r(local_p(t))$ such that $\sigma_p = \pi_p(\sigma)$. In addition, we also have that $\mathcal{T}r(M_m,t) = \mathcal{T}r(M_m) \cap \mathcal{T}r(t)$ and $\mathcal{T}r(M_m,t_l) = \mathcal{T}r(M_m) \cap \mathcal{T}r(t_l)$, and so $\mathcal{T}r(t) \subseteq \mathcal{T}r(t_l)$. This completes the proof of the first part of the result.

Concerning the second part of the result, we begin with the right to left implication. Let us assume that t is weakly controllable for M_m. We will prove that for all $\sigma \in \mathcal{T}r(M_m,t_l)$ we have that $\sigma \in \mathcal{T}r(M_m,t)$. We will prove the result by induction on the length of σ. Clearly the result holds for the base case $\sigma = \epsilon$. Thus, let us assume that the result holds for all traces of length less than $k > 0$ and σ has length k. Thus, $\sigma = a\sigma'$ for some $a \in \mathcal{A}ct$. We distinguish two cases:

1. $a = \delta$. Then $t_p \xrightarrow{\delta} t'_p$ for all $p \in \mathcal{P}$ and $t \xrightarrow{\delta} t'$, $t'_p = local_p(t')$, and t' is weakly controllable for the process M'_m such that $M_m \stackrel{\delta}{\Longrightarrow} M'_m$. The result thus follows from the inductive hypothesis.
2. $a \in I_p \cup O_p$ for port p. In this case there exists t'_p such that $t_p \xrightarrow{a} t'_p$. Since $t_p = local_p(t)$, it must be possible to have a at p in t before any other event at p and before any marked states. Let σ_p be the shortest sequence in $((I \setminus I_p) \cup (O \setminus O_p))^*$ such that $\sigma_p a \in \mathcal{T}r(t)$ and no path of M_m with label σ_p contains marked states. But $\pi_p(\sigma_p) = \pi_p(\epsilon)$ and neither contains marked states and so, since t is weakly controllable for M_m, we have that $\sigma_p = \epsilon$. Thus, there exists t' such that $t \xrightarrow{a} t'$. In addition, $t'_p = local_p(t')$, $t_{p'} = local_{p'}(t')$ for $p' \in \mathcal{P} \setminus \{p\}$, and t' is weakly controllable for the process M'_m such that $M_m \stackrel{a}{\Longrightarrow} M'_m$. The result thus follows from the inductive hypothesis.

In order to prove the left to right implication, we assume that $\mathcal{T}r(M_m,t_l) \subseteq \mathcal{T}r(M_m,t)$ and will prove that t is weakly controllable for M_m. We use proof by contradiction, assuming that t is not weakly controllable for M_m and so there exist $\sigma,\sigma' \in \mathcal{T}r(M_m,t)$ and port $p \in \mathcal{P}$ such that the following hold:

1. $\sigma = \sigma_1 \ldots \sigma_{r+1}$, where $\sigma_1 \ldots \sigma_r$ are the prefixes of σ that reach marked states in M_m,

DOI: 10.1002/spe

Input: Specification M_m^{Spec} and a deterministic reduced global test case t.
Output: If t is weakly controllable for M_m then output `True` else output `False`.

{`Initialisation`}
Let N denote the set of nodes of $M_m^{Spec}||t$;
For every node $n \in N$ and port p, let n_p be the projection of the trace of $M_m^{Spec}||t$ reaching n;
{`Main loop`}
For all $n, n' \in N$ with $n \neq n'$ and $p \in \mathcal{P}$ do

- Define $\sigma_1, \ldots, \sigma_r$ such that the trace of $M_m^{Spec}||t$ that reaches n is $\sigma_1 ... \sigma_r$ and the marked states of this path in $M_m^{Spec}||t$ to n are after the subtraces of the form $\sigma_1 \ldots \sigma_i$, $0 \leq i \leq r$, with the possible exception of the initial and final states of the path;
- Similarly, define $\sigma'_1, \ldots, \sigma'_{r'}$ such that the trace of $M_m^{Spec}||t$ that reaches n' is $\sigma'_1 ... \sigma'_{r'}$;
- If $n_p = n'_p$, $r = r'$, for all $1 \leq i < r$ we have that $\sigma_i \sim \sigma'_i$, and there is an input $?i_p \in I_p$ such that $n \xrightarrow{?i_p}$ and we do not have that $n' \xrightarrow{?i_p}$ then output `False` and terminate;

output `True` and terminate.

Figure 14. Algorithm 2: deciding whether a global test case is weakly controllable for a process.

2. $\sigma' = \sigma'_1 \ldots \sigma'_{r+1}$, where $\sigma'_j \sim \sigma_j$, for $1 \leq j \leq r$ and $\pi_p(\sigma'_{r+1}) = \pi_p(\sigma_{r+1})$, and
3. there exists $?i_p \in I_p$ such that $\sigma ?i_p \in \mathcal{T}r(M_m, t)$ and $\sigma' ?i_p \notin \mathcal{T}r(M_m, t)$.

We therefore have that the tester at p cannot distinguish between $\sigma_1 \ldots \sigma_r$ and $\sigma'_1 \ldots \sigma'_r$ and also then between σ_{r+1} and σ'_{r+1} (since $\pi_p(\sigma_{r+1}) = \pi_p(\sigma'_{r+1})$). Thus, the local tester t_p must be able to have $\pi_p(\sigma'_{r+1})?i_p$ after $\sigma'_1 \ldots \sigma'_r$.

Further, for $q \in \mathcal{P} \setminus \{p\}$, we have that $\pi_q(\sigma') = \pi_q(\sigma) \in \mathcal{T}r(t_q)$ and so $\sigma' ?i_p \in \mathcal{T}r(t_l)$. Finally, since $\sigma' \in \mathcal{T}r(M_m, t)$ and M_m is input enabled we have that $\sigma' ?i_p \in \mathcal{T}r(M_m, t)$, providing a contradiction as required. □

The following result is an immediate corollary from Proposition 9, Definition 16, and a global test case being deterministic.

Corollary 1
Let $M_m \in m\mathcal{IOTS}(I, O)$ with set of ports $\mathcal{P} = \{1, \ldots, n\}$ and t be a weakly controllable reduced global test case for M_m. We have that $t_l = (local_1(t), local_2(t), \ldots, local_n(t))$ is deterministic for M_m.

It is desirable to apply weakly controllable test cases since this allows each local tester to know when to apply an input. Essentially, a global test case is a tree in which each leaf has a self loop for each output. Algorithm 2 given in Figure 14 simply considers the nodes of $M_m^{Spec}||t$ that can be followed by an input and determines whether any two of these define a situation in which there is a controllability problem. It thus initially forms the tree that represents $M_m^{Spec}||t$. For each pair of nodes n and n' reached by traces σ and σ' it then checks the conditions required under weak controllability. First, is splits σ into $\sigma_1, \ldots, \sigma_r$ and σ' into $\sigma'_1, \ldots, \sigma'_{r'}$ based on marked states. Thus, each trace is divided into a sequence of scenarios followed by a final trace (σ_r and $\sigma'_{r'}$). We can assume that the testers synchronise in marked states. Thus, for example, at the end of σ each tester knows that there has been a sequence of $r-1$ previous scenarios and knows that the traces of these were equivalent, under $\sim$, to $\sigma_1, \ldots, \sigma_{r-1}$. In addition, the tester at p has observed $\pi_p(\sigma_r)$ after this sequence of scenarios. This can only look like σ', to the tester at p, if σ' splits into a sequence of equivalent scenarios. Thus, there is only a problem under weak controllability if we have the same number of

DOI: 10.1002/spe

scenarios ($r' = r$), these are all equivalent under $\sim$ (for all $1 \leq i < r$ we have that $\sigma'_i \sim \sigma_i$) and the observations after the last scenario is the same at p ($\pi_p(\sigma'_r) = \pi_p(\sigma_r)$. These are the conditions specified under the definition of weak controllability and the conditions explicitly checked in Algorithm 2. The following result easily follows.

Theorem 2
Let M_m be an $m\mathcal{IOTS}(I, O)$ and t be a reduced deterministic global test case. We have that t is weakly controllable for M_m^{Spec} if and only if Algorithm 2 returns `True` for M_m^{Spec} and t. In addition, this algorithm operates in time that is polynomial in terms of the size of $M_m^{Spec}||t$.

When considering weak controllability we have assumed that each local tester knows when it can synchronise with the other testers. As we observed, when this is not the case we can instead use controllability but clearly it is beneficial to use weak controllability when possible since this allows more tests to be applied in a controllable manner. This assumption, that the local testers know when they can synchronise, is reasonable in at least the following situations.

1. If the local tester at port p can be in the situation in which it can synchronise after the local trace σ_p at p then it can always synchronise after observing σ_p.
2. Each local tester can observe the opportunity to synchronise. Let us suppose, for example, that scenarios correspond to transactions in a database system in which each user interacts with the system through a terminal. Then, we might have that a message appears on the screen of each terminal stating that the transaction has finished. We could model this type of situation by representing the opportunity to synchronise as a self-loop transition with a label η that is not in $\mathcal{A}ct$ and defining the projection function π_p so that η is observed at every port.

5.2. Implementations relations for testing with controllable tests

Once we have studied the main properties of controllable test cases, we can define new implementation relations if we restrict testing to the use of controllable test cases.

Definition 20
Let $M_m^{SUT}, M_m^{Spec} \in \mathcal{IOTS}(I, O)$. We write M_m^{SUT} **c-dioco** M_m^{Spec} if for every strongly controllable local test case t_l for M_m^{Spec} we have that M_m^{SUT} **passes** t_l for M_m^{Spec}. We write M_m^{SUT} **c-sdioco** M_m^{Spec} if for every weakly controllable local test case t_l for M_m^{Spec} we have that M_m^{SUT} **passes** t_l for the scenarios given by M_m^{Spec}. □

We now study how **dioco** and **sdioco** relate to **c-dioco** and **c-sdioco**, respectively.

Proposition 10
Let $M_m^{SUT}, M_m^{Spec} \in \mathcal{IOTS}(I, O)$. We have M_m^{SUT}**dioco** M_m^{Spec} implies M_m^{SUT} **c-dioco** M_m^{Spec}. Further, there exists processes M_m and M'_m such that M'_m**c-dioco** M_m but we do not have that M'_m**dioco** M_m.
Proof
The first part follows from the definitions, with **c-dioco** restricting consideration to strongly controllable local test cases. Let us consider the processes s_1 and i_1 shown in Figure 15 which are incomparable under **dioco**. The only strongly controllable local test cases for s_1 involve input at no more than one port and for each such test case neither process can produce output. We therefore have that i_1 **c-dioco** s_1 as required.

Proposition 11
Let $M_m^{SUT}, M_m^{Spec} \in m\mathcal{IOTS}(I, O)$. We have M_m^{SUT}**sdioco** M_m^{Spec} implies M_m^{SUT}**c-sdioco** M_m^{Spec}. Further, there exists processes M_m and M'_m such that M'_m**c-sdioco** M_m but we do not have that M'_m**sdioco** M_m.
Proof
The first part follows again from the definitions, with **c-sdioco** restricting consideration to weakly controllable local test cases. In order to show the second part, let us consider the processes s_1 and i_1 shown in Figure 15, which are incomparable under **sdioco**. Again, the

DOI: 10.1002/spe

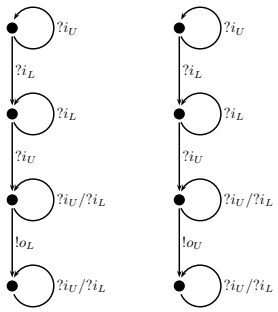


Figure 15. Processes s_1 (left) and i_1 (right).

only weakly controllable local test cases involve input at no more than one port and for each such test case neither process can produce output. We therefore have that i_1 **c-sdioco** s_1 as required. □

6. CONCLUSIONS

The work reported in this paper represents a continuation of our work on formal testing of systems with distributed ports. We have introduced two new formalisms that allow us to specify situations where all the components of a distributed system wait for a certain operation to happen or where even though a total global trace cannot be constructed it can be inferred that a certain action took place before another one. We have shown that the two formalism have similar expressive power and we have given a translation mechanism from one of the formalisms to the other.

We have introduced three implementation relations, one for the trace-based formalism and two for the formalism in which we mark states. These represent suitable extensions of previously established relations. Since we are mainly interested in formal testing frameworks, we have defined what it means for a system under test to pass a test case under the new conditions. We have studied the special case of *controllable* test cases and analysed how the new conditions affect the notion of controllability.

Even though this paper represents a step forward with respect to our previous contribution on testing systems with distributed ports and scenarios, there are several lines to continue our work. First, we have to define a test derivation algorithm so that we only apply those test cases that are somehow *related* to the corresponding specification. We will take as initial step the one for **dioco** and **c-dioco** given in [31]. We would also like to take into account some variants that were sketched in this paper but not fully exploited. For example, an interesting alternative to marking states in the specification is to mark states in local/global testers extracted from the specification and *forget* the marked states of the specification.

REFERENCES

1. Clarke E, Grumberg O, Peled D. *Model Checking*. MIT Press, 2000.
2. Baier C, Katoen JP. *Principles of Model Checking*. MIT Press, 2008.
3. Myers G. *The Art of Software Testing*. 2nd edn., John Wiley and Sons, 2004.
4. Ammann P, Offutt J. *Introduction to Software Testing*. Cambridge University Press, 2008.
5. Broy M, Jonsson B, Katoen JP, Leucker M, Pretschner A ((eds.)). *Model-based Testing of Reactive Systems, LNCS 3472*. Springer, 2005.
6. Hierons R, Bowen J, Harman M ((eds.)). *Formal Methods and Testing, LNCS 4949*. Springer, 2008.
7. Jacky J, Veanes M, Campbell C, Schulte W. *Model-Based Software Testing and Analysis with C#*. Cambridge University Press, 2008.
8. Frantzen L, Merayo M, Núñez M. A brief history of A-MOST. *Journal of Logic and Algebraic Programming* 2009; **78**(6):417–424.

DOI: 10.1002/spe

9. Utting M, Legeard B. *Practical Model-Based Testing: A Tools Approach*. Morgan-Kaufmann, 2007.
10. Hierons R, Bogdanov K, Bowen J, Cleaveland R, Derrick J, Dick J, Gheorghe M, Harman M, Kapoor K, Krause P, *et al.*. Using formal methods to support testing. *ACM Computing Surveys* 2009; **41**(2).
11. ISO/IEC JTC 1 JTC. *International Standard ISO/IEC 9646-1. Information Technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts*. ISO/IEC, 1994.
12. Sarikaya B, Bochmann Gv. Synchronization and specification issues in protocol testing. *IEEE Transactions on Communications* 1984; **32**:389–395.
13. Dssouli R, Bochmann Gv. Error detection with multiple observers. *5th WG6.1 Int. Conf. on Protocol Specification, Testing and Verification, PSTV'85*, North-Holland, 1985; 483–494.
14. Dssouli R, Bochmann Gv. Conformance testing with multiple observers. *6th WG6.1 Int. Conf. on Protocol Specification, Testing and Verification, PSTV'86*, North-Holland, 1986; 217–229.
15. Boyd S, Ural H. The synchronization problem in protocol testing and its complexity. *Information Processing Letters* 1991; **40**(3):131–136.
16. Luo G, Dssouli R, Bochmann Gv. Generating synchronizable test sequences based on finite state machine with distributed ports. *6th IFIP Workshop on Protocol Test Systems, IWPTS'93*, North-Holland, 1993; 139–153.
17. Tai KC, Young YC. Synchronizable test sequences of finite state machines. *Computer Networks and ISDN Systems* 1998; **30**(12):1111–1134.
18. Rafiq O, Cacciari L. Coordination algorithm for distributed testing. *The Journal of Supercomputing* 2003; **24**(2):203–211.
19. Ural H, Williams C. Constructing checking sequences for distributed testing. *Formal Aspects of Computing* 2006; **18**(1):84–101.
20. Tretmans J. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools* 1996; **17**(3):103–120.
21. Tretmans J. Model based testing with labelled transition systems. *Formal Methods and Testing, LNCS 4949*, Springer, 2008; 1–38.
22. Hierons R, Merayo M, Núñez M. Implementation relations for the distributed test architecture. *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*, Springer, 2008; 200–215.
23. Hierons R, Merayo M, Núñez M. Controllable test cases for the distributed test architecture. *6th Int. Symposium on Automated Technology for Verification and Analysis, ATVA'08, LNCS 5311*, Springer, 2008; 201–215.
24. Brinksma E, Heerink L, Tretmans J. Factorized test generation for multi-input/output transition systems. *11th IFIP Workshop on Testing of Communicating Systems, IWTCS'98*, Kluwer Academic Publishers, 1998; 67–82.
25. Hierons R, Merayo M, Núñez M. Scenarios-based testing of systems with distributed ports. *10th Int. Conf. on Quality Software, QSIC'10*, IEEE Computer Society Press, 2010; 52–61.
26. Jard C, Jéron T, Kahlouche H, Viho C. Towards automatic distribution of testers for distributed conformance testing. *Joint Int. Conf. on Formal Description Techniques and Protocol Specification, Testing and Verification, FORTE'98*, Kluwer Academic Publishers, 1998; 353–368.
27. Jacob J. Refinement of shared systems. *The Theory and Practice of Refinement: Approaches to the Formal Development of Large-Scale Software Systems*, McDermid J (ed.). Butterworths, 1989; 27–36.
28. Haar S, Jard C, Jourdan GV. Testing input/output partial order automata. *Joint 19th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'07, and 7th Int. Workshop on Formal Approaches to Software Testing, FATES'07, LNCS 4581*, Springer, 2007; 171–185.
29. Bochmann Gv, Haar S, Jard C, Jourdan GV. Testing systems specified as partial order input/output automata. *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*, Springer, 2008; 169–183.
30. Huo J, Petrenko A. On testing partially specified IOTS through lossless queues. *16th Int. Conf. on Testing Communicating Systems, TestCom'04, LNCS 2978*, Springer, 2004; 76–94.
31. Hierons R, Merayo M, Núñez M. Implementation relations and test generation for systems with distributed interfaces. Submitted 2010.

DOI: 10.1002/spe