Scenarios-based Testing of Systems with distributed Ports*

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

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

Abstract

Current distributed systems are usually composed of several distributed components that communicate through specific ports. When testing these systems 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. In this paper we concentrate on the problem of formally testing systems with distributed components that, in general, have independent behaviors but that at certain points of time synchronization can occur. These situations appear very often in large real systems that regularly go through maintenance and/or update operations. 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 encode these special operations by marking some of the states of the specification. 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).*

*Research partially supported by the Spanish MEC projects WEST/FAST (TIN2006-15578-C02-01) and 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).

1 Introduction

The complexity of current systems is increasing both in terms of their size and of the capabilities that they incorporate. Therefore, software engineering techniques relying 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 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. However, a correct specification does not imply that we will obtain a correct system.

Software testing [1,11] is the technique most widely used to assess the correctness of software systems with respect to formal specifications. 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 [3, 5]. The main advantage of using a formal approach is that many testing processes can be automated (see [17] for a discussion on the advantages of formal testing and [4] for a survey on formal methods and testing).

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 then we are testing in the distributed test architecture, which has been standardised by the ISO [9]. It is already well known that the use of the distributed test architecture reduces test effectiveness (see, for example, [10, 12–14, 16]).

Most previous work on testing in the distributed test architecture has considered Deterministic Finite State Ma-

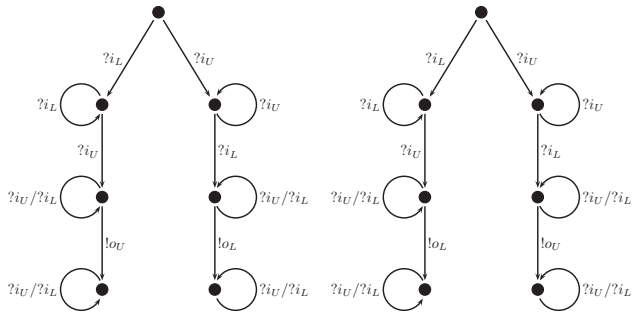

Figure 1. M_1 and M_2 are not related under ioco but are related under dioco.

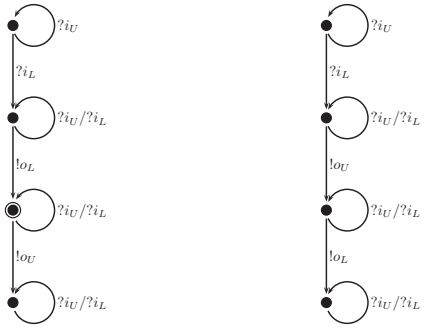

Figure 2. M_3 and M_4 are related under dioco but are not related under sdioco.

chines (DFSMs). However, the Input Output Transition System (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** [15], that is usually used in testing from an 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 IOTS in the distributed test architecture been investigated [6, 7].[1] This work introduced an implementation relation **dioco** with a very special feature: We compare traces of the SUT and of the specification only if we reach *quiescent* states, that is, states that are somehow *stable* because 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 [7]).

It is clear that the distinguishing power of our **dioco** relation is smaller 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. However, the theoretical 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 according to **ioco** because we can find sequences of actions that can be performed by M_2 that cannot be performed by M_1 (left hand side of Figure 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 (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 the distributed test architecture 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. However, there are frequent situations when we need that all the components of the system 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 any of the sequence of events, up to the causality relation underlying **dioco**, that takes the specification from its initial state to any of these marked states. For example, let us consider Figure 2. The system M_3 (left hand side of the figure) has a marked state. Therefore, M_4 (right hand side of the figure) is not a good implementation according to the new relation because, for example, it cannot perform the sequence $?i_L!o_L$. However, M_4 conforms to M_3 if we consider **dioco**. An additional motivation for the use of scenarios is as follows. Let us suppose that agents A and B interact with M 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

[1] An implementation relation **mioco** has been defined for testing from an IOTS that has multiple ports. However, this implementation relation assumes that there is a single tester that controls and observes the ports [2].

the same year. If A and B later communicate then they can deduce that a occurred before b even if they cannot reconstruct the total 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.

The implementation relation introduced in this paper is called **sdioco**, standing for Scenarios-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 would be possible to alternatively define our new relation by associating a set of traces with a specification. However, since a model defines a set of traces and a set of traces defines a model, there is little difference in the expressiveness of these two approaches and definitions are more compact when marked states are used and so this is the approach that we take in this paper.

In order to show that our new relation is a suitable extension of the previously mentioned implementation relations, we will 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.

The rest of the paper is structured as follows. In Section 2 we introduce our formalism to define distributed systems and give our new implementation relation. In Section 3 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 4 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 [15].

Definition 1 An *Input Output Transition System* (`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 (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 `IOTS`s with input set I and output set O.

Any state $q \in Q$ induces an `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 remark 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 we said in the introduction of the paper, 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 the distributed testing architecture quiescent states can be used to combine the traces observed at each port and reach a verdict. 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 remark 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**.

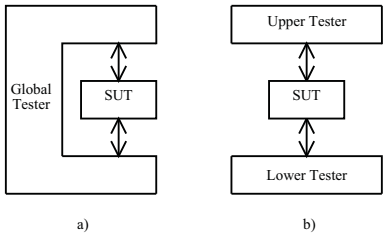

Figure 3. The local and distributed test architectures.

Definition 2 Let $M = (Q, I, O, T, q_{in})$ be an 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 $q_0, \dots, q_m$, 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' 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 IOTS it is usual to use the **ioco** relation [15] 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.

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 $out(M' \textbf{ after } \sigma) \subseteq out(M \textbf{ after } \sigma)$. □

2.2 Multi-port IOTSs with marked states

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 [9]. We use the term mIOTS 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 IOTS.

Definition 4 We will denote by $\mathcal{P}$ the set of ports. A *marked* IOTS (mIOTS) is a pair $M_m = (M, \mathcal{Q})$, where $M = (Q, I, O, T, q_{in})$ is an 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 mIOTSs 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 mIOTSs the concepts introduced in Definitions 1 and 2 for IOTSs. 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.

In order to keep compatibility with the **ioco** theory, we consider that specifications and implementations are defined by using the same formalism, that is, input-enabled, non output-divergent mIOTSs. However, we will not use the set of marked states associated with 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 those *local traces* that can be obtained from a global trace. In the following definition we also give an auxiliary function to compute the inputs appearing in a sequence of actions and introduce a relation $\sim$ to relate traces.

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 $\pi_p(z\sigma) = \pi_p(\sigma)$.

Let $\sigma \in \mathcal{A}ct^*$. We let $in(\sigma)$ denote the sequence of inputs appearing in σ. The function in can be defined by the following rules.

1. $in(\epsilon) = \epsilon$.
2. If $z \in I$ then $in(z\sigma) = z\, in(\sigma)$.
3. If $z \in O \cup \{\delta\}$ then $in(z\sigma) = in(\sigma)$.

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 $!o_U?i_U?i_L$. Next we define the **dioco** implementation relation [7].

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 $M_m^{SUT} \stackrel{\sigma}{\Longrightarrow} q$ for some q that is in a quiescent state, if there is a trace $\sigma_1 \in \mathcal{T}r(M_m^{Spec})$ such that $in(\sigma_1) \sim in(\sigma)$ then there exists a trace $\sigma' \in \mathcal{T}r(M_m^{Spec})$ such that $M_m^{Spec} \stackrel{\sigma'}{\Longrightarrow}$ and $\sigma' \sim \sigma$. □

Only traces reaching quiescent states 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 [7]. Let us remark that we have not used marked states in the previous definition since this is a feature relevant only for our new relation. Therefore, in the context of **dioco** it is the same to consider an `mIOTS` or its associated `IOTS`.

Given the fact that in this paper all processes are input-enabled we can simplify the previous definition.

Proposition 1 Let $M_m^{Spec}, M_m^{SUT} \in m\mathcal{IOTS}(I,O)$. We have M_m^{SUT} **dioco** M_m^{Spec} if and only if for every trace σ such that $M_m^{SUT} \stackrel{\sigma}{\Longrightarrow} q$ for some quiescent state q, there exists a trace σ' such that $M_m^{Spec} \stackrel{\sigma'}{\Longrightarrow}$ and $\sigma' \sim \sigma$. □

As we discussed in the introduction of the paper, 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. Therefore, in this paper we introduce a new implementation relation among `mIOTSs`.

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 $M_m^{SUT} \stackrel{\sigma}{\Longrightarrow} q$ for some q that is in a quiescent state, there exists a trace $\sigma' = a_1, \ldots, a_m$ such that the following two conditions hold:

- $M_m^{Spec} \stackrel{\sigma'}{\Longrightarrow}$ and $\sigma' \sim \sigma$.
- There is a derivation $M_m^{Spec} \stackrel{a_1}{\longrightarrow} q_1 \stackrel{a_2}{\longrightarrow} q_2 \cdots q_{m-1} \stackrel{a_m}{\longrightarrow} q_m$, with $q_1, \ldots, q_m$ states of M_m^{Spec}, in which $J = \{j_1, \ldots, j_r\} \subseteq \{1, \ldots, m\}$ is the set of indexes such that $q_j \in \mathcal{Q}$ if and only if $j \in J$ and $\sigma'_1, \ldots, \sigma'_{r+1}$ are the sequences such that $\sigma' = \sigma'_1 \cdots \sigma'_{r+1}$ and $M_m^{Spec} \stackrel{\sigma'_1}{\Longrightarrow} q_{j_1} \stackrel{\sigma'_2}{\Longrightarrow} q_{j_2} \cdots q_{j_r} \stackrel{\sigma'_{r+1}}{\Longrightarrow} q_m$. 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 remark that J is a set of indexes to label 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 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 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 **sdioco**$'$.

The next result indicates that our new relation 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 on the suitability of **sdioco** as a good implementation relation for distributed systems.

Proposition 2 Let $M_m^{Spec}, M_m^{SUT} \in m\mathcal{IOTS}(I,O)$, where $M_m^{Spec} = (M, \mathcal{Q})$ and $M = (Q, I, O, T, q_{in})$. 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}.

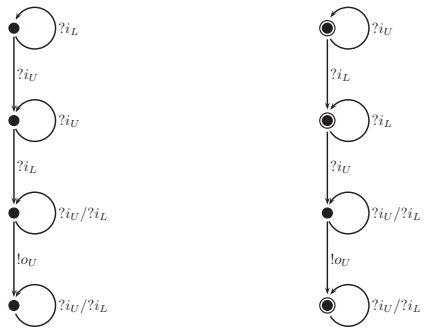

Figure 4. Quiescence alone does not capture marked states.

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 $out(M_m^{SUT}\mathbf{after}\ \sigma) \subseteq out(M_m^{Spec}\mathbf{after}\ \sigma)$. Let us suppose that $a \in out(M_m^{SUT}\mathbf{after}\ \sigma)$ 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 $a \in out(M_m^{Spec}\mathbf{after}\ \sigma)$ 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 1 Let us consider Figure 4: M_m^{SUT} is given in its left hand side while M_m^{Spec} is given in its right hand side. The marked states of M_m^{Spec} are 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 quiescent, and so could be 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 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 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.

Definition 8 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 $\mathcal{IOTS}(I, O \cup \{\delta\})$. 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 $t_p \in \mathcal{IOTS}(I_p, O_p \cup \{\delta\})$. 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 $q \stackrel{!o}{\longrightarrow}$ (resp. $q \stackrel{!o_p}{\longrightarrow}$).

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 $\perp_p$. □

The following function, an adaption of the one given in [7], 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.

Definition 9 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. $Q_{in} = \left\{ q \in Q \;\middle|\; \begin{array}{r} \exists \sigma \in (I \cup O \cup \{\delta\})^*.\ q_{in} \stackrel{\sigma}{\Longrightarrow} q \wedge \\ \pi_p(\sigma) = \epsilon \end{array} \right\}$.
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 $a \in I \cup O$ 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 definition is useful from the theoretical point of view since it provides an easy way to construct local test cases from global test cases. However, it produces local testers with huge amounts of states. Therefore, if we need to actually construct local test cases we use an adaption of the algorithm given in [8], a revised and extended version of [6, 7], to construct local test cases from controllable global test cases that works in low polynomial time. We omit this algorithm due to space limitations.

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

Definition 10 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). Marked states of the composition are given by the reached marked states of the system M_m.
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. Again, marked states of the composition are given by the reached marked states of the system M_m.
3. Since $M_m||t$ and $M_m||t_l$ are `mIOTSs`, 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 remark that we have two notions of *passing* a test: Taking into account marked states or not.

Definition 11 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 $M_m^{Spec} \stackrel{a_1}{\longrightarrow} q_1 \stackrel{a_2}{\longrightarrow} q_2 \ldots q_{m-1} \stackrel{a_m}{\longrightarrow} q_m$, with $q_1, \ldots, q_m$ states of M_m, in which $J = \{j_1, \ldots, j_r\} \subseteq \{1, \ldots, m\}$ is the set of indexes such that $q_j \in \mathcal{Q}$ if and only if $j \in J$. 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 $1 \leq j \leq r$ we have that $\sigma'_j \sim \sigma_j$ and $\sigma = \sigma'_1 \cdots \sigma'_r$.

   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 remark 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. Let us note that we are just using the specification as an *oracle* in a similar way to what is usually done in model-based testing where pass/fail states are assigned to test cases depending on whether the sequence of actions reaching those states is expected or not.

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 [7]. 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 12 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 $s||t_l \stackrel{\sigma_1 ?i_1}{\Longrightarrow}$ and $s||t_l \stackrel{\sigma_2 ?i_2}{\Longrightarrow}$. □

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_U, t_L) is deterministic for M_m^{Spec} but 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 [6], and is a necessary and sufficient condition under which we avoid this form of nondeterminism. This essentially corresponds to the testers not taking the opportunity to synchronise in marked states and so we use the term strongly controllable.

Definition 13 A (local or global) test case t is *strongly controllable* for $M_m \in m\mathcal{IOTS}(I,O)$ if there do not exist port $p \in \mathcal{P}$, $\sigma_1, \sigma_2 \in \mathcal{T}r(s,t)$ 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 14 A (local or global) test case t is *weakly controllable* for $M_m \in m\mathcal{IOTS}(I,O)$ if there do not exist port $p \in \mathcal{P}$ 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 $1 \leq j \leq r$, $\pi_p(\sigma_{r+1}) = \pi_p(\sigma_{r+1})$, $q_{r+1} \stackrel{?i_p}{\longrightarrow}$ and there is no q' such that $q'_{r+1} \stackrel{?i_p}{\longrightarrow} q'$. 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.

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 [6]. A similar result holds in the current framework, but with the advantage that scenarios reduce the set of traces that can occur.

Proposition 3 Let $M_m^{SUT}, M_m^{Spec} \in m\mathcal{IOTS}(I,O)$ and t be a weakly controllable local test case for 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. The next result answers the question of whether we can always implement a controllable global test case using a weakly controllable local test case.

Proposition 4 Let $\mathcal{P} = \{1, \ldots, n\}$ be the set of ports and $M_m \in m\mathcal{IOTS}(I, O)$. If t is a 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.

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. 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^{Spec} 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,
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. □

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 15 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 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 we have that M_m^{SUT} **passes** t_l for the scenarios given by M_m^{Spec}. □

In [6] we showed that M_m^{SUT} **dioco** M_m^{Spec} implies M_m^{SUT} **c-dioco** M_m^{Spec}, while the reverse implication does not hold in general. We have a similar result for our new implementation relations (the proof is also similar).

Proposition 5 Let $M_m^{SUT}, M_m^{Spec} \in \mathcal{IOTS}(I, O)$. 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. □

4 Conclusions

This paper represents a continuation of our previous work on formal testing of systems with distributed ports.

We have introduced a new formalism that allows 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. This intuition has been reflected in a new implementation relation that represents a suitable extension 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 analyzed how the new conditions affect the notion of controllability.

There are several possible areas of future work. First, we have to provide an algorithm to decide whether a test case is controllable. We plan to adapt the algorithm presented in [6], working in low order polynomial time, to the new framework. Second, 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 given in [8] in the context of **dioco** and **c-dioco**. Finally, we would like to take into account some variants that were sketched in this paper but not fully exploited. We would like to study the effect of using a *for all* approach in the definition of our new implementation relation. In addition, an interesting alternative to marking states in the specification is to mark states in local testers extracted from the specification and *forget* the marked states of the specification.

References

[1] P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, 2008.

[2] E. Brinksma, L. Heerink, and J. Tretmans. Factorized test generation for multi-input/output transition systems. In *11th IFIP Workshop on Testing of Communicating Systems, IWTCS'98*, pages 67–82. Kluwer Academic Publishers, 1998.

[3] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors. *Model-based Testing of Reactive Systems, LNCS 3472*. Springer, 2005.

[4] R.M. Hierons, K. Bogdanov, J.P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Luettgen, A.J.H Simons, S. Vilkomir, M.R. Woodward, and H. Zedan. Using formal methods to support testing. *ACM Computing Surveys*, 41(2), 2009.

[5] R.M. Hierons, J.P. Bowen, and M. Harman, editors. *Formal Methods and Testing, LNCS 4949*. Springer, 2008.

[6] R.M. Hierons, M.G. Merayo, and M. Núñez. Controllable test cases for the distributed test architecture. In *6th Int. Symposium on Automated Technology for Verification and Analysis, ATVA'08, LNCS 5311*, pages 201–215. Springer, 2008.

[7] R.M. Hierons, M.G. Merayo, and M. Núñez. Implementation relations for the distributed test architecture. In *Joint 20th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'08, and 8th Int. Workshop on Formal Approaches to Software Testing, FATES'08, LNCS 5047*, pages 200–215. Springer, 2008.

[8] R.M. Hierons, M.G. Merayo, and M. Núñez. Implementation relations and test generation for systems with distributed interfaces. Submitted, 2010.

[9] Joint Technical Committee ISO/IEC JTC 1. *International Standard ISO/IEC 9646-1. Information Technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts*. ISO/IEC, 1994.

[10] G. Luo, R. Dssouli, and G. von Bochmann. Generating synchronizable test sequences based on finite state machine with distributed ports. In *6th IFIP Workshop on Protocol Test Systems, IWPTS'93*, pages 139–153. North-Holland, 1993.

[11] G.J. Myers. *The Art of Software Testing*. John Wiley and Sons, 2nd edition, 2004.

[12] O. Rafiq and L. Cacciari. Coordination algorithm for distributed testing. *The Journal of Supercomputing*, 24(2):203–211, 2003.

[13] B. Sarikaya and G. von Bochmann. Synchronization and specification issues in protocol testing. *IEEE Transactions on Communications*, 32:389–395, 1984.

[14] K.-C. Tai and Y.-C. Young. Synchronizable test sequences of finite state machines. *Computer Networks and ISDN Systems*, 30(12):1111–1134, 1998.

[15] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools*, 17(3):103–120, 1996.

[16] H. Ural and C. Williams. Constructing checking sequences for distributed testing. *Formal Aspects of Computing*, 18(1):84–101, 2006.

[17] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan-Kaufmann, 2007.