Towards Automatic Distribution of Testers for Distributed Conformance Testing

Claude Jard, Thierry Jéron, Hakim Kahlouche & César Viho
IRISA - Campus de Beaulieu, F35042 Rennes, France
e-mail: {jard, jeron, kahlouch, viho}@irisa.fr

Abstract

This paper presents first steps towards automatic generation of distributed tests. We first define a characterization of the tests for which the property of unbias is preserved by the existence of an asynchronous environment. Then, starting from a centralized test case, we propose a method to derive automatically its corresponding distributed test case in an asynchronous environment. We prove that the generated distributed test case is not biased, it tests the same behaviors of an implementation and has the same testing power as the centralized test case.

Keywords

Conformance Test Generation, Unbiased Test Case, Distributed Testing

1 INTRODUCTION

In the context of black box conformance testing, an implementation under test (IUT) is tested in order to obtain the conviction that its behavior conforms with its specification. The testers stimulate the IUT by sending messages on points of control and observation (PCOs) and observe on these same PCOs the reactions of the IUT. Within sight of the reactions, a verdict (Fail, Pass or Inconclusive) is emitted. More recently, the conformance testing methodology has been formalized [1, 2, 3] and brought more confidence in the testing activity. Most of works in this field using formal methods concentrated on the production of a centralized single tester and make the implicite assumption of synchronicity between the tester and the IUT [4, 5]. In practice however, one cannot always avoid taking into account the test environment intercalated between the tester and the IUT. The most frequent example is that of remote testing architecture in which the tester reaches the IUT through a network connection. In this case, PCOs can be seen as composed of two FIFO queues, for each direction of the interaction: one speaks then about asynchronous interaction between the tester and the IUT. In the prolongation of this concept, we propose to aim at a test architecture in which several testers coexist and interact in an asynchronous way not only with the IUT but also between them to coordinate the testing activity. It is about a generalization of the multi-party testing technique which benefits from the various possibilities offered by the introduction of concurrency into TTCN [6]. By considering that the IUT itself could be distributed, we are in the general context of distributed testing of a distributed architecture, illustrated by figure 1. To face up to the problem of the design of such testers, we distinguish several subproblems.

The first main problem is that of the asynchronism on PCOs. The possibility of disorder on the observations collected on different PCOs as well as the possible collision on a PCO between a stimulus and an observation makes difficult the design of testers. During the examination of existing test suites, one realizes that these are the main reasons of non-validity of some tests: precisely a correct test not rejecting conformant implementations in a synchronous environment can be biased (i.e. it rejects a conformant implementation) when it is carried out in an asynchronous environment [7]. It is thus significant to know if a test is correct in an asynchronous environment. We will show that contrary to what is suggested in the ISO/9646 methodology [6, 8] with the concept of generic abstract test case, independent of the testing environment, this question depends on the specification of the system to be tested.

The second main problem is to design the distributed tester (a system of parallel testers) and their coordination procedure. We propose to start from a centralized test case and automatically to produce by compilation the communicating testers. Insofar as the initial centralized test case is sequential and describes a set of behaviors to be tested, we try to reproduce in a distributed way, exactly the same set of behaviors, by exploiting some ideas from protocol synthesis [18]. The possible concurrencies in the whole distributed test come from the desynchronization between the IUT and the testers and between local actions of the testers. To extract the potential concurrency of the initial test case is a prospect for our approach. This should impact the generation phase of test cases from formal specifications (as advocated in [15] using event structures), or demand some causal information at runtime (as suggested by [16, 17]).

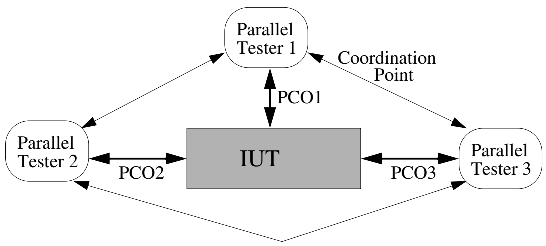

Figure 1 Distributed testing in an asynchronous environment

In order to found our approach mathematically, we use the standard model of input-output transition systems to represent all the objects (testers, specifications and IUTs) involved in the conformance testing. The notations and models used in this paper are described in the section 2. In section 3, we precisely state the condition under which the property of unbias of the test is preserved by the existence of an asynchronous environment. We show that it can be computed starting from the specifications of the IUT and the tester. Then we propose, in section 4, a compilation rule which produces the distributed tester starting from an initial centralized test case. The difficult question is the resolution of distributed choices. We show that under the assumption of an underlying service of election, one obtains a system whose traces reduced to the events on PCOs are the same ones as the traces defined by the centralized test case. A proof of the correctness of the distributed test case is given followed by an example of distribution. Concluding remarks and future works are presented in section 5.

2 MODELS FOR CONFORMANCE TESTING

In this section we describe the models used for the description of the different objects involved in the generation of test cases.

2.1 Input-Output labelled transition systems

The models used are all based on Input-Output Labelled Transition Systems (IOLTS) in which input and output actions on input and output ports are differentiated because of the asymmetrical nature of the testing activity.

Definition 21 *An IOLTS is a tuple* M=$(Q^M, P_I^M, P_O^M, A_I^M, A_O^M, T^M, q_{init}^M)$ *where* Q^M *is a set of states,* $q_{init}^M \in Q^M$ *is the initial state,* P_I^M *and* P_O^M *are finite sets of input and output ports,* A_I^M *and* A_O^M *respectively are finite input and output alphabets,* $A^M = P_I^M \times \{?\} \times A_I^M \cup P_O^M \times \{!\} \times A_O^M$ *is the alphabet of observable actions constructed from the sets of input-output ports and input-output alphabets,* $\tau \notin A^M$ *denotes an internal action,* $T^M \subseteq Q^M \times A^M \cup \{\tau\} \times Q^M$ *is the transition relation. We note* $q \xrightarrow{\alpha}_M q'$ *for* $(q, \alpha, q') \in T^M$.

An IOLTS is a particular Labelled Transition System (LTS). Consequently we adopt the following standard notations of LTS.

Let $\alpha_i \in A^{\mathrm{M}}$, $\mu_i \in A^{\mathrm{M}} \cup \{\tau\}$, $\sigma \in (A^{\mathrm{M}})^*$, $q, q', q_i \in Q^{\mathrm{M}}$,

- $q \xrightarrow{\mu_1 \cdots \mu_n} q' =_\Delta \exists q_0 = q, q_1 \ldots, q_n = q', \forall i \in [1, n], q_{i-1} \xrightarrow{\mu_i} q_i$,
- $q \xrightarrow{\mu_1 \cdots \mu_n} =_\Delta \exists q', q \xrightarrow{\mu_1 \cdots \mu_n} q'$ and $q \not\xrightarrow{\mu_1 \cdots \mu_n} q' =_\Delta \textbf{not}(q \xrightarrow{\mu_1 \cdots \mu_n})$,
- $q \overset{\epsilon}{\Rightarrow} q' =_\Delta q = q'$ or $q \xrightarrow{\tau \cdots \tau} q'$ and $q \overset{\alpha}{\Rightarrow} q' =_\Delta \exists q_1, q_2, q \overset{\epsilon}{\Rightarrow} q_1 \xrightarrow{\alpha} q_2 \overset{\epsilon}{\Rightarrow} q'$,
- $q \overset{\alpha_1 \cdots \alpha_n}{\Longrightarrow} q' =_\Delta \exists q_0 = q, q_1 \ldots, q_n = q', \forall i \in [1, n], q_{i-1} \overset{\alpha_i}{\Rightarrow} q_i$,

- $enable(q) =_\Delta \{\alpha \in A^{\mathrm{M}} \mid \exists q' \text{ and } q \overset{\alpha}{\Rightarrow}_{\mathrm{M}} q'\}$ is the set of observable actions possible in q,
- $In(q) =_\Delta \{a \in A_I^{\mathrm{M}} \mid \exists p_{\mathrm{I}} \in P_I^{\mathrm{M}}, p_{\mathrm{I}}?a \in enable(q)\}$ is the set of possible inputs in q and $Out(q) =_\Delta \{a \in A_O^{\mathrm{M}} \mid \exists p_{\mathrm{O}} \in P_O^{\mathrm{M}}, p_{\mathrm{O}}!a \in enable(q)\}$ is the set of possible outputs in q,
- q **after** $\sigma =_\Delta \{q' \in Q^{\mathrm{M}} \mid q \overset{\sigma}{\Rightarrow}_{\mathrm{M}} q'\}$ is the set of reachable states from q by the sequence of observable actions σ,
- $traces(q) =_\Delta \{\sigma \in (A^{\mathrm{M}})^* \mid q \textbf{ after } \sigma \neq \emptyset\}$
- if $\alpha \in A^{\mathrm{M}}$ is an observable action, we note $\overline{\alpha}$ its mirror action defined by $\overline{p!a} = p?a$ and $\overline{p?a} = p!a$. This notation is extended to sequences of actions.

When needed, we will identify a transition system with its initial state.

2.2 Specifications, implementations, test cases and conformance

A **specification** is modelled by an IOLTS $S = (Q^{\mathrm{S}}, P_I^{\mathrm{S}}, P_O^{\mathrm{S}}, A_I^{\mathrm{S}}, A_O^{\mathrm{S}}, T^{\mathrm{S}}, q_{\mathrm{init}}^{\mathrm{S}})$ If we want to model the absence of output in a specification, as in [9], we can model it by adding a loop with output $\delta \in A_O^{\mathrm{S}}$ in each state where no output and no τ transition is possible. Here, for the sake of simplicity, we will not distinguish δ from other outputs.

Though an **implementation** of S is not necessarily a transition system (it may be a physical system), as in all testing theories, we have to reason formally about it and model its behaviour. As it is only considered by its interactions with the environment, it is also modelled by an IOLTS $I = (Q^{\mathrm{I}}, P_I^{\mathrm{I}}, P_O^{\mathrm{I}}, A_I^{\mathrm{I}}, A_O^{\mathrm{I}}, T^{\mathrm{I}}, q_{\mathrm{init}}^{\mathrm{I}})$, with $P_I^{\mathrm{I}} = P_I^{\mathrm{S}}$ and $P_O^{\mathrm{I}} = P_O^{\mathrm{S}}$ and $A_I^{\mathrm{S}} \subseteq A_I^{\mathrm{I}}$, $A_O^{\mathrm{S}} \subseteq A_O^{\mathrm{I}}$. We will suppose that an implementation I can never refuse an input: $\forall \sigma \in (A^{\mathrm{I}})^*, In(I \textbf{ after } \sigma) = A_I^{\mathrm{I}}$ *.

A **test case** is a set of sequences of actions describing all the interactions occurring between an IUT I and a tester which wants to verify that I conforms to its specification S. A test case is an IOLTS $T = (Q^{\mathrm{T}}, P_I^{\mathrm{T}}, P_O^{\mathrm{T}}, A_I^{\mathrm{T}}, A_O^{\mathrm{T}}, T^{\mathrm{T}}, q_{\mathrm{init}}^{\mathrm{T}})$ such that $A_I^{\mathrm{T}} = A_O^{\mathrm{I}}$ i.e. every possible output of the implementation must be considered as an input of the test case, $A_O^{\mathrm{T}} = A_I^{\mathrm{S}}$ i.e. a test case should only send outputs that are waited by the system as described in the specification, $\{\textbf{pass}, \textbf{fail}\} \in Q^{\mathrm{T}}$ with $enable(\textbf{pass}) = enable(\textbf{fail}) = \emptyset$, T is controllable i.e. in each state, either an output is enabled or all inputs are enabled which is formally described by $\forall \sigma \in (A^{\mathrm{T}})^*, In(T \textbf{ after } \sigma) = A_I^{\mathrm{T}}$ or $\emptyset$, the state **fail** is directly accessible only by inputs i.e. $\forall q \in Q^{\mathrm{T}}, \forall \alpha \in A^{\mathrm{T}}, q \overset{\alpha}{\Rightarrow}_T \textbf{fail} \Longrightarrow \exists p \in P_I^{\mathrm{T}}, a \in A_I^{\mathrm{T}}, \alpha = p?a$ *. Notice that we do not suppose that testers are

*In a way identical to specifications, the absence of output is modeled by the output $\delta \in A_O^{\mathrm{I}}$.
*In practice however $A_I^{\mathrm{T}} = A_O^{\mathrm{I}}$ is unknown. Thus only inputs not leading to **fail** can be mentioned, the other inputs are implicitly leading to fail or are mentioned by **? otherwise** with verdict **Fail**, like in TTCN.

deterministic as it would be to strong for distributed testers. Anyway, we only consider the traces of testers.

We consider a **conformance relation** quite similar to those in [10, 11, 9]. Informally, the conformance relation states that after a trace of the specification, outputs produced by the implementation must be foreseen by the specification. This authorizes the fact that outputs of the environment which are not accepted by the specification may be accepted by the implementation.

Let S and I be two IOLTS describing a specification and an implementation, I **ioconf** $S =_{\Delta} \forall \sigma \in traces(S),\ Out(I \textbf{ after } \sigma) \subseteq Out(S \textbf{ after } \sigma)$

Remark: $traces(I) \subseteq traces(I') \Longrightarrow (I' \textbf{ ioconf } S \Longrightarrow I \textbf{ ioconf } S)$.
But $traces(S) \subseteq traces(S') \not\Longleftrightarrow (I \textbf{ ioconf } S \Longrightarrow I \textbf{ ioconf } S')$
and $traces(S) \subseteq traces(S') \not\Longleftrightarrow (I \textbf{ ioconf } S' \Longrightarrow I \textbf{ ioconf } S)$.

3 CHARACTERIZATION OF UNBIASED TEST CASES

Our goal is to distribute test cases without loss of observation power. But before studying this problem we have to define what kind of test cases we want to distribute. So we start by giving a characterization of unbiased test cases with respect to a specification. This characterization supposes a synchronous communication between the tester and the implementation. As we want to distribute test cases for asynchronous architectures, we extend this characterization to testers which communicate asynchronously with implementations.

3.1 Synchronous testing

The synchronous application of a test case to an implementation is defined as a parallel composition $\|_s$ of the test case T and the implementation I.

$$\frac{I \xrightarrow{\tau} I'}{T\|_s I \xrightarrow{\tau} T\|_s I'} \qquad \frac{T \xrightarrow{\tau} T'}{T\|_s I \xrightarrow{\tau} T'\|_s I} \qquad \frac{T \xrightarrow{\alpha} T' \quad I \xrightarrow{\overline{\alpha}} I'}{T\|_s I \xrightarrow{\alpha} T'\|_s I'}$$

We can now define test failure and unbiased test cases with respect to **ioconf** and a given specification:

Definition 31 T **fails** $I \ =_{\Delta}\ \exists I', \exists \sigma, T\|_s I \stackrel{\sigma}{\Rightarrow} \textbf{fail}\|_s I'$
T is unbiased w.r.t. S $=_{\Delta}\ \forall I, T \textbf{ fails } I \Rightarrow not(I \textbf{ ioconf } S)$

Now the following proposition gives a characterization of unbiased test cases w.r.t. a specification in a synchronous environment. It says that after each trace of the test case leading to a state where an input is allowed, each possible output of the specification after the same trace should be taken into account and should not lead to a fail verdict.

Proposition 31 *T is unbiased w.r.t. S* $\Longleftrightarrow$
$\forall \sigma \in traces(T)$ *s.t.* $In(T \textbf{ after } \sigma) \neq \emptyset$, $Out(S \textbf{ after } \sigma) \subseteq In_{\neg fail}(T \textbf{ after } \sigma)$ *with* $In_{\neg fail}(T') = \{a \in In(T') \mid \exists p, T' \stackrel{p?a}{\Rightarrow} T'' \wedge \textbf{fail} \notin T''\}$ [*].

Proof. Let T be a test case such that $\forall \sigma \in traces(T)$ s.t. $In(T \textbf{ after } \sigma) \neq \emptyset$, $Out(S \textbf{ after } \sigma) \subseteq In_{\neg fail}(T \textbf{ after } \sigma)$. We have to prove that T is unbiased w.r.t. S. Suppose that for some I, T **fails** I . By definition, **fail** verdicts are only given on inputs of T (i.e. outputs of I), thus a trace leading to a **fail** verdicts ends with an input of T. T **fails** I can be written as $\exists I_1, I', \exists T_1, \exists \sigma', \exists p \in P_I^{\mathrm{T}}, \exists a \in A_I^{\mathrm{T}}$ s.t. $T \|_s I \stackrel{\sigma_1}{\Rightarrow} T_1 \|_s I_1 \stackrel{p?a}{\Rightarrow} \textbf{fail} \|_s I'$. Thus $a \notin In_{\neg fail}(T \textbf{ after } \sigma_1)$ and as $Out(S \textbf{ after } \sigma_1) \subseteq In_{\neg fail}(T \textbf{ after } \sigma_1)$, we also have $a \notin Out(S \textbf{ after } \sigma_1)$. But we have $I \stackrel{\sigma_1}{\Rightarrow} I_1 \stackrel{\overline{p?a}=p!a}{\Rightarrow} I'$ which implies $a \in Out(I \textbf{ after } \sigma_1)$. Consequently, $\exists \sigma_1 \in traces(S), \exists p, \exists a, a \in Out(I \textbf{ after } \sigma_1) \wedge a \notin Out(S \textbf{ after } \sigma_1)$ which proves that not (I **ioconf** S) and thus T is unbiased w.r.t. S.

Now suppose that $\exists \sigma \in traces(T), In(T \textbf{ after } \sigma) \neq \emptyset$ and $Out(S \textbf{ after } \sigma) \not\subseteq In_{\neg fail}(T \textbf{ after } \sigma)$. Then $\exists a \in A_I^{\mathrm{T}}, \exists p$, $(S \textbf{ after } \sigma \stackrel{p!a}{\Rightarrow}) \wedge (T \textbf{ after } \sigma \stackrel{p?a}{\Rightarrow} \textbf{fail})$. Let $I = \sigma.p!a$. Clearly we have I **ioconf** S. But as $T \|_s I \stackrel{\sigma.p?a}{\Rightarrow} \textbf{fail} \|_s \emptyset$ we get T **fails** I and T is biased w.r.t. S. $\square$

Remark: Proposition 31 exactly characterizes unbiased test cases with respect to a specification. This is defined for a particular conformance relation **ioconf** and with some hypothesis on the structure of test cases. But **ioconf** seems to be exactly what manual test developers have in mind and the hypothesis made on test cases are weaker than those made on manual test cases.

In the context of automatic generation of test cases it is important to ensure that test cases are unbiased and this kind of characterization should be verified by tools. Notice that the prototype tool TGV [3] ensures this by construction.

3.2 Asynchronous testing

Before extending the preceding characterization to asynchronous testing, we have to define what is asynchronous testing. It is defined from synchronous testing by considering a transformation of the specifications and the implementations by a queue context like in [12].

Let $M = (Q^{\mathrm{M}}, P_I^{\mathrm{M}}, P_O^{\mathrm{M}}, A_I^{\mathrm{M}}, A_O^{\mathrm{M}}, T^{\mathrm{M}}, q_{\mathrm{init}}^{\mathrm{M}})$ be an IOLTS. We define an IOLTS $\mathcal{A}(M)$ which models the behavior of M in an asynchronous environment.
$\mathcal{A}(M) = (Q^{\mathrm{A(M)}}, P_I^{\mathrm{A(M)}}, P_O^{\mathrm{A(M)}}, A_I^{\mathrm{A(M)}}, A_O^{\mathrm{A(M)}}, T^{\mathrm{A(M)}}, q_{\mathrm{init}}^{\mathrm{A(M)}})$ with

- $Q^{\mathrm{A(M)}} = Q^{\mathrm{M}} \times \prod_{p \in P_I^{\mathrm{M}}} A_I^{\mathrm{M}*} \times \prod_{p \in P_O^{\mathrm{M}}} A_O^{\mathrm{M}*}$ and $q_{\mathrm{init}}^{\mathrm{A(M)}} = < M, (\epsilon \cdots \epsilon), (\epsilon \cdots \epsilon) >$

---
[*] Recall that $In(T') = A_I^{\mathrm{T}}$ or $\emptyset$ due to the controllability property.

- $P_I^{\mathrm{A(M)}} = P_I^{\mathrm{A}}$ and $P_O^{\mathrm{A(M)}} = P_O^{\mathrm{A}}$, $A_I^{\mathrm{A(M)}} = A_I^{\mathrm{A}}$ and $A_O^{\mathrm{A(M)}} = A_O^{\mathrm{A}}$.
- $T^{\mathrm{A(M)}}$ is described by the following operational rules defined for all $q, q' \in Q^{\mathrm{M}}, a \in A_I^{\mathrm{M}}, b \in A_O^{\mathrm{M}}, p_{\mathrm{I}k} \in P_I^{\mathrm{M}}, p_{\mathrm{O}l} \in P_O^{\mathrm{M}}$:

R1 $< q, (p_{\mathrm{I}1} \cdots p_{\mathrm{I}k} = w \cdots), (p_{\mathrm{O}1} \cdots) > \stackrel{p_{\mathrm{I}k}?a}{\rightarrow}_{\mathrm{A(M)}} < q, (p_{\mathrm{I}1} \cdots p_{\mathrm{I}k}' = w.a \cdots), (p_{\mathrm{O}1} \cdots) >$

R2 $< q, (p_{\mathrm{I}1} \cdots), (p_{\mathrm{O}1} \cdots p_{\mathrm{O}l} = b.w \cdots) > \stackrel{p_{\mathrm{O}l}!b}{\rightarrow}_{\mathrm{A(M)}} < q, (p_{\mathrm{I}1} \cdots), (p_{\mathrm{O}1} \cdots p_{\mathrm{O}l} = w \cdots) >$

R3 $$\frac{q \stackrel{\tau}{\rightarrow}_{\mathrm{M}} q'}{< q, (p_{I1} \cdots), (p_{O1} \cdots) > \stackrel{\tau}{\rightarrow}_{\mathrm{A(M)}} < q', (p_{\mathrm{I}1} \cdots), (p_{\mathrm{O}1} \cdots) >}$$

R4 $$\frac{q \stackrel{p_{\mathrm{I}k}?a}{\rightarrow}_{\mathrm{M}} q'}{< q, (p_{\mathrm{I}1} \cdots p_{\mathrm{I}k} = a.w \cdots), (p_{\mathrm{O}1} \cdots) > \stackrel{\tau}{\rightarrow}_{\mathrm{A(M)}} < q', (p_{\mathrm{I}1} \cdots p_{\mathrm{I}k} = w \cdots), (p_{\mathrm{O}1} \cdots) >}$$

R5 $$\frac{q \stackrel{p_{\mathrm{O}l}!b}{\rightarrow}_{\mathrm{M}} q'}{< q, (p_{\mathrm{I}1} \cdots), (p_{\mathrm{O}1} \cdots p_{\mathrm{O}l} = w \cdots) > \stackrel{\tau}{\rightarrow}_{\mathrm{A(M)}} < q', (p_{\mathrm{I}1} \cdots), (p_{\mathrm{O}1} \cdots p_{\mathrm{O}l} = w.b \cdots) >}$$

Remark: The first rule allows the IOLTS $\mathcal{A}(M)$ to receive any input in any state. This leads to an infinite states IOLTS. When this transformation is used to model the behaviour of a specification in an asynchronous environment in order to derive test cases, this of course poses algorithmic problems. Thus it is reasonable to limit the behaviour of the specification in order to derive test cases. An example of limitation consists in allowing the environment to send only messages that are possibly waited and to limit the sum of lengths of input queues to 1. In this case **R1** could then be replaced by:

R1' $$\frac{q \stackrel{p_{Ik}?a}{\rightarrow}_{\mathrm{M}}}{< q, (\epsilon \cdots \epsilon \cdots), (p_{O1} \cdots) > \stackrel{p_k?a}{\rightarrow}_{\mathrm{A(M)}} < q, (\epsilon \cdots p_{Ik} = a \cdots), (p_{O1} \cdots) >}$$

As in [13], we define the conformance of an implementation with respect to a specification in an asynchronous environment as the conformance of the implementation in its asynchronous environment with respect to the specification in the same asynchronous environment. This directly leads to the definition of test failure and unbias in an asynchronous environment and gives a corollary of **Proposition 31**:

Definition 32 *Let I be an implementation of a specification S, T a test case.*
I **ioconf**$_{\mathcal{A}}$ $S =_{\Delta} \mathcal{A}(I)$ **ioconf** $\mathcal{A}(S)$
T **fails**$_{\mathcal{A}}$ $I =_{\Delta} \exists \mathcal{A}', \exists \sigma, T \|_s \mathcal{A}(I) \stackrel{\sigma}{\Rightarrow}$ **fail** $\|_s \mathcal{A}'$. *Thus* T **fails**$_{\mathcal{A}}$ $I = T$ **fails** $\mathcal{A}(I)$.
T *is* **unbiased** *w.r.t* S *in an asynchronous environment* $=_{\Delta} \forall I, T$ **fails**$_{\mathcal{A}}$ $I \Rightarrow$ *not*(I **ioconf**$_{\mathcal{A}}$ S).
It is equivalent to say that test case T is unbiased with respect to $\mathcal{A}(S)$.

Corollary 31 *T is unbiased w.r.t. S in an asynchronous environment* $\Longleftrightarrow$ $\forall \sigma \in traces(T)$ *s.t.* $In(T$ **after** $\sigma) \neq \emptyset$, $Out(\mathcal{A}(S)$ **after** $\sigma) \subseteq In_{\neg fail}(T$ **after** $\sigma)$ *with* $In_{\neg fail}(T') = \{a \in In(T) | \exists p, T' \stackrel{p?a}{\rightarrow} T'' \wedge T'' \neq$ **fail**$\}$.

This corollary characterizes a class of test cases that are unbiased in an asynchronous environment. It is also easy to prove that an unbiased test case in an asynchronous environment is also unbiased in a synchronous environment. This uses the fact that $traces(M) \subseteq traces(\mathcal{A}(M))$ which relies on construction rules of $\mathcal{A}(M)$ and the property that if $traces(S) \subseteq traces(S')$ then, if T is unbiased w.r.t. S' then T is also unbiased w.r.t. S. The converse is of course false as was already noticed in [13].

4 AUTOMATIC DISTRIBUTION OF TEST CASES

In this section we develop the rule used to automatically distribute a centralized tester. Then, we give the correctness proof of that rule wich implies that the unbiased criterion is preserved by the distribution rule and that the distributed tester rejects the same IUTs as the original centralized tester. Some additional notations are needed. Let T be an IOLTS, $q \in Q^{\mathrm{T}}$ and $\alpha \in A^{\mathrm{T}}$:

- $P^{\mathrm{T}} = P_I^{\mathrm{T}} \cup P_O^{\mathrm{T}}$
- if $\alpha = p?a$ or $\alpha = p!a$, where $p \in P^{\mathrm{T}}$, then $port(\alpha) =_\Delta p$.
- $p\_enable(q) =_\Delta \{p \in P^{\mathrm{T}} \mid \exists \alpha \in enable(q) \quad port(\alpha) = p\}$
- if $\alpha = p?a$ then $present(\alpha)$ is a predicate which is true if the label a is present in the input queue associated to the port p, and it is false otherwise. If $\alpha = p!a$ then $present(\alpha)$ is true.
- The asynchronous parallel composition of $T_1, \cdots, T_n$ is denoted by $\|_{as} T_i = \|_s \mathcal{A}(T_i)$ where an output port of a tester may be unified with an input port of another tester to make the coordination possible. We also note $DT = \|_{as} T_i$ and Q^{DT} the set of global states of $\|_{as} T_i$. A global state $Q \in Q^{\mathrm{DT}}$ is a tuple $(q_1, \cdots, q_n, f_1, \cdots, f_n)$, where q_i is the state of T_i and f_i is the state of the input and output queues of T_i.

Starting from a test case T modellized by an IOLTS, the problem is to synthesize a set of asynchronously communicating IOLTSs $\{T_1, \cdots, T_n\}$ where n is the number of PCOs of T. The initial state of each tester T_i is q_{init}. For sake of clarity, we have considered that each tester is associated with each PCO. The extension to a general PCOs mapping is straightforward. In the following we consider that only the actions of A^{T} are observable.

4.1 Distribution rule

We present here bellow the rule used to automatically distribute a centralized tester. The result is a set of communicating testers. Starting from a pattern of the centralized tester defined by a state q with one or several outgoing transitions, the rule [R1] synthesizes a set of transitions of the distributed tester. These transitions can be decomposed into three parts. The part (A) defines the transitions of the testers T_i associated to the ports involved in the

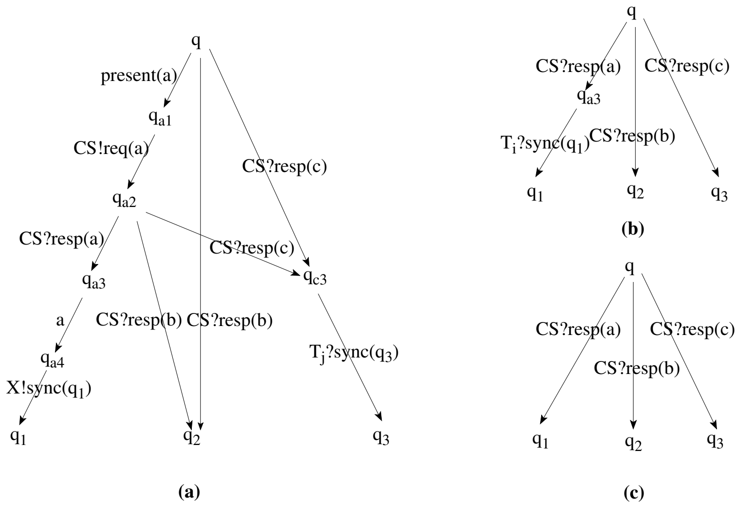

Figure 2 Distribution rule (a) Tester $T_i \mid \; i \in p\_enable(q)$ (b) Tester $T_k \mid k \notin p\_enable(q),\ k \in X$ (c) Tester $T_k \mid \; k \notin p\_enable(q) \cup X$

outgoing transitions of q, namely $p\_enable(q)$ (see Figure 2(a)). The part (B) defines the testers T_k which are not associated to the ports of $p\_enable(q)$ and must receive a synchronization message from T_i (see Figure 2(b)). The part (C) concerns the testers which are not associated to the ports of $p\_enable(q)$ and do not require a synchronization message from T_i (see Figure 2(c)).

$$[\mathrm{R1}]\ \frac{q \in Q^{\mathrm{T}}}{\begin{array}{l}(A)\left\{\begin{array}{l}\forall i \in p\_enable(q) \quad \forall a \in enable(q) \mid port(a) = i,\ q \xrightarrow{a}_{\mathrm{T}} q_1 : \\ q \xrightarrow{present(a)}_{\mathrm{T}_i} q_{a1} \xrightarrow{CS!req(a)}_{\mathrm{T}_i} q_{a2} \xrightarrow{CS?resp(a)}_{\mathrm{T}_i} q_{a3} \xrightarrow{a}_{\mathrm{T}_i} q_{a4} \xrightarrow{X!sync(q_1)}_{\mathrm{T}_i} q_1 \\ \forall b \in enable(q) \mid port(b) \neq i,\ q \xrightarrow{b}_{\mathrm{T}} q_2,\ i \notin p\_enable(q_2) : \\ q_{a2} \xrightarrow{CS?resp(b)}_{\mathrm{T}_i} q_2 \quad q \xrightarrow{CS?resp(b)}_{\mathrm{T}_i} q_2 \\ \forall c \in enable(q) \mid port(c) = j,\ j \neq i,\ q \xrightarrow{c}_{\mathrm{T}} q_3,\ i \in p\_enable(q_3) : \\ q_{a2} \xrightarrow{CS?resp(c)}_{\mathrm{T}_i} q_{c3} \quad q \xrightarrow{CS?resp(b)}_{\mathrm{T}_i} q_{c3} \quad q_{c3} \xrightarrow{T_j?sync(q_3)}_{\mathrm{T}_i} q_3 \end{array}\right. \\ (B)\ \left\{ \forall k \notin p\_enable(q),\ k \in X : \quad q \xrightarrow{CS?resp(a)}_{\mathrm{T}_k} q_{a3} \xrightarrow{T_i?sync(q_1)}_{\mathrm{T}_k} q_1 \right. \\ (C)\ \left\{ \forall k \notin p\_enable(q),\ k \notin X : \quad q \xrightarrow{CS?resp(a)}_{\mathrm{T}_k} q_1 \right. \end{array}}$$

where $X = \{T_j \mid j \neq i,\ j \in p\_enable(q_1)\}$.

A **mapping function** F is introduced between the states of the distributed tester and the states of the centralized tester T. The states provided by F are called *observable states*: $F : \bigcup_{i=1}^{n} Q^{\mathrm{T}_i} \longrightarrow Q^{\mathrm{T}}$. According to the rule [R1]: $F(q_{a1}) = F(q_{a2}) = F(q_{a3}) = F(q_{c3}) = F(q) = q \quad and \quad F(q_{a4}) = F(q_1) = q_1$ This function ensures that the observable state of a tester should change

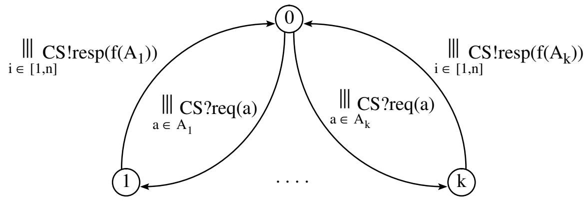

Figure 3 Election Service

only when the tester performs an observable action or when it receives a synchronization message. This function is introduced for correctness proof.

The testers use an **election service** when they attempt to perform an observable action (action of A^{T}). The service accepts at least one request from the testers and delivers to all the testers the observable action which has been selected. The interaction of each tester with the election service is synchronous and it is done through a coordination port named CS. An abstract specification is given in Figure 3. The transition labelled by $|||_{a \in A_i} CS?req(a)$ specifies all possible interleaving of the requests coming from the testers and concerning only with the actions of A_i (where $A_1, \cdots, A_k$ are all possible subsets of A^{T}). This transition specifies an hypercube. The transition labelled by $|||_{i \in [1,n]} CS!resp(a)$ is another hypercube which specifies the broadcasting of the selected* action among those of A_i.

4.2 Correctness proof

We present in this section the correctness proof of the distribution rule. Our purpose if to prove that the asynchronous parallel composition of the synthezised testers (i.e, the distributed tester) is trace-equivalent to the centralized tester. Under the assumption of an election service and reliable and FIFO communication channels between the testers, we start by proving two lemmas. The first one (Lemma 41) stipulates that when two successive observable actions occur in the distributed tester, the observable state in which the second action is done is the same as the one reached by the first action. The observable state of a tester is provided by the mapping function F. This lemma implies that every trace of the distributed tester is a trace of the centralized tester. The second lemma (Lemma 42) states that if the centralized tester can do an observable action in a given observable state, the distributed tester could necessarily do the same action (possibly surrounded by one or several τ-actions). This lemma implies that every trace of the centralized tester is a trace of the distributed tester. Using these two lemmas, we prove the trace-equivalence (Theorem 41).

*For sake of clarity of Figure 3, the selection is done using a function f.

Lemma 41 *$\forall Q, U, V, W \in Q^{DT}$, $\forall a, b \in A^{T}$ such that $Q \xrightarrow{a}_{DT} U \xrightarrow{\tau^*}_{DT} V \xrightarrow{b}_{DT} W$, $\exists q, x, w \in Q^{T}$ such that: $q = F(q_i) \xrightarrow{a}_{T} x = F(u_i) = F(v_i) \xrightarrow{b}_{T} w = F(w_j)$, where $i = port(a)$, $j = port(b)$, q_i and u_i are states of T_i resp. in Q and U, v_j and w_j are states of T_j resp. in V and W, and F is defined in Section 4.1.*

Proof. Let $Q, U, V, W \in Q^{\mathrm{DT}}$, let $a, b \in A^{\mathrm{T}}$ such that:

$$Q \xrightarrow{a}_{\mathrm{DT}} U \xrightarrow{\tau^*}_{\mathrm{DT}} V \xrightarrow{b}_{\mathrm{DT}} W \tag{1}$$

$Q \xrightarrow{a}_{\mathrm{DT}} U \ \Rightarrow \exists T_i \mid i = port(a)$ and $q_i \xrightarrow{a}_{\mathrm{T}_i} u_i \ \Rightarrow F(q_i) \xrightarrow{a}_{\mathrm{T}} F(u_i)$
$V \xrightarrow{b}_{\mathrm{DT}} W \ \Rightarrow \exists T_j \mid j = port(b)$ and $v_j \xrightarrow{b}_{\mathrm{T}_j} w_j \ \Rightarrow F(v_j) \xrightarrow{b}_{\mathrm{T}} F(w_j)$
We should prove a first property (noted P1): $\exists q, x, w \in Q^{\mathrm{T}}$ such that $q \xrightarrow{a}_{\mathrm{T}} x \xrightarrow{b}_{\mathrm{T}} w$, and then a second property (P2): $F(u_i) = F(v_j)$.
From the sequence (1), we know that $\exists q, x \in Q^{\mathrm{T}}$ such that: $q \xrightarrow{a}_{\mathrm{T}} x$. Suppose that: $\forall q, x \in Q^{\mathrm{T}}$ such that $q \xrightarrow{a}_{\mathrm{T}} x$, $\forall \alpha \in enable(x)$, $\alpha \neq b$. The rule [R1] guarantees that in the distributed tester after the observable action a we cannot have b, and this is in contradiction with (1). Therefore, (P1) is necessarily true. To prove (P2) we distinguish 2 cases:
Case 1 ($i \neq j$): From the sequence (1), we can extract the following sequences of the testers T_i and T_j:

$$q_i \xrightarrow{a}_{\mathrm{T}_i} u_i \xrightarrow{\tau^*}_{\mathrm{T}_i} v_i \tag{2}$$

$$u_j \xrightarrow{\tau^*}_{\mathrm{T}_j} v_j \xrightarrow{b}_{\mathrm{T}_j} w_j \tag{3}$$

According to rule [R1], and using (P1), among the τ-actions of (3), there exists necessarily a reception of a synchronisation message ($sync(x)$). So, (3) can be written as follow:

$$u_j \xrightarrow{\tau^*}_{\mathrm{T}_j} u'_j \xrightarrow{\tau}_{\mathrm{T}_j} u''_j \tag{4}$$

$$u''_j \xrightarrow{\tau^*}_{\mathrm{T}_j} v_j \xrightarrow{b}_{\mathrm{T}_j} w_j \tag{5}$$

The last transition of (4) is the reception of $sync(x)$. Using the mapping function F to that transition, we have: $F(u'_j) \xrightarrow{a}_{\mathrm{T}} F(u''_j)$. By identification with $F(q_i) \xrightarrow{a}_{\mathrm{T}} F(u_i)$, we obtain: $F(u'_j) = F(q_i)$ and $F(u''_j) = F(u_i)$. Since the medium is supposed FIFO, another reception among the τ-actions of (5) would imply another observation after a in (2) which is not true. So, there is no reception of synchronization message among the τ-actions of (5). Furthermore, because of the election service, there is no action $CS?resp(x)$ where $x \neq b$. So, (5) contains only the primitives which allow the election of b. From the definition of F, we deduce: $F(u''_j) = F(v_j)$, and consequently: $F(u_i) = F(v_j)$.

Case 2 ($i = j$): From (1), we can extract the following sequence of the tester T_i: $q_i \xrightarrow{a}_{\mathrm{T}_i} u_i \xrightarrow{\tau^*}_{\mathrm{T}_i} v_i \xrightarrow{b}_{\mathrm{T}_i} w_i$. Since $port(a) = port(b) = i$ and according to rule [R1], there is no reception of synchronization message among the τ-actions of that sequence. Furthermore, as in Case 1, the sequence $u_i \xrightarrow{\tau^*}_{\mathrm{T}_i} v_i$ contains only primitives which allow the election of b. Consequently, from the definition of F, we have: $F(u_i) = F(v_j)$ □

Lemma 42 *$\forall q, v \in Q^T$, $\forall a \in A^T$ such that $q \xrightarrow{a}_T v$, $\exists Q, V \in Q^{DT}$ such that: $\exists f_1, \cdots, f_n$ $\quad Q = (q, \cdots, q, f_1, \cdots, f_n) \xrightarrow{\tau^* a \tau^*}_{DT} V = (v, \cdots, v, f_1, \cdots, f_n)$.*

Proof. By applying [R1] to the transition $q \xrightarrow{a}_{\mathrm{T}} v$, we find the following sequences, respectively in T_i (where $i = port(a)$) and in T_j (where $j \neq i$):

$$\begin{cases} q \xrightarrow{\tau^3}_{\mathrm{T}_i} q_{a3} \xrightarrow{a}_{\mathrm{T}_i} q_{a4} \\ q \xrightarrow{\tau^m}_{\mathrm{T}_j} v \quad where \quad m \in [1,2] \quad and \quad j \neq i \end{cases}$$

Using the above sequences we can find the following sequence in DT:
$Q = (q, \cdots, q, f_1, \cdots, f_n) \xrightarrow{\tau^3}_{\mathrm{DT}} (q, \cdots, q_{a3}, \cdots, q, f_1, \cdots, f_n) \xrightarrow{a}_{\mathrm{DT}} Q'$,
$Q' = (q, \cdots, q_{a4}, \cdots, q, f_1, \cdots, f_n) \xrightarrow{\tau^*}_{\mathrm{DT}} (q, \cdots, v, \cdots, q, f'_1, \cdots, f'_n) = Q''$,
$Q'' \xrightarrow{\tau^*}_{\mathrm{DT}} (v, \cdots, v, f_1, \cdots, f_n) = V$. The last τ^* sequence contains the receptions of the synchronization messages which have been sent after the action a (because the medium is supposed to be FIFO and reliable). □

Theorem 41 *If a distributed tester DT is synthesized from a centralized tester T using the rule [R1], then: $traces(DT) = traces(T)$.*

Proof. ($\subseteq$) Let $\sigma \in traces(DT)$, σ can be written as follow: $\sigma = \tau^* a_1 \tau^* a_2 \cdots \tau^* a_n \tau^*$ where $a_i \in A^{\mathrm{T}} \quad \forall i \in [1, n]$. From $Q_{init} \xrightarrow{\sigma}_{\mathrm{DT}} Q$ where $Q_{init}, Q \in Q^{\mathrm{DT}}$, $\exists U_0, \cdots, U_{n-1}, Q_1, \cdots, Q_n \in Q^{\mathrm{DT}}$ such that:

$$Q_{init} \xrightarrow{\tau^*} U_0 \xrightarrow{a_1} Q_1 \xrightarrow{\tau^*} U_1 \xrightarrow{a_2} Q_2 \cdots \xrightarrow{a_{n-1}} Q_{n-1} \xrightarrow{\tau^*} U_{n-1} \xrightarrow{a_n} Q_n = Q \quad (6)$$

From (6), $\exists T_{k_1}$ where $k_1 = port(a_1)$ such that: $q_{k_1} = q_{init} \xrightarrow{\tau^*}_{\mathrm{T}_{k_1}} u^0_{k_1} \xrightarrow{a_1}_{\mathrm{T}_{k_1}} q^1_{k_1}$, where $u^0_{k_1}$ and $q^1_{k_1}$ are the states of the tester T_{k_1} respectively in the global states U_0 and Q_1. Using the mapping function F, we get the first transition in T: $F(u^0_{k_1}) = q_{init} \xrightarrow{a_1}_{\mathrm{T}} F(q^1_{k_1})$ (the τ-actions from q_{init} to $u^0_{k_1}$ contains only the primitives which allow the election of a_1 that is why $F(u^0_{k_1}) = q_{init}$). By applying the Lemma 41 to the sequence (6), we obtain:

$$\forall i \in [1, n] \quad F(u^{i-2}_{k_{i-1}}) \xrightarrow{a_{i-1}}_{\mathrm{T}} F(q^{i-1}_{k_{i-1}}) = F(u^{i-1}_{k_i}) \xrightarrow{a_i}_{\mathrm{T}} F(q^i_{k_i})$$

where $k_i = port(a_i)$. This implies that $a_1 a_2 \cdots a_n \in traces(T)$ and consequently, $traces(DT) \subseteq traces(T)$.

($\supseteq$) Let $\sigma \in traces(T)$, $\exists q_1, \cdots, q_n \in Q^{\mathrm{T}}$ $\exists a_1, \cdots, a_n \in A^{\mathrm{T}}$ such that: $\sigma = a_1 \cdots a_n$ *and* $q_{init} \xrightarrow{a_1}_{\mathrm{T}} q_1 \xrightarrow{a_2}_{\mathrm{T}} \cdots \xrightarrow{a_n}_{\mathrm{T}} q_n$. By applying Lemma 42, $\exists Q_1, \cdots, Q_n$ such that:

$$Q_1 = (q_1, \cdots, q_1, f_1^1, \cdots, f_n^1) \overset{\tau^* a_1 \tau^*}{\longrightarrow}_{\mathrm{DT}} \cdots \overset{\tau^* a_n \tau^*}{\longrightarrow}_{\mathrm{DT}} Q_n = (q_n, \cdots, q_n, f_1^n, \cdots, f_n^n)$$

Consequently, $traces(DT) \supseteq traces(T)$ $\square$

4.3 Possible optimizations

(a) Local choices

For sake of clarity, the compilation rule treats in a uniform way the choices in the initial tester by calling an election service. This election synchronizes the whole of the testers. However, there is no reason so that the testers non-concerned with the choice cannot evolve independently. This has two consequences: (1) it is necessary to add a compilation rule to treat the particular case of local choices, (2) to avoid blocking problems related to the existence of a choice between two τ-transitions from a local tester, it should be considered that the testers obtained are minimal (within the meaning of $\tau^* a$ minimization which eliminates all the τ). The additional rule R2 which is a simplification of R1 can be written:

$$[\mathrm{R2}]\ \frac{q_1 \xrightarrow{a}_{\mathrm{T}} q_2 \quad p\_enable(q_1) = \{i\}}{q_1 \xrightarrow{a}_{\mathrm{T}_i} q_{12} \overset{X!sync(q_2)}{\longrightarrow}_{\mathrm{T}_i} q_2, \forall T_j \in X \quad q_1 \overset{T_i?sync(q_2)}{\longrightarrow}_{\mathrm{T}_j} q_2, \forall T_k \notin X \quad q_1 \xrightarrow{\tau}_{\mathrm{T}_k} q_2}$$

where $i = port(a)$ and $X = \{T_j \mid j \neq i \wedge j \in p\_enable(q_2)\}$

(b) Synthesis of the election service

The distributed choices can be transformed into local choices by using the artifice illustrated in figure 4. The observable traces of the tester remain unchanged. The application of rule R2 produces distributed testers running without the general service of election. More precisely, the election is ensured by the circular transfer of control.

4.4 An example of distribution

Let us consider a test case whose model is presented in the left part of the figure 5. This tester decides to emit interaction a towards the IUT then awaits a reaction b or c. If it receives b, it then awaits c. One wishes to distribute this tester on two sites, one having the responsability of interactions a and b, the other of c.

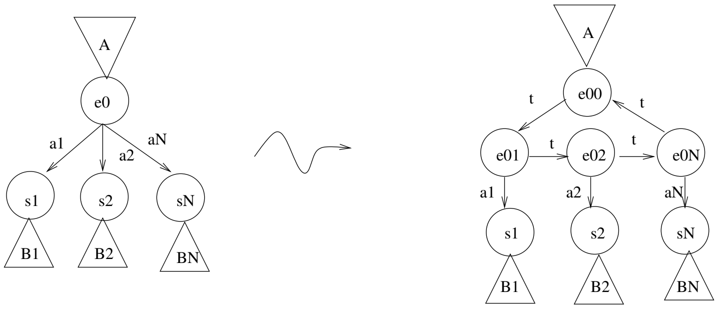

Figure 4 *Preliminary transformation of the test case leading to an integrated election service on a ring of testers where t denotes a τ action.*

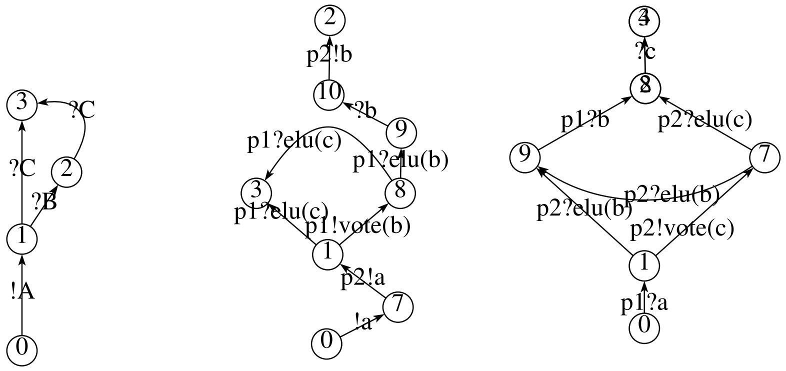

Figure 5 *The initial test case and the result of the synthesis rules.*

The application of the compilation rule R1 produces the two testers introduced in the figure 5: $!a$ and $?b$ are located on tester 1, $?c$ on tester 2. Notice that the synchronizations "P2!a $\rightarrow$ P1?a" and "P2!b $\rightarrow$ P1?b" (through the coordination points P1 and P2) ensure the sequencing of "!A?B?C" and "!A?C", as well as the use of the election service ensures that the testers make the same distributed choice. The figures in Figure 5 and 6 have been obtained using the Aldebaran Toolbox [14].

We consider now that these testers interact asynchronously and use synchronously (locally) the election service whose abstracted form is described in left side of figure 6 (the primitives elu() and vote() correspond respectively to req() and resp() in rule [R1]). The whole behaviors of the system obtained is given by the automaton in the right side of figure 6, which is trace-equivalent to the initial centralized tester.

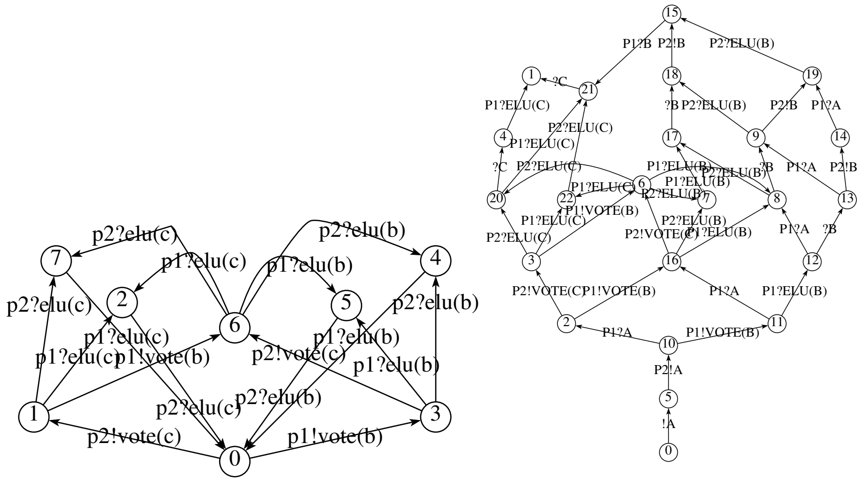

Figure 6 *The abstract spec. of the election service and the result of the composition, which is trace-equivalent to the initial automaton of figure 5*

5 CONCLUSION

In this paper, we gave the first results of our works concerning distributed testing of distributed architecture. We have proposed a method for an automatic distribution of a centralized test case. In this kind of test architecture, several testers coexist and interact in an asynchronous way to coordinate the testing activity. The testers communicate with the IUT, also in an asynchronous way via the PCOs. Under the assumption of an underlying service of election, we generate a distributed test whose traces reduced to the events on PCOs are the same ones as the traces defined by the centralized test case.

As a distributed testing is typically asynchronous, we have to be carefull on the validity of the original test case in an asynchronous environment. In fact, a correct test which does not reject conformant implementations in a synchronous environment can be biased (i.e. it rejects a conformant implementation) when it is carried out in an asynchronous environment. We have also defined a characterization of unbiased test cases preserved by the existence of an asynchronous environment. We show that it can be computed starting from the specifications of the IUT and the tester. Its calculation would be an interesting functionality of a tests generation environment.

As this work is just first steps towards automatic generation of distributed tests, several prospects remain open: (1) investigating a way to generate automatically the service of election as suggested in Section 4.3, (2) the possible concurrencies in distributed testing come from the desynchronization between the IUT and the testers. We will try to find a way to extract the potential concurrency in the test cases and to use it in the distribution, (3) reduction of the number of messages exchanged by the testers for their coordination.

REFERENCES

[1] J. Tretmans. *A formal approach to conformance testing.* PhD thesis, University of Twente, Enschede, The Netherlands, 1992.

[2] M. Phalippou. *Relations d'implantations et Hypothèses de test sur les automates à entrées et sorties.* PhD thesis, Université de Bordeaux, 1994.

[3] J.-C. Fernandez, C. Jard, T. Jéron, and C. Viho. Using on-the-fly verification techniques for the generation of test suites. *Conference on Computer-Aided Verification (CAV '96), USA*, LNCS 1102. Springer, July 1996.

[4] E. Brinksma. A theory for the derivation of tests. *Protocol Specification, Testing and Verification*, North Holland, 1988.

[5] R. Castanet and O. Koné. Deriving Coordinated Testers for interoperability. In *6th Int. Workshop on Protocols Test Systems*, North Holland, 1994.

[6] OSI Open Systems Interconnection. Information Technology - Open Systems Interconnection Conformance Testing Methodology and Framework. *International Standard ISO/IEC 9646-1/2/3*, 1992.

[7] J.-C. Fernandez, C. Jard, T. Jéron, and C. Viho. An experiment in automatic generation of test suites for protocols with verification technology. *Science of Computer Programming - Special Issue on Industrial Relevant Applications of Formal Analysis Techniques*, 1997.

[8] D. Hogrefe and J. Tretmans. Report on the standardization project : Formal methods in conformance testing. *9^{th} International Workshop on Testing of Communicating Systems.* Chapman & Hall, September 1996.

[9] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software - Concepts and Tools*, 17, 1996.

[10] L. Boullier, B. Kelly, M Phalippou, A. Rouger, and N. Webster. Evaluation of some test generation tools on a real protocol example. In *IWPTS'94 : Int. Workshop on Protocols Test Systems*, Tokyo, November 1994.

[11] J. Tretmans. Testing Labelled Transition Systems with Inputs and Outputs. In *8th Int. Workshop on Protocols Test Systems*, Evry, France, Sep 1995.

[12] L. Verhaard and J. Tretmans. A queue model relating synchronous and asynchronous communication. In *Protocol Specification, Testing and Verification, XII*, North Holland, 1992.

[13] L. Verhaard, J. Tretmans, P. Kars, and E. Brinksma. On Asychronous Testing. *5th International Workshop on Protocol Test Systems*, North Holland, 1993.

[14] J.Cl. Fernandez, H. Garavel, L. Mounier, A. Rasse, C. Rodriguez, and J. Sifakis. A Tool Box for the Verification of Lotos Programs. In *14th International Conference on Software Engineering*, Melbourne, Australia, May 1992.

[15] O. Henninger. On test case generation from asynchronously communicating state machines. *Testing of Communicating Systems X.* Chap. & Hall, 1997.

[16] C. Jard. How to observe interoperability at the service level of protocols. *Protocol Test Systems, VII*, pages 259-270, Tokyo, 1994. Chapman & Hall.

[17] M. Kim, S.T. Chanson, and S. Kang. An approach for testing asynchronous communicating systems. *Testing of Communicating Systems, IX*, pages 141–155, Darmstadt, Germany, 1996. Chapmann & Hall.

[18] H. Kahlouche, J.J. Girardot, A Stepwise Refinement Based Approach for Synthesizing Protocol Specifications in an Interpreted Petri Net Model. *Proc. of IEEE INFOCOM'1996*, San Francisco, 1996, pp. 1165-1173.