Distributed Testing of Concurrent Systems: Vector Clocks to the Rescue

Hernán Ponce de León, Stefan Haar, Delphine Longuet

Distributed Testing of Concurrent Systems: Vector Clocks to the Rescue *

Hernán Ponce-de-León[1], Stefan Haar[1], and Delphine Longuet[2]

[1] INRIA and LSV, École Normale Supérieure de Cachan and CNRS, France
ponce@lsv.ens-cachan.fr , stefan.haar@inria.fr
[2] Univ Paris-Sud, LRI UMR8623, Orsay, F-91405
longuet@lri.fr

Abstract. The **ioco** relation has become a standard in model-based conformance testing. The **co-ioco** conformance relation is an extension of this relation to concurrent systems specified with true-concurrency models. This relation assumes a global control and observation of the system under test, which is not usually realistic in the case of physically distributed systems. Such systems can be partially observed at each of their points of control and observation by the sequences of inputs and outputs exchanged with their environment. Unfortunately, in general, global observation cannot be reconstructed from local ones, so global conformance cannot be decided with local tests. We propose to append time stamps to the observable actions of the system under test in order to regain global conformance from local testing.

Model-based Testing. The aim of testing is to execute a software system, the *implementation*, on a set of input data selected so as to find discrepancies between actual behavior and intended behavior described by the *specification*. Model-based testing requires a behavioral description of the system under test. One of the most popular formalisms studied in conformance testing is that of *input output labeled transition systems* (IOLTS). In this framework, the correctness (or conformance) relation the system under test (SUT) and its specification must verify is formalized by the **ioco** relation [1]. This relation has become a standard, and it is used as a basis in several testing theories for extended state-based models: restrictive transition systems [2, 3], symbolic transition systems [4, 5], timed automata [6, 7], multi-port finite state machines [8].

Model-based Testing of Concurrent Systems. Model-based testing of concurrent systems has been studied for a long time [9–11], but mostly in the context of interleaving, or trace, semantics, which is known to suffer from the state space explosion problem. Concurrent systems are naturally modeled as a *network of finite automata*, a formal class of models that can be captured equivalently by *safe Petri nets*. Partial order semantics of a Petri net is given by its *unfolding* [12,

* This work was funded by the DIGITEO / DIM-LSC project TECSTES, convention DIGITEO Number 2011-052D - TECSTES.

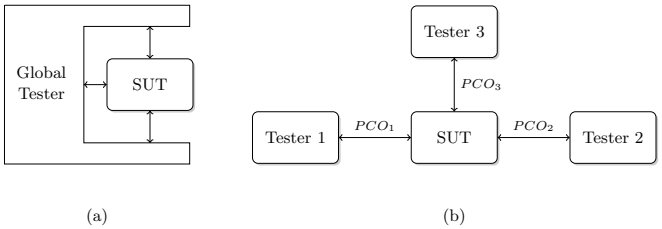

Fig. 1. The global and distributed testing architectures.

13]. Test case generation for concurrent systems based on unfoldings has been studied in [14, 15]. In the same direction, we proposed an extension of the **ioco** conformance relation to concurrent systems, called **co-ioco**, using both interleaving and partial order semantics [16]. We developped a full testing framework for **co-ioco** [17], but in this work, concurrency is only interpreted as independence between actions: actions specified as independent cannot be implemented by interleavings. We introduced a new semantics for unfoldings [18], allowing some concurrency to be implemented by interleavings, while forcing other concurrency to be preserved. The kind of concurrency we consider in this article arises from the distribution of the system, for this reason we restrict to partial order semantics only.

Our previous work [16–18] assumes a global tester which controls and observes the whole system as can be seen in Fig. 1.a. If the system is distributed, the tester interacts with every component, but the observation of such interaction is global.

Distributed Testing. When global observation of the system cannot be achieved, the testing activity needs to be distributed. In a distributed testing environment (see Fig. 1.b), the testers stimulate the implementation by sending messages on points of control and observation (PCOs) and partially observe the reactions of the implementation on these same PCOs. It is known that, in general, global traces cannot be reconstructed from local observations (see for example [19]). This reduces the ability to distinguish different systems. There are three mainly investigated solutions to overcome this problem: *(i)* the conformance relation need to be weaken considering partial observation [8, 20]; *(ii)* testers are allowed to communicate to coordinate the testing activity [21]; *(iii)* stronger assumptions about the implementations are needed, for example that each component have a local clock [22, 23].

Related Work. Hierons et al. [8] argue that when the SUT is to be used in a context in which there are separate testers at the PCO which do not directly communicate with one another, the requirements placed on the SUT do not correspond to traditional implementation relations. In fact, testing the SUT using a method based on a standard implementation relation, such as **ioco**, may return an incorrect verdict. The authors of [8] consider different scenarios, and a dedicated implementation relation for each of them. In the first scenario, there is

a tester at each PCO, and these testers are pairwise independent. In this scenario, it is sufficient that the local behavior observed by a tester be consistent with some global behavior in the specification. This notion is captured by the **p-dioco** conformance relation. In the second scenario, a tester may receive information from other testers, and the local behaviors observed at different PCOs could be combined. Consequently, a stronger implementation relation, called **dioco**, is proposed. They show that **ioco** and **dioco** coincide when the system is composed of a single component, but that **dioco** is weaker than **ioco** when there are several components. Similar to this, Longuet [20] studies different ways of globally and locally testing a distributed system specified with Message Sequence Charts, by defining global and local conformance relations, for which exhaustive test sets are built. Moreover, conditions under which local testing is equivalent to global testing are established under trace semantics.

Jard et al. [21] propose a method for constructing, given a global tester, a set of testers (one for each PCO) such that global conformance can be achieved by these testers. However, they assume that testers can communicate with each other in order to coordinate the testing activity. In addition, they consider the interaction between testers and the SUT as asynchronous.

Bhateja and Mukund [22] propose an approach where they assume each component has a local clock and they append tags to the messages generated by the system under test. These enriched behaviors are then compared against a tagged version of the specification. Hierons et al. [23] make the same assumption about local clocks. If the clocks agree exactly then the sequence of observations can be reconstructed. In practice the local clocks will not agree exactly, but some assumptions regarding how they can differ can be made. They explore several such assumptions and derive corresponding implementation relations.

Contribution. The aim of this paper is to propose a formal framework for the distributed testing of concurrent systems from network of automata specifications, without relying on communications between testers. We show that some, but not all, situations leading to non global conformance wrt **co-ioco** can be detected by local testers without any further information of the other components; moreover we prove that, when vector clocks [24, 25] are used, the information held by each component suffices to reconstruct the global trace of an execution from the partial observations of it at each PCO, and that global conformance can thus be decided by distributed testers.

1 Model of the System

We use *networks of automata* as models for systems, to make concurrency explicit. Networks of automata can be captured equivalently by *Petri nets*.

Network of Automata. We consider a distributed system composed of n components that communicate with each other synchronizing on communication actions. The local model of a component is defined as a deterministic finite automaton (Q, Σ, Δ, q_0), where Q is a finite set of states, Σ is a finite set of actions,

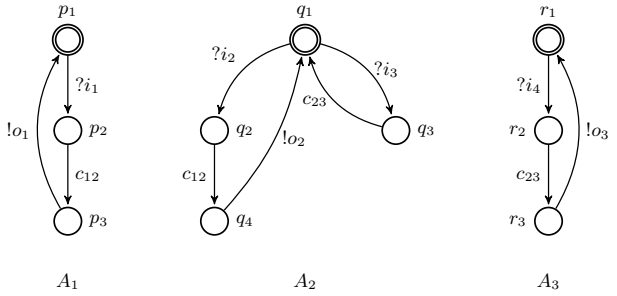

Fig. 2. Network of automata composed of 3 components.

$\Delta : Q \times \Sigma \to Q$ is the transition function and $q_0 \in Q$ is the initial state. We distinguish between the controllable actions $\Sigma_{\mathcal{I}n}$ (inputs proposed by the environment), the observable ones $\Sigma_{\mathcal{O}ut}$ (outputs produced by the system), and communication actions $\Sigma_{\mathcal{C}}$ (invisible for the environment), i.e. $\Sigma = \Sigma_{\mathcal{I}n} \uplus \Sigma_{\mathcal{O}ut} \uplus \Sigma_{\mathcal{C}}$. Several components can communicate over the same communication action, but we assume that observable actions from different components are disjoint[3], i.e. components only share communication actions.

Example 1. Fig. 2 shows a network of automata with three components A_1, A_2 and A_3. Input and output actions are denoted by ? and ! respectively. Components A_1 and A_2 communicate by synchronizing over c_{12} while A_2, A_3 do it over c_{23}. Components A_1 and A_3 do not communicate.

I/O Petri Nets. A *net* is a tuple $N = (P, T, F)$ where *(i)* $P \neq \emptyset$ is a set of *places*, *(ii)* $T \neq \emptyset$ is a set of *transitions* such that $P \cap T = \emptyset$, *(iii)* $F \subseteq (P \times T) \cup (T \times P)$ is a set of *flow arcs*. A *marking* is a multiset M of places, i.e. a map $M : P \to \mathbb{N}$. Let $\mathcal{I}n$ and $\mathcal{O}ut$ be two disjoint non-empty sets of *input* and *output* labels respectively. A *labeled Petri net* is a tuple $\mathcal{N} = (P, T, F, \lambda, M_0)$, where *(i)* (P, T, F) is a finite net; *(ii)* $\lambda : T \to (\mathcal{I}n \uplus \mathcal{O}ut)$ labels transitions by input/output actions; and *(iii)* $M_0 : P \to \mathbb{N}$ is an *initial marking*. Denote by $T^{\mathcal{I}n}$ and $T^{\mathcal{O}ut}$ the input and output transition sets, respectively; that is, $T^{\mathcal{I}n} \triangleq \lambda^{-1}(\mathcal{I}n)$ and $T^{\mathcal{O}ut} \triangleq \lambda^{-1}(\mathcal{O}ut)$. Elements of $P \cup T$ are called the *nodes* of $\mathcal{N}$. For a transition $t \in T$, we call ${}^\bullet t = \{p \mid (p, t) \in F\}$ the *preset* of t, and $t^\bullet = \{p \mid (t, p) \in F\}$ the *postset* of t. In figures, we represent as usual places by empty circles, transitions by squares, F by arrows, and the marking of a place p by black tokens in p. A transition t is *enabled* in marking M, written $M \xrightarrow{t}$, if $\forall p \in {}^\bullet t,\ M(p) > 0$. This enabled transition can *fire*, resulting in a new marking $M' = M - {}^\bullet t + t^\bullet$. This firing relation is denoted by $M \xrightarrow{t} M'$. A marking M is *reachable* from M_0 if there exists a *firing sequence*, i.e. transitions $t_0 \dots t_n$ such that $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} \dots \xrightarrow{t_n} M$. The set of markings reachable from M_0 (in $\mathcal{N}$) is denoted $\mathbf{R}_{\mathcal{N}}(M_0)$ (we drop the subscript referring to $\mathcal{N}$ when it is clear

[3] Action a from component A_i is labeled by a_i if necessary.

from the context). A Petri net $\mathcal{N} = (P, T, F, M_0)$ is *(1-)safe* iff for all reachable markings $M \in \mathbf{R}(M_0)$, $M(p) \in \{0, 1\}$ for all $p \in P$.

$\mathcal{N}$ is called *deterministically labeled* iff no two transitions with the same label are simultaneously enabled, i.e. for all $t_1, t_2 \in T$ and $M \in \mathbf{R}(M_0)$ we have $(M \xrightarrow{t_1} \wedge M \xrightarrow{t_2} \wedge \lambda(t_1) = \lambda(t_2)) \Rightarrow t_1 = t_2$. Notice that 1-safeness of the Petri net is not sufficient for guaranteeing deterministic labeling. Deterministic labeling ensures that the system behavior is locally discernible through labels, either through distinct inputs or through observation of different outputs.

When testing reactive systems, we need to differentiate situations where the system can still produce some outputs and those where the system cannot evolve without an input from the environment. Such situations are captured by the notion of *quiescence* [26]. A marking is said quiescent if it does not enable output transitions, i.e. $M \xrightarrow{t}$ implies $t \in T^{\mathcal{I}n}$. The observation of quiescence is usually instrumented by timers.

From Automata to Nets. The translation from an automaton $A = (Q, \Sigma, \Delta, q_0)$ to a labeled Petri net $\mathcal{N}_A = (P, T, F, \lambda, M_0)$ is immediate: *(i)* places are the states of the automaton, i.e. $P = Q$; *(ii)* for every transition $(s_i, a, s_i') \in \Delta$ we add t to T and set ${}^\bullet t = \{s_i\}, t^\bullet = \{s_i'\}$ and $\lambda(t) = a$; *(iii)* the initial state is the only place marked initially, i.e. $M_0 = \{q_0\}$.

The joint behavior of a system composed of automata $A_1, \ldots, A_n$ is modeled by $\mathcal{N}_{A_1} \times \cdots \times \mathcal{N}_{A_n}$ where $\times$ represents the product of labeled nets [27] and we only synchronize on communication transitions (which are invisible for the environment and thus labeled by τ). As different components are deterministic and they only share communication actions, the net obtained by this product is deterministically labeled. Product of nets prevents us from the state explosion problem that usually arises in product of automata. This product naturally allows to distinguish its components by means of a distribution [28]. A distribution $D : P \cup T \to \mathcal{P}(\{1, \ldots, n\})$ is a function that relates each place/transition with its corresponding automata. In the case of communication actions, the distribution relates the synchronized transition with the automata that communicate over it.

Example 2. Fig. 3 shows the net obtained from the automata in Fig. 2 and its distribution D represented by colors. The transition corresponding to action $?i_1$ corresponds to the first component, i.e. $D(?i_1) = \{1\}$, while communication between A_1 and A_2 is converted into a single transition τ_1 with $D(\tau_1) = \{1, 2\}$.

2 Partial Order Semantics

The semantics associated to a Petri net is given by its unfolding into an *occurrence net*, which can also be seen as an *event structure*. We will present and use both notions, according to the contexts. The execution traces for this semantics are not sequences but *partial orders*, in which concurrency is represented by absence of precedence. We recall here the basic notions.

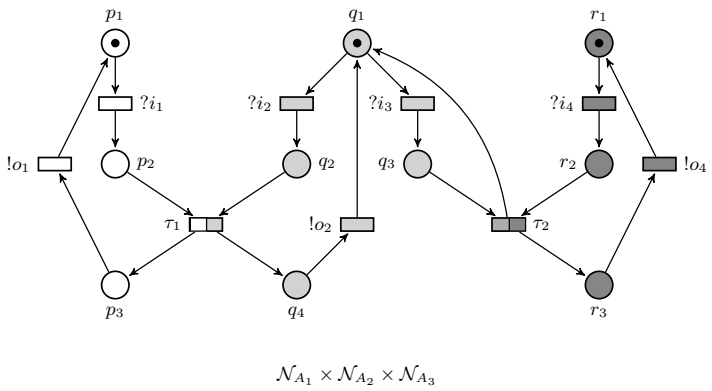

Fig. 3. Distributed Petri net.

2.1 Occurrence Nets and Unfoldings

Occurrence nets can be seen as Petri nets[4] with a special acyclic structure that highlights *conflict* between transitions that compete for resources. Formally, let $N = (P, T, F)$ be a net, $<_N$ the transitive closure of F, and $\leqslant_N$ the reflexive closure of $<_N$. We say that transitions t_1 and t_2 are in *structural conflict*, written $t_1 \#^\omega t_2$, if and only if $t_1 \neq t_2$ and ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$. *Conflict* is inherited along $<_N$, that is, the conflict relation $\#$ is given by $a \# b \Leftrightarrow \exists t_a, t_b \in T : t_a \#^\omega t_b \wedge t_a \leqslant_N a \wedge t_b \leqslant_N b$. Finally, the *concurrency relation* **co** holds between nodes $a, b \in P \cup T$ that are neither ordered nor in conflict, i.e. a **co** $b \Leftrightarrow \neg(a \leqslant b) \ \wedge \ \neg(a \# b) \ \wedge \ \neg(b < a)$.

Definition 1. *A net $ON = (B, E, G)$ is an* occurrence net *if and only if*

1. *$\leqslant_{ON}$ is a partial order;*
2. *for all $b \in B$, $|{}^\bullet b| \in \{0, 1\}$;*
3. *for all $x \in B \cup E$, the set $[x] = \{y \in E \mid y \leqslant x\}$ is finite;*
4. *no self-conflict, i.e. there is no $x \in B \cup E$ such that $x \# x$;*
5. *$\bot \in E$ is the only $\leq$-minimal node (event $\bot$ creates the initial conditions)*

Call the elements of E *events*, those of B *conditions*. A set of conditions is a co-set if its elements are pairwise in **co** relation. A maximal co-set with respect to set inclusion is called a *cut*. An occurrence net can also be given as a tuple $(B, E \backslash \{\bot\}, F, cut_0)$, where $cut_0 = \bot^\bullet$ is the set of minimal conditions. Occurrence nets are the mathematical form of the *partial order unfolding semantics* [13]. A *branching process* of a 1-safe Petri net $\mathcal{N} = (N, \lambda, M_0)$ is given by a pair $\Phi = (ON, \varphi)$, where $ON = (B, E, G)$ is an occurrence net, and $\varphi : B \cup E \to P \cup T$ is such that:

1. it is a homomorphism from ON to N, i.e.
   - $\varphi(B) \subseteq P$ and $\varphi(E) \subseteq T$, and

[4] when one allows Petri nets to be *infinite*

– for every $e \in E$, the restriction of φ to ${}^\bullet e$ is a bijection between the set ${}^\bullet e$ in ON and the set ${}^\bullet\varphi(e)$ in N, and similarly for $e^\bullet$ and $\varphi(e)^\bullet$;
2. the restriction of φ to cut_0 is a bijection from cut_0 to M_0; and
3. for every $e_1, e_2 \in E$, if ${}^\bullet e_1 = {}^\bullet e_2$ and $\varphi(e_1) = \varphi(e_2)$ then $e_1 = e_2$.

The unique (up to isomorphism) maximal branching process $\mathcal{U} = (ON_\mathcal{U}, \varphi_\mathcal{U})$ of $\mathcal{N}$ is called the *unfolding* of $\mathcal{N}$.

Remark 1. The unfolding of a distributed net is also a distributed net with distribution $D \circ \varphi$. By abuse of notation, we denote the distribution of a net and the distribution of its unfolding by the same name.

Remark 2. Notice that for every component A and its corresponding net $\mathcal{N}_A$ we have $\forall t \in T : |{}^\bullet t| = 1$. In the product between components and its unfolding, this property is only violated by communication transitions. Therefore, any input or output event in the unfolding can be enabled by exactly one condition.

2.2 Input/Output Labeled Event Structures

Occurrence nets give rise to event structures in the sense of Winskel et al [29]; as usual, we will use both the event structure and the occurrence net formalism, whichever is more convenient. An *input/output labeled event structure (IOLES)* over an alphabet $L = \mathcal{I}n \uplus \mathcal{O}ut$ is a 4-tuple $\mathcal{E} = (E, \leq, \#, \lambda)$ where *(i)* E is a set of events, *(ii)* $\leq \subseteq E \times E$ is a partial order (called *causality*) satisfying the property of *finite causes*, i.e. $\forall e \in E : |\{e' \in E \mid e' \leq e\}| < \infty$, *(iii)* $\# \subseteq E \times E$ is an irreflexive symmetric relation (called *conflict*) satisfying the property of *conflict heredity*, i.e. $\forall e, e', e'' \in E : e \# e' \wedge e' \leq e'' \Rightarrow e \# e''$, *(iv)* $\lambda : E \to (\mathcal{I}n \uplus \mathcal{O}ut)$ is a labeling mapping. In addition, we assume every IOLES $\mathcal{E}$ has a unique minimal event $\bot_\mathcal{E}$.

An event structure together with a distribution form a distributed IOLES. In such structures, we can distinguish events of different components, i.e. $E_d \triangleq \{e \in E \mid d \in D(e)\}$ for $d \in \{1, \ldots, n\}$. Such a distinction allows to project the unfolding onto a single component by just considering the events in E_d, and the restrictions of $\leq, \#$ and λ to E_d. The projection of an event structure to component $d \in \{1, \ldots, n\}$ is denoted by $\mathcal{E}_d$. We denote the class of all distributed input/output labeled event structures over L by $\mathcal{IOLES}(L)$.

Example 3. Fig. 4 shows the initial part of the unfolding of the net $\mathcal{N}_{A_1} \times \mathcal{N}_{A_2} \times \mathcal{N}_{A_3}$ given as a distributed event structure with its distribution represented by colors. As usual, we represent events by rectangles, causality by arrows and direct conflict with dashed lines. As in the case of the net in Fig. 3, communication events belong to more than one component.

The *local configuration* of an event e in $\mathcal{E}$ is defined as $[e]_\mathcal{E} \triangleq \{e' \in E \mid e' \leq e\}$, and its set of *causal predecessors* is $\langle e \rangle_\mathcal{E} \triangleq [e] \backslash \{e\}$. Two events $e, e' \in E$ are said to be concurrent (e **co** e') iff neither $e \leq e'$ nor $e' \leq e$ nor $e \# e'$ hold; $e, e' \in E$ are in *immediate conflict* ($e_1 \#^\mu e_2$) iff $[e_1] \times [e_2] \cap \# = \{(e_1, e_2)\}$. A *configuration*

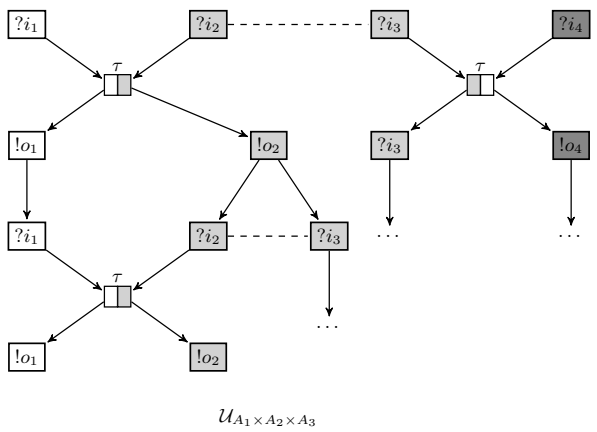

Fig. 4. Unfolding as an event structure.

of an IOLES is a non-empty set $C \subseteq E$ that is *(i) causally closed*, i.e. $e \in C$ implies $[e] \subseteq C$, and *(ii) conflict-free*, i.e. $e \in C$ and $e\#e'$ imply $e' \notin C$. Note that we define, for technical convenience, all configurations to be non-empty; the initial configuration of $\mathcal{E}$, containing only $\bot_{\mathcal{E}}$ and denoted by $\bot_{\mathcal{E}}$, is contained in every configuration of $\mathcal{E}$. We denote the set of all configurations of $\mathcal{E}$ by $\mathcal{C}(\mathcal{E})$.

Remark 3. In a distributed system, every configuration C generates a cut of the form $C^\bullet = \{q_1, \ldots, q_n\}$ where each condition q_d represents the current state of component A_d.

2.3 Executions

We are interested in testing distributed systems where concurrent actions occur in different components of the system. That is, the specifications we consider do not impose any order of execution between concurrent events. Labeled partial orders can then be used to represent executions of such systems.

Labeled Partial Orders. A *labeled partial order* (lpo) is a tuple $lpo = (E, \leq, \lambda)$ where E is a set of events, $\leq$ is a reflexive, antisymmetric, and transitive relation, and $\lambda : E \to L$ is a labeling mapping to a fix alphabet L. We denote the class of all labeled partial orders over L by $\mathcal{LPO}(L)$. Consider $lpo_1 = (E_1, \leq_1, \lambda_1)$ and $lpo_2 = (E_2, \leq_2, \lambda_2) \in \mathcal{LPO}(L)$. A bijective function $f : E_1 \to E_2$ is an isomorphism between lpo_1 and lpo_2 iff *(i)* $\forall e, e' \in E_1 : e \leq_1 e' \Leftrightarrow f(e) \leq_2 f(e')$ and *(ii)* $\forall e \in E_1 : \lambda_1(e) = \lambda_2(f(e))$. Two labeled partial orders lpo_1 and lpo_2 are isomorphic if there exists an isomorphism between them. A *partially ordered multiset* (pomset) is an isomorphism class of lpos. We will represent such a class by one of its objects. Denote the class of all non empty pomsets over L by $\mathcal{POMSET}(L)$.

The observable behavior of a system can be captured by abstracting the internal actions from the executions of the system. A pomset ω is the τ-abstraction of

another pomset μ, denoted by $abs(\mu) = \omega$, iff there exist $lpo_\mu = (E_\mu, \leq_\mu, \lambda_\mu) \in \mu$ and $lpo_\omega = (E_\omega, \leq_\omega, \lambda_\omega) \in \omega$ such that $E_\omega = \{e \in E_\mu \mid \lambda_\mu(e) \neq \tau\}$ and $\leq_\omega$ and λ_ω are the restrictions of $\leq_\mu$ and λ_μ to this set. Pomsets are observations; the observable evolution of the system is captured by the following definition:

Definition 2. *For $\mathcal{E} = (E, \leq, \#, \lambda) \in \mathcal{IOLES}(L)$, $\omega \in \mathcal{POMSET}(L)$ and $C, C' \in \mathcal{C}(\mathcal{E})$, define*

$$\begin{aligned} C \stackrel{\omega}{\Longrightarrow} C' &\triangleq \exists lpo = (E_\mu, \leq_\mu, \lambda_\mu) \in \mu : E_\mu \subseteq E \backslash C, C' = C \cup E_\mu, \\ &\qquad \leq \cap\, (E_\mu \times E_\mu) = \leq_\mu \textit{ and } \lambda_{|E_\mu} = \lambda_\mu \textit{ and } abs(\mu) = \omega \\ C \stackrel{\omega}{\Longrightarrow} &\triangleq \exists C' : C \stackrel{\omega}{\Longrightarrow} C' \end{aligned}$$

We can now define the notions of traces and of configurations reachable from a given configuration by an observation. Our notion of traces is similar to that of Ulrich and König [15].

Definition 3. *For $\mathcal{E} \in \mathcal{IOLES}(L)$, $\omega \in \mathcal{POMSET}(L), C, C' \in \mathcal{C}(\mathcal{E})$, define*

$$\begin{aligned} \textit{traces}(\mathcal{E}) &\triangleq \{\omega \in \mathcal{POMSET}(L) \mid \perp_\mathcal{E} \stackrel{\omega}{\Longrightarrow}\} \\ C \textbf{ \textit{after} } \omega &\triangleq \{C' \mid C \stackrel{\omega}{\Longrightarrow} C'\} \end{aligned}$$

Note that for deterministically labeled I/O Petri nets, the corresponding IOLES is deterministic and the set of reachable configurations is a singleton.

In a distributed system, global observation of the whole system is not available in general, i.e. the system is partially observed. This partial observation is captured by the projection of a global execution onto one of its component. As in the case of event structures, the projection of an execution only considers events belonging to a single component and restricts $\leq$ and λ to it. The projection of an execution ω onto component A_d is denoted by ω_d.

3 Testing Framework for IOPNs

3.1 Testing Hypotheses

We assume that the specification of the system under test is given as a network of deterministic automata $A_1, \ldots, A_n$ or as a 1-safe, distributed and deterministically labeled I/O Petri net $\mathcal{N} = (N, \lambda, M_0)$ over alphabet $L = \mathcal{I}n \uplus \mathcal{O}ut$. To be able to test an implementation against such a specification, we make a set of testing assumptions. First of all, we make the usual testing assumption that the behavior of the SUT itself can be modeled by a 1-safe distributed I/O Petri net over the same alphabet of labels. We also assume as usual that the specification does not contain cycles of outputs actions, so that the number of expected outputs after a given trace is finite.

Assumption 1 *The net $\mathcal{N}$ has no cycle containing only output transitions.*

Third, in order to allow the observation of both the outputs produced by the system and the inputs it can accept, markings where conflicting inputs and outputs are enabled should not be reachable. As a matter of fact, if conflicting input and output are enabled in a given marking, once the output is produced, the input is not enabled anymore, and vice versa. Such markings prevent from observing the inputs enabled in a given configuration, which we will see is one of the key points of our conformance relation. For this reason, we restrict the form of the nets we consider via the following assumption on the unfolding:[5]

Assumption 2 *The unfolding of the net $\mathcal{N}$ has no immediate conflict between input and output events, i.e.* $\forall e_1 \in E^{\mathcal{I}n}, e_2 \in E^{\mathcal{O}ut} : \neg(e_1 \#^{\mu} e_2)$.

3.2 Conformance Relation

A formal testing framework relies on the definition of a conformance relation to be satisfied by the SUT and its specification. In the LTS framework, the **ioco** conformance relation compares the outputs and blockings in the implementation after a trace of the specification to the outputs and blockings authorized after this trace in the specification.

Classically, the produced outputs of the system under test are elements of $\mathcal{O}ut$ (single actions) and blockings are observable by a special action $\delta \notin L$ which represents the expiration of a timer. By contrast, in partial order semantics, we need any set of outputs to be entirely produced by the SUT before we send a new input; this is necessary to detect outputs depending on extra inputs. For this reason we define the expected outputs from a configuration C as the pomset of outputs leading to a quiescent configuration. Such a configuration always exists, and must be finite by Assumption 1.

The notion of quiescence is inherited from nets, i.e. a configuration C is quiescent iff $C \stackrel{\omega}{\Longrightarrow}$ implies $\omega \in \mathcal{POMSET}(\mathcal{I}n)$. We assume as usual that quiescence is observable by a special δ action, i.e. C is quiescent iff $C \stackrel{\delta}{\Longrightarrow}$.

Definition 4. *For $\mathcal{E} \in \mathcal{IOLES}(L)$, $C \in \mathcal{C}(\mathcal{E})$, the outputs produced by C are*

$$out_{\mathcal{E}}(C) \triangleq \{!\omega \in \mathcal{POMSET}(\mathcal{O}ut) \mid C \stackrel{!\omega}{\Longrightarrow} C' \ \wedge \ C' \stackrel{\delta}{\Longrightarrow}\} \cup \{\delta \mid C \stackrel{\delta}{\Longrightarrow}\}.$$

The **ioco** theory assumes input enabledness of the implementation [1], i.e. in any state of the implementation, every input action is enabled. By constrast, we do not make this assumption, which is not always realistic [2, 3], and extend our conformance relation to consider refusals of inputs. Any possible input in a given state of the specification should be possible in a correct implementation.

Definition 5. *For $\mathcal{E} \in \mathcal{IOLES}(L)$ and $C \in \mathcal{C}(\mathcal{E})$, the possible inputs in C are*

$$poss_{\mathcal{E}}(C) \triangleq \{?\omega \in \mathcal{POMSET}(\mathcal{I}n) \mid C \stackrel{?\omega}{\Longrightarrow}\}$$

[5] Gaudel et al [3] assume a similar property called *IO-exclusiveness*.

Consider a given cut $C^\bullet = \{q_1, \ldots, q_n\}$ and a configuration C_d of a component A_d with $d \in \{1, \ldots, n\}$ such that $C_d^\bullet = \{q_d\}$. An event which is not enabled in C_d cannot be enabled in C. The following result is central and will help proving that global conformance can be achieved by local testers.

Proposition 1. *Let C (C_d) be a configuration of a distributed system (of the system component A_d) with the corresponding cut $C^\bullet = \{q_1, \ldots, q_n\}$ ($C_d^\bullet = \{q_d\}$). Then:*

(a) if $?i \notin poss_{\mathcal{E}_d}(C_d)$, then $?i \notin poss_{\mathcal{E}}(C)$,
(b) if $!o \notin out_{\mathcal{E}_d}(C_d)$, then for all $!\omega \in out_{\mathcal{E}}(C)$ we have $!\omega_d \neq !o$.

Proof. If an input or output event is not enabled in configuration C_d, then by Remark 2, there is no token in condition q_d. This absence prohibits such an event to be part of an execution of any larger configuration (w.r.t set inclusion). □

Notice the distinction between possible inputs and produced outputs. Whenever the systems reaches a configuration C that enables input actions in every component, i.e. $?i_d \in \text{poss}_{\mathcal{E}_d}(C_d)$ for all $d \in \{1, \ldots, n\}$, from the global point of view, not only $?i_1$ **co** $\ldots$ **co** $?i_n$ is possible for the system, but also every single input, i.e. $?i_d \in \text{poss}_{\mathcal{E}}(C)$. The same is not true for produced outputs. Consider a system that enables output $!o_d$ in component A_d, leading to a quiescent configuration in A_d, i.e. $!o_d \in \text{out}_{\mathcal{E}_d}(C_d)$. If other components also enable outputs actions, $!o_d \notin \text{out}_{\mathcal{E}}(C)$ as the global configuration after $!o_d$ is not quiescent.

Our **co-ioco** conformance relation for labeled event structures can be informally described as follows. The behavior of a correct **co-ioco** implementation after some observations (obtained from the specification) should respect the following restrictions: (1) the outputs produced by the implementation should be specified; (2) if a quiescent configuration is reached, this should also be the case in the specification; (3) any time an input is possible in the specification, this should also be the case in the implementation. These restrictions are formalized by the following conformance relation.

Definition 6 ([17]). *Let $\mathcal{S}$ and $\mathcal{I}$ be respectively the specification and implementation of a distributed system; then*

$$\begin{aligned} \mathcal{I} \textbf{ co-ioco } \mathcal{S} \Leftrightarrow\ & \forall \omega \in traces(\mathcal{S}) : \\ & poss_{\mathcal{S}}(\perp \textbf{ after } \omega) \subseteq poss_{\mathcal{I}}(\perp \textbf{ after } \omega) \\ & out_{\mathcal{I}}(\perp \textbf{ after } \omega) \subseteq out_{\mathcal{S}}(\perp \textbf{ after } \omega) \end{aligned}$$

When several outputs in conflicts are possible, our conformance relation allows implementations where at least one of them is implemented. Extra inputs are allowed in any configuration, but extra outputs, extra quiescence and extra causality between events specified as concurrent are forbidden.

Non conformance of the implementation is given by the absence of a given input or an unspecified output or quiescence in a configuration of the implementation. In a distributed system, a configuration defines the local state of each components as shown in Remark 3. Thus, non conformance of a distributed system is due to one of the following reasons:

(NC1) An input which is possible in a state of a component in the specification is not possible in its corresponding state in the implementation,

(NC2) A state of a component in the implementation produces an output or is quiescent while the corresponding state of the specification does not,

(NC3) The input (resp. output) actions that the configuration is ready to accept (resp. produce) are the same in both implementation and specification, but they do not form the same partial order, i.e. concurrency is added or removed.

The next section shows how we can detect these situations in a distributed testing environment.

4 Global Conformance by Distributed Testers

The **co-ioco** framework assumes a global view of the distributed system. In practice this assumption may not be satisfied and we can only observe the system partially, i.e. only the behavior of a local component in its PCO is observed. In a distributed testing environment, we place a local tester in each PCO. If we are in a pure distributed testing setting, these testers cannot communicate with each other during testing, and there is no global clock. We propose in this section a method that allows to decide global conformance by the conformance of every single component.

4.1 Local Testing

The last section described the three possible reasons for which a system may not conform to its specification. Non-conformance resulting from (NC1) and (NC2) can be locally tested under **co-ioco** by transforming each component into a net.

Theorem 1. *If $\mathcal{S}$ and $\mathcal{I}$ are, respectively, the specification and implementation of a distributed system, then $\mathcal{I}$ **co-ioco** $\mathcal{S}$ implies that for every $d \in \{1, \ldots, n\}$, $\mathcal{I}_d$ **co-ioco** $\mathcal{S}_d$.*

Proof. Assume there exists $d \in \{1, \ldots, n\}$ for which $\neg(\mathcal{I}_d$ **co-ioco** $\mathcal{S}_d)$, then there exists $\sigma \in \text{traces}(\mathcal{S}_d)$ such that one of the following holds:

- there exists $?i \in \text{poss}_{\mathcal{S}_d}(\perp$ **after** $\sigma)$, but $?i \notin \text{poss}_{\mathcal{I}_d}(\perp$ **after** $\sigma)$. Consider the global trace $\omega = \langle ?i \rangle_{\mathcal{S}}$ which enables $?i$ in $\mathcal{S}$, i.e. $?i \in \text{poss}_{\mathcal{S}}(\perp$ **after** $\omega)$. As $?i$ is not possible in $\mathcal{I}_d$, by Proposition 1 we have $?i \notin \text{poss}_{\mathcal{I}}(\perp$ **after** $\omega)$, and therefore $\neg(\mathcal{I}$ **co-ioco** $\mathcal{S})$.
- there exists $!o \in \text{out}_{\mathcal{I}_d}(\perp$ **after** $\sigma)$ such that $!o \notin \text{out}_{\mathcal{S}_d}(\perp$ **after** $\sigma)$. Consider the global trace $\omega = \langle !o \rangle_{\mathcal{I}}$ which enables $!o$ in $\mathcal{I}$, i.e. there exists $!\omega \in \text{out}_{\mathcal{I}}(\perp$ **after** $\omega)$ such that $!\omega_d = !o$. As $!o$ is not enabled in $\mathcal{S}_d$, by Proposition 1 we know that $!o$ cannot be enabled after ω in $\mathcal{S}$. Therefore, $!\omega \notin \text{out}_{\mathcal{S}}(\perp$ **after** $\omega)$ and $\neg(\mathcal{I}$ **co-ioco** $\mathcal{S})$.

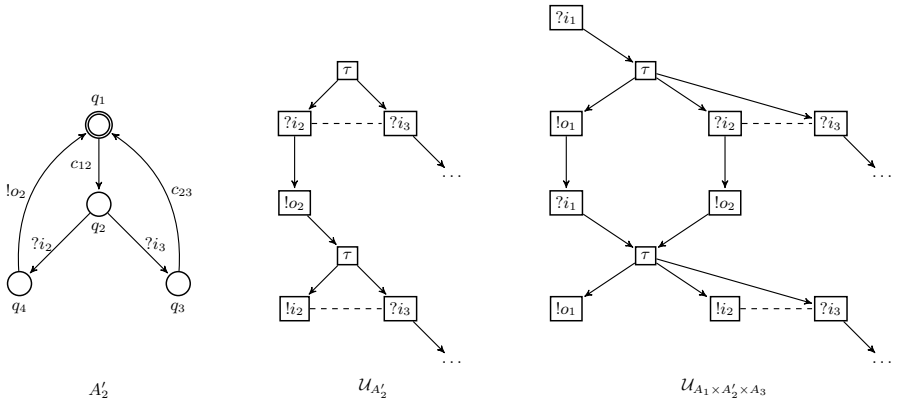

Fig. 5. Non conformant implementation.

– $\delta \in \text{out}_{\mathcal{I}_d}(\perp \textbf{ after } \sigma)$, while $\delta \notin \text{out}_{\mathcal{S}_d}(\perp \textbf{ after } \sigma)$. Let C_d be the configuration reached by component A_d of the implementation after σ and denote $C_d^\bullet = \{q_d\}$. Consider ω such that it leads the implementation to a quiescent configuration C with $q_d \in C^\bullet$ (such configuration always exists by Assumption 1); we have $\delta \in \text{out}_{\mathcal{I}}(\perp \textbf{ after } \omega)$. As the reached configuration in $\mathcal{S}_d$ is not quiescent, it enables some output and $\delta \notin \text{out}_{\mathcal{S}}(\perp \textbf{ after } \omega)$, therefore $\neg(\mathcal{I} \textbf{ co-ioco } \mathcal{S})$.

□

The simplest kind of conformance relations that we can obtain in a distributed architecture are those that only consider the observation of the system executions at each PCO without any further information. Such kind of relations include **p-dioco**, where the local behavior need to be consistent only with *some* global behavior. Stronger relations can be obtained if consistency between local observations is considered, as in the case of **dioco** where local behaviors must be projections of the *same* global behavior. However, not even this kind of relations test the dependencies between actions occurring on different components. Relations that assume global observation are usually stronger, as it is shown by the implication **ioco** ⇒ **p-dioco** or by the theorem above.

Example 4. Consider a component A'_2 where inputs $?i_2$ and $?i_3$ are not possible before this component synchronizes with A_1 and therefore they cannot occur before $?i_1$ occurs in A_1. Let $\mathcal{I} = A_1 \times A'_2 \times A_3$ and $\mathcal{S} = A_1 \times A_2 \times A_3$. Component A'_2, its unfolding and the unfolding of $\mathcal{I}$ are shown in Fig. 4.1. Component A'_2 conforms to A_2 as every possible input and produced outputs are implemented (only the order of synchronization events change, but those are not observable). Therefore, $\mathcal{I}_2$ **co-ioco** $\mathcal{S}_2$ and clearly, as **co-ioco** is reflexive, $\mathcal{I}_1$ **co-ioco** $\mathcal{S}_1$ and $\mathcal{I}_3$ **co-ioco** $\mathcal{S}_3$. In addition, the local behaviors $?i_1!o_1?i_1!o_1$ and $?i_2!o_2?i_2$ from components A_1 and A'_2, respectively, are projections of the same global behavior in $\mathcal{S}$, even if the causalities between components are not preserved. The **co-ioco** relation preserves causalities between actions in different components: the

possible input of the specification $?i_1$ **co** $?i_2 \in \text{poss}_{\mathcal{S}}(\perp)$ is not possible in the implementation, the actions are the same, but there is extra causality between them, i.e. $?i_1?i_2 \in \text{poss}_{\mathcal{I}}(\perp)$. We can conclude that $\neg(\mathcal{I}$ **co-ioco** $\mathcal{S})$ even if every component of the implementation conforms to the specification w.r.t **co-ioco** and local behaviors are projections of the same global behavior.

4.2 Adding Time Stamps

The example above shows that global conformance cannot always be achieved by local testers that do not communicate between themselves. This is exactly what happens in situation (NC3). However, as components of the implementation need to synchronize, we propose to use such synchronization to interchange some information that allows the testers to recompute the partial order between actions in different components using *vector clocks* [24, 25].

We assume each component A_d has a local clock that counts the number of interactions between itself and the environment, together with a local table of the form $[t_1^d, \ldots, t_n^d]$ with information about the clocks of every component (information about other components may not be updated). Each time two components communicate via synchronization, their local tables are updated.

We add the information about the tables to the model, i.e. events of the unfolding are tuples representing both the actions and the current values of the table. The unfolding structure allows to compute such tables very efficiently: when event e occurs in A_d, the value of the clock j in the table of A_d is equal to the number of input and outputs events from component j in the past of e, i.e. $t_j^d = [e] \cap (E_j^{\mathcal{I}n} \uplus E_j^{\mathcal{O}ut})$. The unfolding algorithm [13] can be modified to consider time stamps as it is shown in Algorithm 1. The behavior of system $\mathcal{E}$ where time stamps are considered is denoted by $\mathcal{E}^{ts}$.

Example 5. Consider the time stamped unfolding $\mathcal{U}^{ts}_{A_1 \times A_2 \times A_3}$ on Fig. 6. The first occurrence of action $!o_1$ is stamped by $[2, 1, 0]$, meaning that it is the second interaction with the environment in component A_1, and at least there was one interaction between the environment and component A_2 before the occurrence of $!o_1$. The information of component A_2 is propagated to the table of component A_1 after their synchronize over the first occurrence of τ.

The global trace of an execution of the system can be reconstructed from the local traces of this execution observed in PCOs using the information provided by time stamps.

Example 6. Consider Fig. 7 and the time stamped local traces σ_1 and σ_2 of the first and second component. From event $(!o_2, 1, 2, 0)$, we know that at least one event from the first component precedes $!o_2$, and as $?i_1$ is the first action in such component, we can add the causality $(?i_1, 1, 0, 0) \leq (!o_1, 1, 2, 0)$ as shown in the partial order ω.

Given two time stamped LPOs $\omega_i = (E_i, \leq_i, \lambda_1)$ and $\omega_j = (E_j, \leq_j, \lambda_2)$, their joint causality is given by the LPO $\omega_i + \omega_j = (E_i \uplus E_j, \leq_{ij}, \lambda_1 \uplus \lambda_2)$ where for

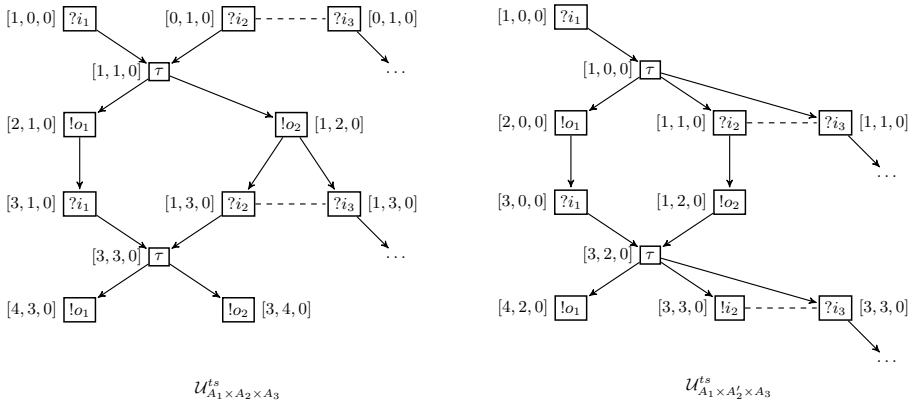

Fig. 6. Part of the time stamped unfolding $\mathcal{S}$ and $\mathcal{I}'$.

Algorithm 1 Time stamped unfolding algorithm

Require: A 1-safe I/O Petri net $\mathcal{N} = (T, P, F, M_0, \lambda)$ where $M_0 = \{s_1, \ldots, s_n\}$ and a distribution $D : T \cup P \to \{1, \ldots, n\}$.
Ensure: The time stamped unfolding of $\mathcal{N}$

$B := (s_0, \emptyset), \ldots, (s_k, \emptyset)$
$E = \emptyset$
$pe := PE(S)$
while $pe \neq \emptyset$ **do**
  choose an event $e = (t, {}^\bullet t)$ in pe
  for $d \in \{1, \ldots, n\}$ **do**
    $t_d(e) := |\{(t', {}^\bullet t') \in E \mid D(t') = d \wedge \lambda(t) \neq \tau\}| + 1$
  end for
  append to E the event $e \times t_1(e) \times \cdots \times t_n(e)$
  for every place s in $t^\bullet$ add a condition (s, e)
  $pe := PE(S)$;
end while
return (B, E)

each pair of events $e_1 = (a, t_1^i, \ldots, t_n^i) \in E_i$ and $e_2 \in E_i \uplus E_j$, we have

$$e_2 \leq_{ij} e_1 \Leftrightarrow e_2 \leq_i e_1 \vee |[e_2]_j| = t_j^i$$

When communication between components is asynchronous, a configuration C is called *consistent* if for every sending message in C, its corresponding receive message is also in C. Mattern [24] shows that consistent configurations have unambiguous time stamps; hence global causality can be reconstructed from local observations in an unique way. Under synchronous communication, the send and receive actions are represented by the same event, and therefore every configuration is consistent.

Proposition 2. *When the communication between components is synchronous, the partial order obtained by + is unique.*

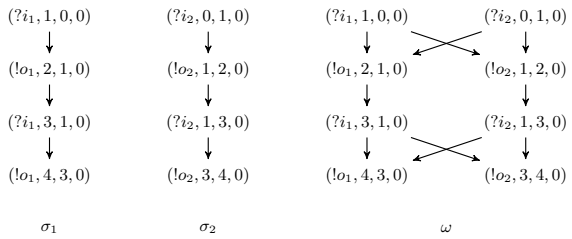

Fig. 7. From local traces to partial orders using time stamps.

Therefore, non conformance coming from (NC3) can be detected by testing the time stamped system in a distributed way.

Theorem 2. *If $\mathcal{S}$ and $\mathcal{I}$ are, respectively, the specification and implementation of a distributed system, then $\forall d \in \{1, \ldots, n\}$: $\mathcal{I}_d^{ts}$ **co-ioco** $\mathcal{S}_d^{ts}$ implies $\mathcal{I}$ **co-ioco** $\mathcal{S}$.*

Proof. Assume $\mathcal{I}_d^{ts}$ **co-ioco** $\mathcal{S}_d^{ts}$ for every $d \in \{1, \ldots n\}$. Let $\omega \in \text{traces}(\mathcal{S})$ and consider the following situations:

- if $?\omega \in \text{poss}_{\mathcal{S}}(\bot \textbf{ after } \omega)$, then for every d there exists a time stamped input $(?i_d, t_1^d, \ldots, t_n^d) \in \text{poss}_{\mathcal{S}_d^{ts}}(\bot \textbf{ after } \omega_d)$ such that $?\omega_d = ?i_d$ and $?\omega = ?i_1 + \cdots + ?i_n$. As for all d, we have $\mathcal{I}_d$ **co-ioco** $\mathcal{S}_d$, then $(?i_d, t_1^d, \ldots, t_n^d) \in \text{poss}_{\mathcal{I}_d^{ts}}(\bot \textbf{ after } \omega_d)$. By Proposition 2, $?\omega \in \text{poss}_{\mathcal{I}}(\bot \textbf{ after } \omega)$.
- if $!\omega \in \text{out}_{\mathcal{I}}(\bot \textbf{ after } \omega)$, then for every d there exists a time stamped output $(!o_d, t_1^d, \ldots, t_n^d) \in \text{out}_{\mathcal{I}_d^{ts}}(\bot \textbf{ after } \omega_d)$ such that $!\omega_d = !o_d$ and $!\omega = !o_1 + \cdots + !o_n$. As every component of the implementation conforms to its specification, we have $(!o_d, t_1^d, \ldots, t_n^d) \in \text{out}_{\mathcal{S}_d^{ts}}(\bot \textbf{ after } \omega_d)$. By Proposition 2, we have $!\omega \in \text{out}_{\mathcal{S}}(\bot \textbf{ after } \omega)$.
- if $\delta \in \text{out}_{\mathcal{I}}(\bot \textbf{ after } \omega)$, let C be the configuration reached by the implementation after ω and denote $C^\bullet = \{q_1, \ldots, q_n\}$. As configuration C is quiescent, then so they are each configuration C_d such that $C_d^\bullet = \{q_d\}$, thus $\delta \in \text{out}_{\mathcal{I}_d^{ts}}(\bot \textbf{ after } \omega_d)$. As $\mathcal{I}_d$ **co-ioco** $\mathcal{S}_d$ for each d, we have $\delta \in \text{out}_{\mathcal{S}_s^{ts}}(\bot \textbf{ after } \omega_d)$. This implies that the local configurations of the specification do not enable any output; by Remark 2, there is no output enabled in the global configuration and $\delta \in \text{out}_{\mathcal{S}}(\bot \textbf{ after } \omega)$.

These three cases allow us to conclude that $\mathcal{I}$ **co-ioco** $\mathcal{S}$. □

Example 7. The non conformance situation presented in Example 4 can now be tested locally. The time stamped unfoldings of $\mathcal{S}$ and $\mathcal{I}$ are presented in Fig. 6. If we project $\mathcal{S}$ to its second component, we see that $(?i_2, 0, 1, 0)$ is a possible input from the initial configuration., i.e. $(?i_2, 0, 1, 0) \in \text{poss}_{\mathcal{S}_2}(\bot)$. However, if we project $\mathcal{I}$ to the second component, the action $?i_2$ is the same, but the time stamps are different, therefore $(?i_2, 0, 1, 0) \notin \text{poss}_{\mathcal{I}_2}(\bot)$ and we have $\neg(\mathcal{I}_2$ **co-ioco** $\mathcal{S}_2)$.

From a global test case to local test cases. We have shown that global conformance can be achieved by distributed testers. However testers need to consider time stamp information which cannot be computed locally. The test case generation algorithm that we proposed for concurrent systems [17] can easily be adapted to consider the time stamped unfolding presented in this article. The global test case obtained is a distributed IOLES and can therefore be projected to each component to obtain a distributed test case, i.e. a set of local test cases.

We also gave a sound and exhaustive test set for **co-ioco** [17]. The set of distributed test cases obtained by projecting global test cases of this complete test set naturally is also complete for **co-ioco** by Theorem 2. Therefore, it allows to decide global conformance w.r.t. **co-ioco** by distributed testing, with independent local testers.

5 Conclusion

We have presented a distributed testing framework for concurrent systems specified as a network of automata or, equivalently, as a 1-safe Petri net. The **co-ioco** conformance relation introduced in our previous work is put into a distributed testing architecture, where nets are distributed, observation of the system is partial, and global configurations are represented by the collection of local states of components. When the implementation is equipped with local clocks, a global test case can be constructed adapting the test generation algorithm for **co-ioco** for handling time stamps. Global test cases can be projected into local test cases, which allows to achieve global conformance via conformance of local components.

This approach considers synchronous communication. Future work includes the extension to asynchronous communication not only between components, but also between the testers and the SUT.

References

1. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. Software - Concepts and Tools **17**(3) (1996) 103–120
2. Heerink, L., Tretmans, J.: Refusal testing for classes of transition systems with inputs and outputs. In: FORTE. Volume 107 of IFIP Conference Proceedings., Chapman & Hall (1997) 23–38
3. Lestiennes, G., Gaudel, M.C.: Test de systèmes réactifs non réceptifs. Journal Européen des Systèmes Automatisés **39**(1-2-3) (2005) 255–270
4. Faivre, A., Gaston, C., Le Gall, P., Touil, A.: Test purpose concretization through symbolic action refinement. In: TestCom. Volume 5047 of LNCS., Springer (2008) 184–199
5. Jéron, T.: Symbolic model-based test selection. ENTCS **240** (2009) 167–184
6. Krichen, M., Tripakis, S.: Conformance testing for real-time systems. Formal Methods in System Design **34**(3) (2009) 238–304
7. Hessel, A., Larsen, K.G., Mikucionis, M., Nielsen, B., Pettersson, P., Skou, A.: Testing real-time systems using UPPAAL. In: Formal Methods and Testing. Volume 4949 of LNCS., Springer (2008) 77–117

8. Hierons, R.M., Merayo, M.G., Núñez, M.: Implementation relations for the distributed test architecture. In: TestCom. Volume 5047 of LNCS., Springer (2008) 200–215
9. Hennessy, M.: Algebraic Theory of Processes. MIT Press (1988)
10. Peleska, J., Siegel, M.: From testing theory to test driver implementation. In: FME. (1996) 538–556
11. Schneider, S.: Concurrent and Real Time Systems: The CSP Approach. 1st edn. John Wiley & Sons, Inc., New York, NY, USA (1999)
12. McMillan, K.L.: A technique of state space search based on unfolding. Formal Methods in System Design **6**(1) (1995) 45–65
13. Esparza, J., Römer, S., Vogler, W.: An improvement of McMillan's unfolding algorithm. In: TACAS. Volume 1055 of LNCS., Springer (1996) 87–106
14. Jard, C.: Synthesis of distributed testers from true-concurrency models of reactive systems. Information & Software Technology **45**(12) (2003) 805–814
15. Ulrich, A., König, H.: Specification-based testing of concurrent systems. In: FORTE. Volume 107 of IFIP Conference Proceedings., Chapman & Hall (1998) 7–22
16. Ponce de León, H., Haar, S., Longuet, D.: Conformance relations for labeled event structures. In: Tests and Proofs. Volume 7305 of LNCS., Springer (2012) 83–98
17. Ponce de León, H., Haar, S., Longuet, D.: Unfolding-based test selection for concurrent conformance. In: ICTSS. Volume 8254 of LNCS., Springer (2013) 98–113
18. Ponce de León, H., Haar, S., Longuet, D.: Model based testing for concurrent systems with labeled event structures. http://hal.inria.fr/hal-00796006 (2012)
19. Bhateja, P., Gastin, P., Mukund, M., Kumar, K.N.: Local testing of message sequence charts is difficult. In: FCT. Volume 4639 of LNCS. (2007) 76–87
20. Longuet, D.: Global and local testing from message sequence charts. In: SAC, Software Verification and Testing track, ACM (2012) 1332–1338
21. Jard, C., Jéron, T., Kahlouche, H., Viho, C.: Towards automatic distribution of testers for distributed conformance testing. In: FORTE. Volume 135 of IFIP Conference Proceedings., Kluwer (1998) 353–368
22. Bhateja, P., Mukund, M.: Tagging make local testing of message-passing systems feasible. In: SEFM, IEEE Computer Society (2008) 171–180
23. Hierons, R.M., Merayo, M.G., Núñez, M.: Using time to add order to distributed testing. In: FM. Volume 7436 of LNCS. (2012) 232–246
24. Mattern, F.: Virtual time and global states of distributed systems. In: Parallel and Distributed Algorithms, North-Holland (1989) 215–226
25. Fidge, C.J.: Timestamps in Message-Passing Systems that Preserve the Partial Ordering. In: 11th Australian Computer Science Conference, University of Queensland, Australia (1988) 55–66
26. Segala, R.: Quiescence, fairness, testing, and the notion of implementation. Information and Computation **138**(2) (1997) 194–210
27. Winskel, G.: Petri nets, morphisms and compositionality. In: Applications and Theory in Petri Nets. (1985) 453–477
28. van Glabbeek, R.J., Goltz, U., Schicke-Uffmann, J.W.: On distributability of Petri nets - (extended abstract). In: FoSSaCS. Volume 7213 of LNCS. (2012) 331–345
29. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, part I. Theoretical Computer Science **13** (1981) 85–108