

Distributed computation of vector clocks in Petri nets unfolding for test selection

Loïc Jezequel¹, Agnes Madalinski², and Stefan Schwoon³

¹ Université de Nantes, LS2N, UMR CNRS 6004, France
(e-mail: loig.jezequel@ls2n.fr)

² Chair of Software Engineering, OvGU Magdeburg, Germany
(e-mail: agnes.madalinski@ovgu.de)

³ LSV (CNRS & ENS Cachan), Univ. Paris-Saclay & Inria, France
(e-mail: schwoon@lsv.ens-cachan.fr)

Abstract. It has been shown that annotating Petri net unfoldings with time stamps allows for building distributed testers for distributed systems. However, the construction of the annotated unfolding of a distributed system currently remains a centralized task. In this paper we extend a distributed unfolding technique in order to annotate the resulting unfolding with time stamps. This allows for distributed construction of distributed testers for distributed systems.

1 Introduction

The **co-ioco** framework proposed in [7] introduced partial-order semantics to the well-known **ioco** theory of [11]. In both cases, inputs and outputs (or their absence) of the implementation are compared to those of the specification. However, in the **co-ioco** setting, traces, inputs, and outputs are considered as partial orders, where actions specified as concurrent need to be implemented as such. This is of essential importance for distributed systems where concurrency captures the physical distribution of the components of the system.

Test architectures for distributed systems can be classified into two types: *global testers* that have control over the entire system under test, and *distributed testers* where several single and concurrent testers are controlling the components of the system under test. In [7, 1] it is shown how, starting from a system specified as a Petri net or a network of automata, a global tester can be constructed using Petri net unfoldings and SAT. Additionally, [8] show that if the unfolding procedure is extended with time stamps, the resulting global tester can be transformed into a distributed one. However, the computation of the global tester is still centralized and constructs (a prefix of) the unfolding of the entire system.

In this work we provide a novel framework for constructing single testers (together constituting a distributed tester) containing the time-stamp information necessary to test global conformance. These testers are computed in a distributed way. This work is grounded on the results of [9] and [3], extending them with

timed information. Our timed information is *logical*: it counts occurrences of actions. This is similar to vector clocks in distributed systems, cf. [6, 5, 10].

This paper is organized as follows. In Section 2 we provide a running example. In Section 3, we give the basic notations for our formal model. The notion of Petri-net unfolding and its link to testers are defined in Section 4. After that, we recall distributed Petri-net unfolding in Section 5, and we extend these to incorporate time stamps in Section 6. Finally, we show how to build interface summaries with time stamps, which are central for distributed unfoldings, in Section 7.

The main contribution of this paper is the method we propose for computing time stamps during distributed unfolding (described in Section 6 and at the end of Section 7). Unlike [8], our approach avoids the expensive operation of unfolding the entire system, working on individual components instead. This also allows to unify the results of [9] and [3] in order to build a distributed algorithm for unfolding Petri nets.

2 Running example

As a running example, we shall use the Petri net depicted in Figure 2, showing the interaction of a consumer (in the middle) with two producers (left and right). For various purposes, we will use two different decompositions of it:

1. into the five *subnets* indicated in Figure 2, where A, C are the producers, B is the consumer, and X, Y are the interfaces between the consumer and the left (respectively right) producer;
2. into three (overlapping) *components* $\mathcal{A}, \mathcal{B}, \mathcal{C}$, where \mathcal{A} consists of producer A with its interface X , \mathcal{C} consists of producer C with its interface Y , and \mathcal{B} consists of consumer B with both interfaces.

Notice that each place and transition belongs to exactly one subnet. The places and transitions of the interfaces belong to exactly two components, and the others belong to exactly one.

Intuitively, producer A offers some product that it can create from certain base materials. Each of these materials may be available (places p', q', r') or not (p, q, r). Interface X acts as an agent handling the interaction between A and B . This agent, upon receiving an order from the consumer B , registers a demand for the product with A . Depending on the availability of the base materials, the product can be either delivered to the consumer, or one of the sides decides to cancel the transaction. Producer C works in a similar fashion, using different base materials (places s, t and s', t') and communicating through interface Y . Finally, B can be thought of as a customer wishing to perform certain actions x, y , or z . For x and z only one of the producers is needed, whereas for y both are required.

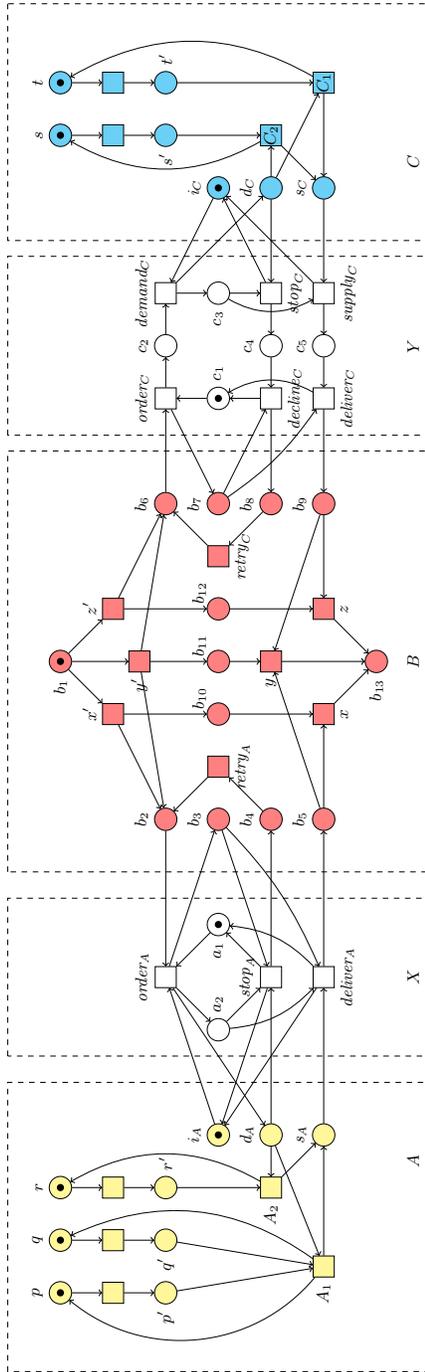


Fig. 1. An example with one consumer B and two producers A, C connected by interfaces X, Y .

3 Distributed systems modeling

We now formalize the framework exemplified above: compound systems represented as *labelled Petri nets*. We fix an alphabet Σ of *labels* and a set of *components* \mathbb{C} .

A *Petri net* is a tuple $\text{pn} = \langle P, T, F, M_0, \ell, \gamma \rangle$ where P and T are two disjoint sets of nodes called *places* and *transitions*, respectively, $F \subseteq P \times T \cup T \times P$ is a *flow relation*, $M_0 \subseteq P$ is an *initial marking*, $\ell : (P \cup T) \rightarrow \Sigma$ associates labels to nodes, and $\gamma : (P \cup T) \rightarrow 2^{\mathbb{C}}$ associates each node with a subset of the components. The elements of F are called the *arcs*.

For consistency, we require that for all $p \in P$, $t \in T$, $\langle p, t \rangle \in F$ or $\langle t, p \rangle \in F$ implies $\gamma(p) \subseteq \gamma(t)$, i.e. a transition can only interact with places of its own components. Moreover, every component $c \in \mathbb{C}$ has at least one initially marked place, i.e. there exists $p \in M_0$ with $c \in \gamma(p)$.

Example. Figure 2 shows an example of a Petri net. Places are represented as circles (with black tokens for the initial marking), transitions as rectangles, and the flow relation as arrows. Labels are shown next to the nodes, and $\mathbb{C} = \{A, B, C\}$, corresponding to $\mathcal{A}, \mathcal{B}, \mathcal{C}$ from Section 2, where nodes in X belong to both A and B, those in Y to both B and C, and all others to exactly one of A, B, C.

A *marking* is a set of places. A transition t is said to be *firable* from *marking* $M \subseteq P$ iff $\bullet t = \{p \mid \langle p, t \rangle \in F\} \subseteq M$. In this case the *firing* of t leads to the new marking $(M \setminus \bullet t) \cup t^\bullet$, where $t^\bullet = \{p \mid \langle t, p \rangle \in F\}$. A marking M is *reachable* in pn if and only if there exists a sequence of transition firings leading from M_0 to M .

Notice that our definition of marking and our semantics of firing are simplified with respect to the standard definitions; this is justified by the fact that we deal with *safe* nets only, where for any reachable marking M and any transition t , $\bullet t \subseteq M$ implies $M \cap (t^\bullet \setminus \bullet t) = \emptyset$.

Projections and interfaces. Let $\mathbb{C}' \subseteq \mathbb{C}$. The *projection of pn to \mathbb{C}'* is the net $\pi_{\mathbb{C}'}(\text{pn}) = \langle P', T', F', M'_0, \ell', \gamma' \rangle$ that preserves only nodes from \mathbb{C}' , i.e. $P' \subseteq P$, $T' \subseteq T$, and $x \in P' \cup T'$ iff $\gamma(x) \cap \mathbb{C}' \neq \emptyset$; moreover, F', M'_0, ℓ', γ' are the respective restrictions of F, M_0, ℓ, γ to $P' \cup T'$. Note that γ' is still a mapping to \mathbb{C} , not just to \mathbb{C}' . Abusing notation, we shall write π_c for $\pi_{\{c\}}$ when $c \in \mathbb{C}$. For instance, when pn is the net in Figure 2, we obtain $\mathcal{A} = \pi_{\mathcal{A}}(\text{pn})$, $\mathcal{B} = \pi_{\mathcal{B}}(\text{pn})$, and $\mathcal{C} = \pi_{\mathcal{C}}(\text{pn})$. With this in mind, we also call $\mathcal{A}, \mathcal{B}, \mathcal{C}$ “components” when there is no confusion.

For two components $c, c' \in \mathbb{C}$, the *interface* of pn between c and c' is $\pi_c(\pi_{c'}(\text{pn})) = \pi_{c'}(\pi_c(\text{pn}))$. For instance, the interface of A, B in Figure 2 is the subnet X , and Y is the interface of B, C. A net is said to contain a *gateway* to component c if it contains nodes that belong to c but none that belong to c alone. For instance, $\mathcal{A} = \pi_{\mathcal{A}}(\text{pn})$ contains a gateway to B (which is exactly the interface X).

The *interaction graph* of pn is the undirected graph whose nodes are the elements of \mathbb{C} , and where $\{c, c'\}$ is an edge iff c and c' have a non-empty interface. A net is *tree-like* if its interaction graph is a tree.

An *automaton* is a Petri net such that for any transition t one has $|\bullet t| = |t\bullet| = 1$, and where M_0 is a singleton. For instance, the net in Figure 3 obeys this condition.

Timed nets. Finally, in certain cases we will wish to include (logical) timing information in Petri nets. A *vector clock* is a mapping from \mathbb{C} to \mathbb{N} . In practice, vector clocks shall count how many events have fired per component. A *timed net* is a pair $\overline{\text{pn}} = \langle \text{pn}, \theta \rangle$, equipping a Petri net pn with a *clock mapping* θ that associates a vector clock to each transition of pn . The projection of $\overline{\text{pn}}$ to $\mathbb{C}' \subseteq \mathbb{C}$ is $\overline{\pi_{\mathbb{C}'}}(\overline{\text{pn}}) := \langle \pi_{\mathbb{C}'}(\text{pn}), \theta' \rangle$, where θ' is the restriction of θ to the nodes of $\pi_{\mathbb{C}'}(\text{pn})$. Note that this operation retains vector clock information for all components in \mathbb{C} , including components outside \mathbb{C}' .

4 Petri nets unfolding and test

The unfolding of a Petri net pn is another, acyclic Petri net $\text{un}(\text{pn})$ that describes the complete behaviour of pn : all reachable markings and all partial orders of transition firings are preserved. The unfolding of a Petri net in fact enumerates the partial orders of transition firings of pn , merged on common prefixes. In general, the unfolding is an infinite structure, but our algorithms only consider finite prefixes of it. As shown in [8], adding well-chosen time stamps to the unfolding of a Petri net allows to build a distributed tester for the compound system modeled by the original net – simply by projecting a finite prefix of the unfolding onto each component.

In this section, we formally define unfoldings. We then show how time stamps should be added to an unfolding.

Unfoldings are usually formalized from the notion of *branching process*. A *branching process* of a Petri net $\text{pn} = \langle P, T, F, M_0, \ell, \gamma \rangle$ is a Petri net $\text{pn}' = \langle Q, E, F', M'_0, \ell', \gamma' \rangle$ (where places are usually called conditions, and transitions are usually called events) associated with a function $\lambda : Q \cup E \rightarrow P \cup T$. (Abusing notation, we naturally extend λ to sets.) Assuming $M_0 = \{p_0, \dots, p_k\}$, the finite branching processes of pn are defined inductively as follows:

- The net with conditions q_0, \dots, q_k so that $\lambda(q_0) = p_0, \dots, \lambda(q_k) = p_k$, no events, and initial marking $\{q_0, \dots, q_k\}$ is a branching process of pn .
- Let pn' be a branching process of pn where for some reachable marking M of pn' , $\lambda(M)$ makes some transition t of pn firable. Let M' be the subset of M such that $\lambda(M') = \bullet t$. If pn' has no event e with $\lambda(e) = t$ such that $\bullet e = M'$, then the net obtained by adding to pn' a new event e with $\lambda(e) = t$, a new condition for every place p of $t\bullet$ labelled by p , new arcs from each condition of M' to e and from e to each new condition, is also a branching process of pn . In this case, we call e an extension of pn' .

- For all nodes x , $\ell'(x) = \ell(\lambda(x))$ and $\gamma'(x) = \gamma(\lambda(x))$.

The set of all branching processes of a Petri net pn , finite and infinite, is defined by closing the finite branching processes under countable unions (see [2]). In particular, the union of all finite branching processes is called the *unfolding* of pn , denoted $\text{un}(\text{pn})$.

Example. Consider Figure 2 (left). It depicts a finite branching process of the net from Figure 2 restricted to component A, i.e. producer A and its interface X. The labelling λ is given next to the conditions and events.

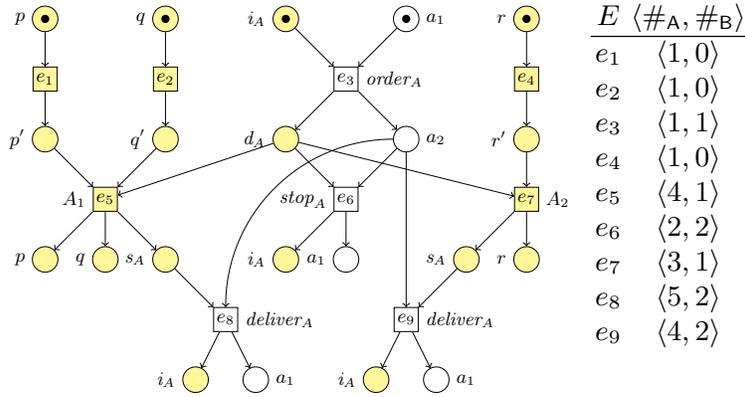


Fig. 2. A branching process of $\pi_A(\text{pn})$, where pn is the net from Figure 2, and associated time stamps.

Given two nodes x, y of a branching process, we say that: x is a *causal predecessor* of y , denoted $x < y$, if there exists a non-empty path of arcs from x to y . By $x \leq y$ we mean $x < y$ or $x = y$, and if $x \leq y$ or $y \leq x$ we say that x and y are *causally related*. The nodes x and y are in *conflict* if there exists a condition c (different from x and y) so that one can reach both x and y from c via two paths that start with different arcs. The nodes x and y are *concurrent* if they are neither causally related nor in conflict.

Given an event e of a branching process, we define its *configuration*, noted $[e]$, as the set of its causal predecessors events: $[e] = \{e' \mid e' \leq e\}$.

A *timed branching process* of pn is the pair $\langle \text{pn}', \theta \rangle$, where pn' is a branching process of pn , and $\theta(e)(c)$ is the number of events e' satisfying $e' \leq e$ and $c \in \gamma'(e')$. In this case, θ is also called a collection of *time stamps*. We denote $\overline{\text{un}}(\text{pn})$ as the *timed unfolding* of pn .

Example. Consider again Figure 2, and recall that nodes of the interface X belong to both A and B. The column marked $\#_A$ provides the value $\theta(e)(A)$ for every event e (ditto for B).

In the following, we consider projections of timed or untimed unfoldings to components $\mathbb{C}' \subseteq \mathbb{C}$. In this context, notice that $\bar{\pi}_{\mathbb{C}'}(\bar{\text{un}}(\text{pn}))$ is not the same as $\bar{\text{un}}(\pi_{\mathbb{C}'}(\text{pn}))$; the latter contains vector clock information from transitions in $\pi_{\mathbb{C}'}(\text{pn})$ alone, while the former has it for all of pn .

Example. Figure 3 shows the projection of the timed branching process from Figure 2 to \mathbb{B} . Only the conditions of X and the events in which it participates are preserved.

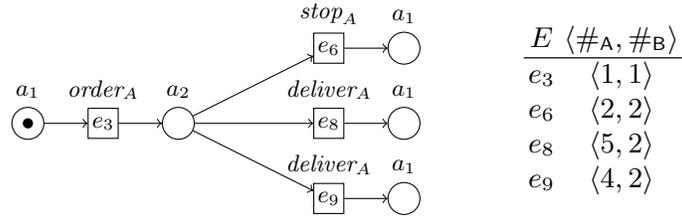


Fig. 3. Projection of Figure 2 to component \mathbb{B} .

5 Distributed unfolding

In this section, we review a *distributed* algorithm for computing (a prefix of) the unfolding of a net *without time stamps*. There exists a straightforward, centralized algorithm for this task, which applies the inductive characterization of a branching processes, adding one event at a time (see [2]). The result is easily equipped with time stamps, and it is known ([8]) how to use a finite unfolding prefix to produce distributed testers. However, the intermediate unfolding prefix can be much larger than those testers.

We are interested by the approach of [9], which computes unfolding prefixes component by component. In this section, we briefly describe a slightly generalized version of this algorithm (the original algorithm computes *complete* prefixes whereas testers are built on arbitrary prefixes). This method uses smaller unfolding prefixes, but adding time stamps is not straightforward. Starting from Section 6, we show how to close this gap.

Let $\text{pn} = \langle P, T, F, M_0, \ell, \gamma \rangle$ be a compound system represented as a Petri net. In order to perform distributed unfolding, we impose the following restrictions:

1. pn is tree-like.
2. Every (non-empty) interface in pn is an automaton.

The first assumption is required by the distributed unfolding approach of [9], which can be slightly weakened (see [4]). Notice that this implies $|\gamma(x)| \leq 2$ for any node x . The second restriction comes from our solution to distributed unfoldings with timing information, based on [3].

Example. Consider again Figure 2. This decomposition respects our restrictions: the interfaces X and Y are automata, and \mathcal{B} communicates with \mathcal{A} and \mathcal{C} .

Let $c, c' \in \mathbb{C}$ be two components. A $\{c, c'\}$ -*automaton* is an automaton in which $\gamma(x) = \{c, c'\}$ for all nodes x . According to our restriction, every non-empty interface X in pn is a $\{c, c'\}$ -automaton (for some c, c'). If moreover X is a gateway to c' , we call it a (c, c') -gateway.

Let \mathcal{A} be a tree-like net with a (c, c') -gateway X . Then we denote $\mathbb{C}[\mathcal{A}, X] \subseteq \mathbb{C}$ the set of all components having at least one node in \mathcal{A} , except c' . Let \mathcal{B} be another tree-like net with a (c', c) -gateway Y such that $\{c, c'\}$ is the only edge appearing in the interaction graphs of both \mathcal{A} and \mathcal{B} . Then we say that \mathcal{A}, \mathcal{B} *meet at* $\{c, c'\}$. Notice that in this case, $\mathbb{C}[\mathcal{A}, X]$ and $\mathbb{C}[\mathcal{B}, Y]$ are disjoint.

Distributed unfoldings rely on two basic operations: the projection $\pi_{c'}$, allowing to extract components from compound systems, and a composition operation \parallel . The latter will allow us to replace an interface X between components c and c' by another $\{c, c'\}$ -automaton (which, in general, will be a behavioural refinement of X). Thus, let \mathcal{A}, \mathcal{B} be tree-like nets meeting at $\{c, c'\}$, with X a (c, c') -gateway of \mathcal{A} and Y a (c', c) -gateway of \mathcal{B} . We present their composition in three steps:

1. building the synchronous product Z of X and Y ,
2. replacing X and Y by Z , and
3. merging \mathcal{A} and \mathcal{B} on Z .

We formalize these notions below. In a more general setting this composition would be the synchronous product of \mathcal{A} and \mathcal{B} . We present it this way to facilitate intuition on the distributed unfolding approach. For notational simplicity, we shall assume that the sets of places, transitions etc of $\mathcal{A}, \mathcal{B}, X, Y, Z$ are indexed with the name of the net, and denote with $N_{\mathcal{A}}, N_{\mathcal{B}}$ etc the sets of nodes (i.e. places and transitions) of \mathcal{A}, \mathcal{B} etc.

Synchronous product. The *synchronous product* of $\{c, c'\}$ -automata X and Y is the $\{c, c'\}$ -automaton $X \parallel Y = \langle P, T, F, M_0, \ell, \gamma \rangle$ with

- $P = P_X \times P_Y$ and $M_0 = M_{0,X} \times M_{0,Y}$,
- $T = \{ \langle t_1, t_2 \rangle \in T_X \times T_Y \mid \ell_X(t_1) = \ell_Y(t_2) \}$,
- $F = \{ \langle \langle x_1, x_2 \rangle, \langle x'_1, x'_2 \rangle \rangle \in P \times T \cup T \times P \mid (x_1, x'_1) \in F_X \wedge (x_2, x'_2) \in F_Y \}$,
- $\ell(\langle t_1, t_2 \rangle) = \ell_X(t_1) = \ell_Y(t_2)$, and $\gamma(\langle t_1, t_2 \rangle) = \{c, c'\}$.

Replacement. Let \mathcal{A} have a (c, c') -gateway X and $Z = X \parallel Y$ for some Y . The *replacement* of X by Z is $\mathcal{A}[X/Z] := \langle P', T', F', M'_0, \ell', \gamma' \rangle$, where, with $N''_{\mathcal{A}} := N_{\mathcal{A}} \setminus N_X$:

- $P' = (P_{\mathcal{A}} \setminus P_X) \cup P_Z$ and $T'_{\mathcal{A}} = (T_{\mathcal{A}} \setminus T_X) \cup T_Z$;
- $F' = (F_{\mathcal{A}} \cap N''_{\mathcal{A}} \times N''_{\mathcal{A}}) \cup F_Z$
 - $\cup \{ \langle n, \langle n_{\mathcal{A}}, n_{\mathcal{B}} \rangle \rangle \in N''_{\mathcal{A}} \times N_Z \mid \langle n, n_{\mathcal{A}} \rangle \in F_{\mathcal{A}} \}$
 - $\cup \{ \langle \langle n_{\mathcal{A}}, n_{\mathcal{B}} \rangle, n \rangle \in N_Z \times N''_{\mathcal{A}} \mid \langle n_{\mathcal{A}}, n \rangle \in F_{\mathcal{A}} \}$

- $M'_0 = (M_{0,\mathcal{A}} \setminus M_{0,X}) \cup M_{0,Z}$;
- for all $n \in N''_{\mathcal{A}}$, $\ell'(n) = \ell_{\mathcal{A}}(n)$ and $\gamma'(n) = \gamma_{\mathcal{A}}(n)$;
- for all $n \in N_Z$, $\ell'(n) = \ell_Z(n)$ and $\gamma'(n) = \{c, c'\}$.

The definition for the case where $Z = Y \parallel X$ is analogous.

Merging. Consider two tree-like nets \mathcal{A}, \mathcal{B} meeting at $\{c, c'\}$ such that Z is equally a (c, c') -gateway in \mathcal{A} and a (c', c) -gateway in \mathcal{B} . The *merging* of \mathcal{A} and \mathcal{B} on Z is the net $\langle P, T, F, M_0, \ell, \gamma \rangle$ with:

- $P = P_{\mathcal{A}} \cup P_{\mathcal{B}}$; $T = T_{\mathcal{A}} \cup T_{\mathcal{B}}$; $F = F_{\mathcal{A}} \cup F_{\mathcal{B}}$;
- $M_0 = M_{0,\mathcal{A}} \cup M_{0,\mathcal{B}}$;
- $\ell(n) = \ell_{\mathcal{A}}(n)$ for $n \in N_{\mathcal{A}} \setminus N_Z$, $\ell(n) = \ell_{\mathcal{B}}(n)$ for $n \in N_{\mathcal{B}} \setminus N_Z$, and $\ell(n) = \ell_Z(n)$ for $n \in N_Z$;
- $\gamma(n) = \gamma_{\mathcal{A}}(n)$ for $n \in N_{\mathcal{A}} \setminus N_Z$, $\gamma(n) = \gamma_{\mathcal{B}}(n)$ for $n \in N_{\mathcal{B}} \setminus N_Z$, and $\gamma(n) = \{c, c'\}$ for $n \in N_Z$.

Notice that this definition of step (3) relies on the fact that Z is exactly the same in \mathcal{A} and \mathcal{B} , so in particular the places and transitions have the same names. Moreover, it is easy to see that the resulting net is also tree-like.

Finally, from the above three steps, we can define the composition operation for compound systems: For $\mathcal{A}, \mathcal{B}, X, Y$ and $Z = X \parallel Y$ as above, we denote the *composition* of \mathcal{A} and \mathcal{B} as the merging of $\mathcal{A}[X/Z]$ and $\mathcal{B}[Y/Z]$ on Z .

Distributed unfolding. From now on, for simplicity of presentation, we shall consider a compound system pn with three components $\mathcal{A}, \mathcal{B}, \mathcal{C}$, giving rise to projections $\mathcal{A}, \mathcal{B}, \mathcal{C}$. We assume that \mathcal{A} and \mathcal{B} interact through interface X , while \mathcal{B} and \mathcal{C} interact through interface Y . Hence $\text{pn} = \mathcal{A} \parallel \mathcal{B} \parallel \mathcal{C}$. As described in [9], all the results given below for these three components extend directly to compound systems with tree-shaped interaction graphs.

The distributed unfolding of [9] is based on the following factorability properties:

$$\text{un}(\text{pn}) = \text{un}(\mathcal{A}) \parallel \text{un}(\mathcal{B}) \parallel \text{un}(\mathcal{C}), \quad (1)$$

$$= \pi_{\mathcal{A}}(\text{un}(\text{pn})) \parallel \pi_{\mathcal{B}}(\text{un}(\text{pn})) \parallel \pi_{\mathcal{C}}(\text{un}(\text{pn})). \quad (2)$$

The approach of [9] consists in computing the factors $\pi_{\mathcal{A}}(\text{un}(\text{pn}))$, $\pi_{\mathcal{B}}(\text{un}(\text{pn}))$, and $\pi_{\mathcal{C}}(\text{un}(\text{pn}))$ by local computations only, that is, without computing $\text{un}(\text{pn})$. These computations are strongly independent and thus can be distributed. One can remark that these factors, are exactly forming a distributed tester, provided that one can add correct time stamps to them.

The idea for distributed computing of the unfolding factors comes from the recursive application of equation (1) in (2):

$$\begin{aligned} \pi_{\mathcal{A}}(\text{un}(\text{pn})) &= \pi_{\mathcal{A}}(\text{un}(\mathcal{A}) \parallel \text{un}(\mathcal{B}) \parallel \text{un}(\mathcal{C})) \\ &= \text{un}(\mathcal{A}) \parallel \pi_{\mathcal{A}}(\text{un}(\mathcal{B}) \parallel \text{un}(\mathcal{C})) \\ &= \text{un}(\mathcal{A}) \parallel \pi_{\mathcal{A}}(\text{un}(\mathcal{B} \parallel \pi_{\mathcal{B}}(\text{un}(\mathcal{C})))) \end{aligned}$$

This lets one compute $\pi_A(\text{un}(\text{pn}))$ by propagating information from \mathcal{C} to \mathcal{A} : First \mathcal{C} computes $\pi_B(\text{un}(\mathcal{C}))$ and sends it to \mathcal{B} , which can use this to compute $\pi_A(\text{un}(\mathcal{B}) \parallel \pi_B(\text{un}(\mathcal{C})))$ and send it to \mathcal{A} . Then \mathcal{A} is able to compute $\pi_A(\text{un}(\text{pn}))$. A similar reasoning can be applied to the two other components: $\pi_C(\text{un}(\text{pn})) = \text{un}(\pi_C(\text{un}(\pi_B(\text{un}(\mathcal{A}) \parallel \mathcal{B}))) \parallel \mathcal{C})$, and $\pi_B(\text{un}(\text{pn})) = \text{un}(\pi_B(\text{un}(\mathcal{A}) \parallel \mathcal{B}) \parallel \text{un}(\mathcal{C}))$.

This approach is not yet constructive: in order to compute one unfolding factor, one requires the unfolding of other components, which are in general infinite. Thus, we wish to replace them by a finite object describing all the possible behaviours of the interface of this system. We call such an object an *interface summary*.

Consider a net \mathcal{A} having a gateway X to some component c . An *interface summary* of \mathcal{A} with respect to X is any automaton $\text{sum}_X(\mathcal{A}) = \langle P, T, F, M_0, \ell, \gamma \rangle$ satisfying the following: there exists a sequence of transition firings $t_1 \dots t_k$ starting from M_0 with label sequence $w = \ell(t_1) \dots \ell(t_k)$ if and only if there exists a sequence of transition firings $t'_1 \dots t'_k$ in $\pi_c(\text{un}(\mathcal{A}))$ starting from its initial state and having the same label sequence w . Notice that this implies, in particular, that all nodes of $\text{sum}_X(\mathcal{A})$ belong to c .

Example. Consider the component \mathcal{A} (consisting of the subnets A and X) of the net in Figure 2. An interface summary of \mathcal{A} w.r.t. X is shown in Figure 4 (left). Moreover, an interface summary of component \mathcal{B} w.r.t. the interface Y is shown in Figure 4 (right). Notice that the first summary is identical to X (when the latter is viewed as an automaton), whereas the second has a terminating behaviour; this is because, while Y is cyclic, B terminates after deliver_C .

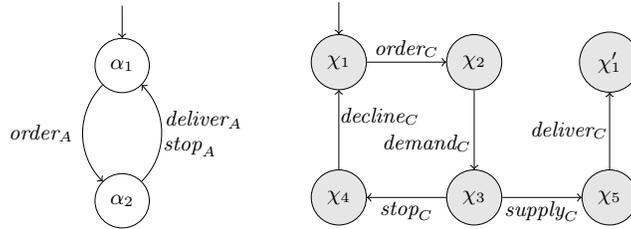


Fig. 4. Summary of \mathcal{A} w.r.t. its interface X (left); summary of \mathcal{B} w.r.t. Y (right).

Interface summaries can then replace unfoldings and projections in the above equations, leading to a new information propagation process:

$$\begin{aligned}
 \pi_A(\text{un}(\text{pn})) &= \text{un}(\mathcal{A} \parallel \pi_A(\text{un}(\mathcal{B} \parallel \pi_B(\text{un}(\mathcal{C})))) \\
 &= \text{un}(\mathcal{A} \parallel \text{sum}_X(\mathcal{B} \parallel \text{sum}_Y(\mathcal{C}))) \\
 \pi_B(\text{un}(\text{pn})) &= \text{un}(\text{sum}_X(\mathcal{A}) \parallel \mathcal{B} \parallel \text{sum}_Y(\mathcal{C})) \\
 \pi_C(\text{un}(\text{pn})) &= \text{un}(\text{sum}_Y(\text{sum}_X(\mathcal{A}) \parallel \mathcal{B}) \parallel \mathcal{C})
 \end{aligned}$$

Example. We illustrate the computation of $\pi_{\mathcal{C}}(\text{un}(\text{pn}))$ using the net from Figure 2. As mentioned in the previous example, $\text{sum}_X(\mathcal{A})$ is identical to X itself, therefore, in our example, $\text{sum}_X(\mathcal{A}) \parallel \mathcal{B}$ is the same as \mathcal{B} . In turn, $\text{sum}_Y(\mathcal{B})$ is shown in Figure 4 (right). The result of unfolding the composition of this summary with \mathcal{C} is shown in Figure 5, where places representing the summary are shown in grey.

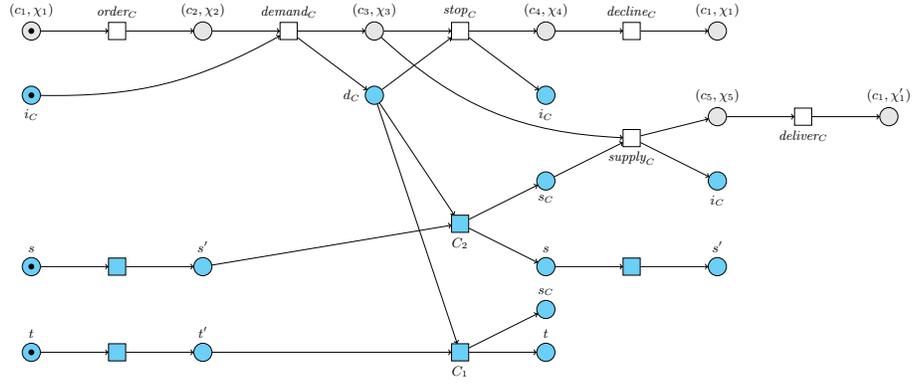


Fig. 5. Unfolding of the composition of \mathcal{C} with the summary of \mathcal{B} w.r.t. Y .

Notice that, for instance, the above computation of $\pi_{\mathcal{C}}(\text{un}(\text{pn}))$ finally unfolds some composition of \mathcal{C} with summaries of X and Y . However, the latter do not contain any timing information on components A and B . Thus, this method cannot be directly used to compute a distributed tester for co-ioco conformance because the latter requires timing information for all components. In the remainder of the paper, we show how to tackle this problem.

6 Distributed time stamps computation

We now study how to incorporate timing information into distributed unfoldings. The distributed approach from Section 5 relies on unfolding one single component, composed with a summary of other components. However, one action in a summary may represent multiple actions in several components. For instance, in the summary of \mathcal{A} from Figure 4 (left), the action deliver_A requires multiple transition firings in \mathcal{A} (cf Figure 2 and Figure 3). A natural way out of this dilemma is to consider the unfolding of timed nets whose vector clocks convey the missing information. In this sense, vector clocks represent *time increments*.

Let $\overline{\text{pn}} = \langle \text{pn}, \theta \rangle$ be a timed net. A *timed branching process* of $\overline{\text{pn}}$ is a pair $\langle \text{pn}', \theta' \rangle$, where pn' is a branching process of pn , and for all events e , $\theta'(e)(c) = \sum_{e' \in [e]} \theta(\lambda(e'))(c)$. We denote $\overline{\text{un}}(\overline{\text{pn}})$ as the *timed unfolding* of $\overline{\text{pn}}$. Notice that the timed unfolding of the untimed net pn , $\overline{\text{un}}(\text{pn})$, corresponds to the timed

unfolding of $\langle \text{pn}, \theta_1 \rangle$, where $\theta_1(t)(c) = 1$ if $c \in \lambda(t)$ and 0 otherwise. We therefore denote $\text{pn}_1 := \langle \text{pn}, \theta_1 \rangle$.

In the rest of this section, we revisit the material from Section 5 and extend it with timed information. This results in a distributed method for computing distributed testers, subject to finding appropriate summaries. This final problem is then solved in Section 7.

Timed merging. Consider two timed nets $\langle \mathcal{A}, \theta_A \rangle$ and $\langle \mathcal{B}, \theta_B \rangle$ meeting at $\{c, c'\}$, where \mathcal{A} has a (c, c') -gateway X , and \mathcal{B} has a (c', c) -gateway Y . Then $\langle \mathcal{A}, \theta_A \rangle \parallel \langle \mathcal{B}, \theta_B \rangle = \langle \mathcal{A} \parallel \mathcal{B}, \theta \rangle$, where for every transition t of $\mathcal{A} \parallel \mathcal{B}$, we have the following:

- if $t = (t_a, t_b)$ is a transition of $X \parallel Y$, then $\theta(t)(c) = \theta_A(t_a)(c)$ for $c \in \mathbb{C}[\mathcal{A}, X]$ and $\theta(t)(c) = \theta_B(t_b)(c)$ for other components c ;
- otherwise, $\theta(t) = \theta_A(t)$ (resp. $\theta_B(t)$) if t is a transition of \mathcal{A} (resp. \mathcal{B}).

Timed summaries. To make the distributed approach practical, a notion of timed interface summary is also necessary. Consider a timed net $\langle \mathcal{A}, \theta_A \rangle$ with gateway X to some component c . A *timed interface summary* of \mathcal{A} w.r.t. X is any timed net $\overline{\text{sum}}_X(\mathcal{A}) = \langle \text{sum}_X(\mathcal{A}), \theta_X \rangle$ such that $\text{sum}_X(\mathcal{A})$ is an interface summary of \mathcal{A} w.r.t. X , and for any $c' \in \mathbb{C}[\mathcal{A}, X]$ and any path $t_1 \dots t_n$ with label sequence w in $\text{sum}_X(\mathcal{A})$, $\sum_{i=1}^n \theta_X(t_i)(c')$ is the minimal number of transitions from c' among all sequences of transitions in \mathcal{A} whose labels contain w as a subsequence. Notice that the sequence achieving the minimum for some component c' is not necessarily the same as for another component c'' .

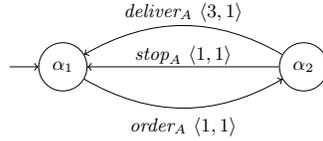


Fig. 6. Timed interface summary of \mathcal{A} w.r.t. interface X .

Example. Consider the component \mathcal{A} of the net in Figure 2 and its interface X . Figure 6 shows a timed summary of \mathcal{A}_1 w.r.t. X , where actions are annotated with time increments over A and B . This summary can be seen as a transformation of the projection shown in Figure 3, where conditions mapped to a_1 are fused together⁴. Time increments correspond to the differences in the time stamps between e_6, e_8, e_9 , respectively, and e_3 in Figure 3. Since the two occurrences of deliver_A in Figure 3 correspond to different time increments, a

⁴ In general, the rule for fusing conditions will be more complicated, see Section 7.

minimum has to be taken to obtain the time increment of $deliver_A$ in Figure 6. Over A the minimum of 3 (obtained from e_8) and 4 (obtained from e_9) gives 3. Similarly, over B the minimum of 1 and 1 gives 1.

By making use of this extension of \parallel and of timed summaries, we can extend the distributed unfolding algorithm of Section 5 to build distributed testers. This is expressed (in the case of a three components) in the following theorem, which is the first part of our contribution.

Theorem 1. *If summaries with time increments exist, then the propagation equations hold:*

$$\begin{aligned}\bar{\pi}_A(\bar{\text{un}}(\text{pn})) &= \bar{\text{un}}(\mathcal{A}_1 \parallel \bar{\text{sum}}_X(\mathcal{B}_1 \parallel \bar{\text{sum}}_Y(\mathcal{C}_1))) \\ \bar{\pi}_B(\bar{\text{un}}(\text{pn})) &= \bar{\text{un}}(\bar{\text{sum}}_X(\mathcal{A}_1) \parallel \mathcal{B}_1 \parallel \bar{\text{sum}}_Y(\mathcal{C}_1)) \\ \bar{\pi}_C(\bar{\text{un}}(\text{pn})) &= \bar{\text{un}}(\bar{\text{sum}}_Y(\bar{\text{sum}}_X(\mathcal{A}_1) \parallel \mathcal{B}_1) \parallel \mathcal{C}_1)\end{aligned}$$

Proof (Sketch of the proof). We consider the particular case of \mathcal{A} . The principle of the proof would be similar for any component in our particular example, as well as for any component in a more complex system, provided its interaction graph is a tree.

The statement clearly holds for the untimed aspects, that is, considering only the sequences of transition firing that are possible. Indeed, the structure of the summaries as well as the definition of the product is exactly the same than in the case without time. The proof of [9] can thus be used. This also allows to identify all the elements (and in particular, the events) of $\pi_A(\text{un}(\text{pn}))$ and $\text{un}(\mathcal{A} \parallel \text{sum}_X(\mathcal{B} \parallel \text{sum}_Y(\mathcal{C})))$, except the time stamps. Let us denote by θ_π the time stamps from $\bar{\pi}_A(\bar{\text{un}}(\text{pn}))$ and by θ_U the time stamps in the timed unfolding $\bar{\text{un}}(\mathcal{A}_1 \parallel \bar{\text{sum}}_X(\mathcal{B}_1 \parallel \bar{\text{sum}}_Y(\mathcal{C}_1)))$. Consider an event e from these two objects and a component c . Looking at how θ_π is computed one gets

$$\theta_\pi(e)(c) = \sum_{e' \in [e]} \theta_1(\lambda(e'))(c)$$

where θ_1 are the clock vectors of pn_1 (this is a direct application of the definition of the branching processes and the definition of the projection). Looking at how θ_U is computed one gets

$$\theta_U(e)(c) = \sum_{e' \in [e]} \theta_A(\lambda(e'))(c) + \theta_X(\lambda(e'))(c)$$

where θ_A gives the time increments in \mathcal{A}_1 and θ_X gives the time increments in $\bar{\text{sum}}_X(\mathcal{B}_1 \parallel \bar{\text{sum}}_Y(\mathcal{C}_1))$. The proof in fact mainly consists in linking θ_X to θ_B and θ_C , the time increments in \mathcal{B}_1 and \mathcal{C}_1 respectively. Indeed, for any event e corresponding to a transition t from X , by definition of timed summaries, one gets that

$$\sum_{e' \in [e]} \theta_X(\lambda(e'))(c) = 0$$

if $c \in \mathbb{C}[\mathcal{A}, X]$, and that otherwise $\sum_{e' \in [e]} \theta_X(\lambda(e'))(c)$ is the total number of transitions to be fired before e in c . This allows to identify $\theta_U(e)(c)$ (as a sum of θ_A , θ_B , and θ_C) and $\theta_\pi(e)(c)$ (in terms of $\theta_{\mathbf{1}}$, which is itself a sum of the three same time increments).

A complete proof would have a closer look at θ_U , using the explicit construction of the summaries described below. These summaries are indeed computed from unfoldings, allowing to identify any event of $\overline{\text{un}}(\text{pn})$ to an event in $\overline{\text{un}}(\mathcal{A}_{\mathbf{1}} \parallel \overline{\text{sum}}_X(\mathcal{B}_{\mathbf{1}} \parallel \overline{\text{sum}}_Y(\mathcal{C}_{\mathbf{1}})))$ as above. This event would itself be identified to a couple of events from the unfolding of $\mathcal{A}_{\mathbf{1}}$ and that of $\mathcal{B}_{\mathbf{1}} \parallel \overline{\text{sum}}_Y(\mathcal{C}_{\mathbf{1}})$. In turn, this couple would be identified to a triple of events in the unfolding of $\mathcal{A}_{\mathbf{1}}$, $\mathcal{B}_{\mathbf{1}}$, and $\mathcal{C}_{\mathbf{1}}$. Linking the time stamps/increments of these different events would allow to conclude, with similar reasoning as the one drawn in the sketch of the proof of Theorem 2 below.

7 Interface summary construction

We now provide the missing part of the puzzle by presenting a method for computing timed summaries. This allows to effectively construct the distributed unfoldings mentioned in Theorem 1. Our method is a modification of the – untimed – summary construction from [3], which we recall in Section 7.1. We then explain how to deal with time increments in Section 7.2.

7.1 Summary construction without time increments

Consider a net \mathcal{A} having a (c, c') -gateway X . We recall how to construct an interface summary of \mathcal{A} with respect to X from a finite prefix of $\text{un}(\mathcal{A})$. To this end, we first introduce some notations.

Let pn' be a branching process of \mathcal{A} . An event (condition) n of pn' is an X -event (X -condition) if $\gamma(n) = \{c, c'\}$. Let e be any event of pn' . We note $M(e)$ the unique set of conditions marked after firing all events in $[e]$ and $St(e) = \{ \lambda(b) \mid b \in M(e) \}$ the places of \mathcal{A} associated with $M(e)$. We note $M(e)_X$ the unique X -condition in $M(e)$. Since by assumption X is an automaton, such a condition always exists. We note $Xp(e)$ the set of X -predecessors of e , that is the X -events among the causal predecessors of e : $Xp(e) = \{ e' \in [e] \mid \gamma(e') = \{c, c'\} \}$. Event e' is a *strong cause* of e , denoted $e' \ll e$, if $e' < e$ and $b' < b$ for every $b \in M(e) \setminus M(e')$, $b' \in M(e') \setminus M(e)$.

Algorithm 1 describes the construction of the interface summary in two steps. The first step (lines 1 to 10) computes a prefix of $\text{un}(\mathcal{A})$ containing sufficient information to construct a summary, which is produced by the second step (lines 11 to 14).

The first step relies on two notions: *cut-off events* (after which the unfolding contains no additional information useful for us) and *cut-off candidates* (where we provisionally stop unfolding but may resume later on). We define both using the notations of the algorithm:

An event e is a cut-off of pn' if it is an X -event and pn' already contains a non-cut-off X -event e' (called *companion* of e) such that $St(e) = St(e')$.

Let $Xco_{\text{pn}'}(e)$ denote the set of non cut-off X -events of pn' concurrent with e . Then event e is a cut-off candidate of pn' if it is not an X -event and pn' contains $e' \ll e$ such that $St(e) = St(e')$, $Xp(e') = Xp(e)$, and $Xco_{\text{pn}'}(e) \subseteq Xco_{\text{pn}'}(e')$. Finally, we say that event e *frees* e_c if e_c is a cut-off candidate of pn' before the addition of e but not after its addition.

Algorithm 1 Summary of a net \mathcal{A} with interface X

```

1: let  $\text{pn}'$  be the branching process of  $\mathcal{A}$  with no events
2: let  $co = \emptyset$  and  $coc = \emptyset$ 
3: While  $Ext(\text{pn}', co, coc) \neq \emptyset$  do
4:   choose an event  $e$  in  $Ext(\text{pn}', co, coc)$ 
5:   If  $e$  is a cut-off event then let  $co = co \cup \{e\}$ 
6:   Elseif  $e$  is a cut-off candidate of  $\text{pn}'$  then
7:     let  $coc = coc \cup \{e\}$ 
8:   Else for every  $e' \in coc$  do
9:     If  $e$  frees  $e'$  then  $coc = coc \setminus \{e'\}$ 
10:  extend  $\text{pn}'$  with  $e$ 
11: let  $\text{aut} := \pi_{c'}(\text{pn}')$ 
12: For every  $e \in co$  with companion  $e'$  do
13:   fuse  $M(e)_X$  with  $M(e')_X$  in  $\text{aut}$ 
14: Return  $\text{aut}$ 

```

In the first step of Algorithm 1, $Ext(\text{pn}', co, coc)$ denotes possible extensions of pn' that are not causal successors of events in $co \cup coc$. The choice of e in this set has to be done carefully, respecting a well chosen order (see [2] for example).

The second step of Algorithm 1 first extracts the interface portion of pn' by projecting onto c' (this suffices because X is a gateway). Moreover, since by assumption X is an automaton, and because of the properties of branching processes, $\pi_{c'}(\text{pn}')$ is an acyclic finite automaton. In fact, each terminal node of $\pi_{c'}(\text{pn}')$ is an X -condition b such that the unique event $e \in \bullet b$ is a cut-off. The cut-off condition ensures that $b' := M(e')_X$, where e' is the companion of e , satisfies $\lambda(b') = \lambda(b)$, and, since $St(e) = St(e')$, b and b' have the same future: isomorphic structures would be built from $M(e)$ and $M(e')$ if the unfolding process was never stopped. This justifies fusing b and b' as one single place in lines 12 and 13.

7.2 Adding time increments

Now, let $\langle \mathcal{A}, \theta_A \rangle$ be a timed net with \mathcal{A} having gateway X as in Section 7.1. We shall produce a timed interface summary of \mathcal{A} w.r.t. X by applying the following modification to Algorithm 1:

The unfolding step (lines 1 to 10) and the fusion of conditions (lines 12 and 13) remain unchanged. However, it does not suffice to simply annotate each

event e in aut with the time increment given by $\theta_A(\lambda(e))$: to obtain the correct time stamp for e , one would have to sum all the time increments from $[e]$ in pn' . This use of time increments rather than time stamps allows one to build correct timed interface summaries.

Thus, to compute the time increments θ_X , we have to take into account the events in pn' outside X that were removed by the projection. Notice that each X -event e can be associated to a unique minimal set $\text{Req}(e)$ of non X -events that have to fire in order to enable e . This set is constituted of all predecessors of e that are not X -predecessors of e , nor predecessors of X -predecessors of e . In other words, it consists of all the events that have to occur between the closest X -predecessor of e and e itself. The time increment associated to e is then:

$$\forall c, \theta_X(e)(c) = \theta_A(\lambda(e))(c) + \sum_{e' \in \text{Req}(e)} \theta_A(\lambda(e'))(c).$$

Moreover, if the fusion of two conditions in line 13 results in two or more automata transitions having the same (singleton) preset, label, and postset, these transitions are fused into one single transition whose time increment is the point-wise minimum of the time increments of the fused transitions.

Example. For Figure 2, the timed summary of \mathcal{A} w.r.t. X produced by this procedure is the one shown in Figure 6.

Theorem 2. *The tuple $\langle \text{aut}, \theta_X \rangle$, as computed by the above modification of Algorithm 1, is a timed interface summary of $\langle \mathcal{A}, \theta_A \rangle$.*

Proof (Sketch of the proof). The correctness of the summary without time increments is due to the use of the construction algorithm from [3].

We now discuss the correctness of the time-increment computation. We illustrate it using the particular case of our running example, when computing the timed summary of $\mathcal{B}' := \mathcal{B} \parallel \overline{\text{sum}}_Y(\mathcal{C})$ w.r.t. X . However, the argument generalizes to any interface summary computation on tree-like nets by noticing that removing any edge from a tree (here between A and B) separates it into two disjoint sub-trees (here A alone and B and C together).

The correctness of the time increments in the particular case of the summary of $\mathcal{B}' := \mathcal{B} \parallel \overline{\text{sum}}_Y(\mathcal{C})$ in our running example comes from the fact that the components in the sub-tree rooted at B can all be considered independently. Indeed, all the timing information about other components than B can only appear on transitions from the interfaces of B with its neighbours. Moreover, for any component $C \neq B$ (interacting through interface Y), timing information on C can only appear on Y -transitions. Since Y is an automaton, no two concurrent actions in the unfolding of \mathcal{B}' can have non-zero timing information on \mathcal{C} . Thus, given an X -event e , the events e' in $\text{Req}(e)$ with $\theta(e')(C) \neq 0$ can be totally ordered. Thus, they are always executed in sequence, which guarantees that they can be added without ambiguity.

From that, the validity of the interface summary construction in systems with costs of [3] guarantees that, for component C and any path $t_1 \dots t_k$ in the

interface summary computed by our algorithm, one has that $\sum_{t_i} \theta_X(t_i)(C)$ is the minimum number of transitions from C that have to be fired in order to fire $t_1 \dots t_n$ in this order. This is the property that we want to achieve for the time increments in each component.

Finally, the fact that the time increments are computed independently for each component guarantees the validity of our interface summary.

8 Conclusion

In this paper we have proposed a procedure for building a tester for distributed systems. This tester is distributed as it complies with the definition given in [8]. The novelty of our approach with respect to this work on concurrent conformance is that the construction of the tester is achieved as the result of a distributed process. The main interest of our approach is thus that it can avoid the construction of a prefix of the unfolding of the Petri net representation of the full distributed system under test. Instead, the Petri net representation of each component of the system is unfolded separately. This allows to perform the generally costly computation of unfolding prefixes on small nets. Moreover, such a distributed approach has another practical interest. By definition, it allows for building distributed test suits for a distributed system, even without having a complete view of this system. It thus extends one of the interest of distributed testers (being able to test a system from local views of its components) to the construction process of the tester itself.

Our work on distributed computation of distributed testers heavily relies on previous theoretical works from the authors. However, the new theoretical contribution associated to it is not negligible. We believe that this theoretical contribution is of interest all by itself, for two reasons. First, it unifies the results from [9] and [3]. This makes the distributed unfolding technique of the former effective, by providing a concrete way to build the interface summaries. Second, and mainly, this theoretical contribution shows that, during the distributed unfolding process, the local unfolding of any component \mathcal{A} can be equipped with time stamps on its events. This information gives the minimum number of transitions that must fire in the other components for any event of the unfolding of \mathcal{A} to happen. In other words, this brings a notion of logical time to Petri nets unfolding in the context of distributed systems modeling. The additional computational cost for building and propagating these time stamps along the distributed unfolding procedure is relatively small with respect to the cost of building (a prefix of) the unfolding itself.

References

1. Athanasiou, K., Ponce de León, H., Schwoon, S.: Test case generation for concurrent systems using event structures. In: TAP. pp. 19–37 (2015)
2. Esparza, J., Heljanko, K.: Unfoldings – A Partial-Order Approach to Model Checking. Springer (2008)

3. Esparza, J., Jezequel, L., Schwoon, S.: Computation of summaries using net unfoldings. In: FSTTCS. pp. 225–236 (2013)
4. Fabre, E.: Bayesian Networks of Dynamic Systems. Habilitation, Université Rennes 1 (June 2007)
5. Fidge, C.J.: Timestamps in message-passing systems that preserve the partial ordering. Australian Computer Science Communications 10(1), 56–66 (1988)
6. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21(7), 558–565 (1978)
7. Ponce de León, H., Haar, S., Longuet, D.: Unfolding-based test selection for concurrent conformance. In: ICTSS. pp. 98–113 (2013)
8. Ponce de León, H., Haar, S., Longuet, D.: Distributed testing of concurrent systems: Vector clocks to the rescue. In: ICTAC. pp. 369–387 (2014)
9. Madalinski, A., Fabre, E.: Modular construction of finite and complete prefixes of Petri net unfoldings. Fundamenta Informaticae 95(1), 219–244 (2009)
10. Mattern, F.: Virtual time and global states of distributed systems. In: International Workshop on Parallel and Distributed Algorithms. pp. 215–226 (1988)
11. Tretmans, J.: Testing concurrent systems: A formal approach. In: CONCUR. pp. 46–65 (1999)