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

Loïg Jezequel * **Agnes Madalinski** ** **Stefan Schwoon** ***

* *Université de Nantes, LS2N, UMR CNRS 6004, France*
(e-mail: loig.jezequel@ls2n.fr)
** *Chair of Software Engineering, OvGU Magdeburg, Germany*
(e-mail: agnes.madalinski@ovgu.de)
*** *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.

Keywords: Petri-nets, Discrete time, Concurrent systems, Distributed models, Test generation

1. INTRODUCTION

The **co-ioco** framework proposed in Ponce de León et al. (2013) introduced partial-order semantics to the well-known **ioco** theory of Tretmans (1999). 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 Ponce de León et al. (2013); Athanasiou et al. (2015) 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, Ponce de León et al. (2014) 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 Madalinski and Fabre (2009) and Esparza et al. (2013), extending them with timed information. Our timed information is *logical*: it counts occurences of actions. This is similar to vector clocks in distributed systems, cf. Lamport (1978); Fidge (1988); Mattern (1988).

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. 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. Proofs can be found in Jezequel et al. (2018).

2. RUNNING EXAMPLE

The Petri net depicted in Figure 1 shows the interaction of a consumer (middle) with two producers (left and right). We will consider two decompositions of it:

(1) as the five *subnets* A, C (producers), B (consumer), and X, Y (producers/consumer interfaces);
(2) as three (overlapping) *components* $\mathcal{A}$ (A with X), $\mathcal{B}$ (B with X and Y), and $\mathcal{C}$ (C with Y).

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

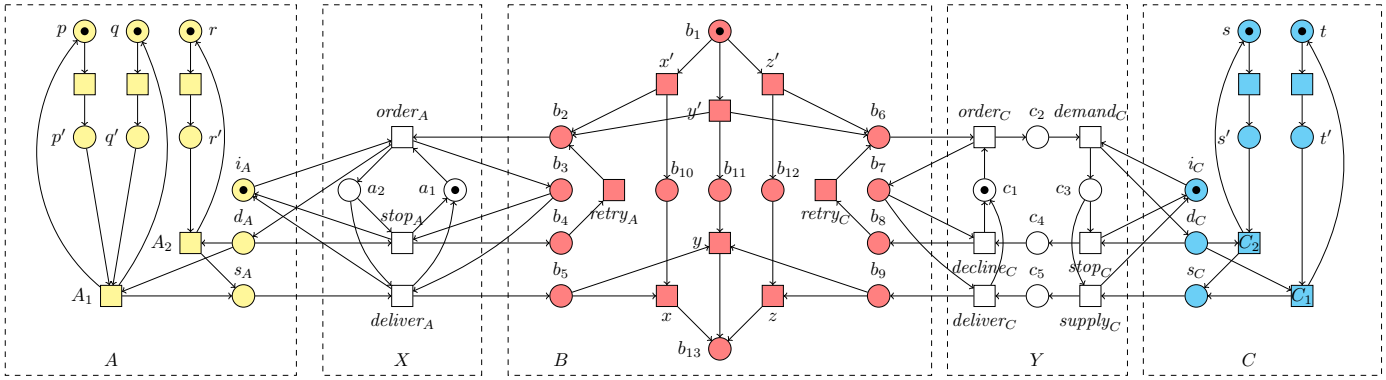


Fig. 1. An example with one consumer B and two producers A, C connected by interfaces X, Y.

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 $\mathrm{pn} = \langle P, T, F, M_0, \ell, \gamma \rangle$ 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*, $\ell : (P \cup T) \to \Sigma$ associates labels to nodes, and $\gamma : (P \cup T) \to 2^{\mathbb{C}}$ 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 1 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 $\mathbb{C} = \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$, corresponding to $\mathcal{A}, \mathcal{B}, \mathcal{C}$ from Section 2.

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

Projections and interfaces. Let $\mathbb{C}' \subseteq \mathbb{C}$. The *projection of* pn *to* $\mathbb{C}'$ is the net $\pi_{\mathbb{C}'}(\mathrm{pn}) = \langle P', T', F', M_0', \ell', \gamma' \rangle$ 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 1, we obtain $\mathcal{A} = \pi_{\mathsf{A}}(\mathrm{pn})$, $\mathcal{B} = \pi_{\mathsf{B}}(\mathrm{pn})$, and $\mathcal{C} = \pi_{\mathsf{C}}(\mathrm{pn})$. 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 $\pi_c(\pi_{c'}(\mathrm{pn})) = \pi_{c'}(\pi_c(\mathrm{pn}))$. For instance, the interface of A, B in Figure 1 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, $\mathcal{A} = \pi_{\mathsf{A}}(\mathrm{pn})$ 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. In certain cases we will include (logical) timing information in Petri nets. A *vector clock* is a mapping from $\mathbb{C}$ to $\mathbb{N}$; it shall count how many events have fired per component. A *timed net* is a pair $\overline{\mathrm{pn}} = \langle \mathrm{pn}, \theta \rangle$, equipping a Petri net pn with a *clock mapping* θ that associates a vector clock to each transition of pn. The projection of $\overline{\mathrm{pn}}$ to $\mathbb{C}' \subseteq \mathbb{C}$ is $\overline{\pi}_{\mathbb{C}'}(\overline{\mathrm{pn}}) := \langle \pi_{\mathbb{C}'}(\mathrm{pn}), \theta' \rangle$, where θ' is the restriction of θ to the nodes of $\pi_{\mathbb{C}'}(\mathrm{pn})$. 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 un(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 Ponce de León et al. (2014), 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.

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

- The net with conditions $q_0, \ldots, q_k$ so that $\lambda(q_0) = p_0, \ldots, \lambda(q_k) = p_k$, no events, and initial marking $\{q_0, \ldots, q_k\}$ 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 $\gamma'(x) = \gamma(\lambda(x))$.

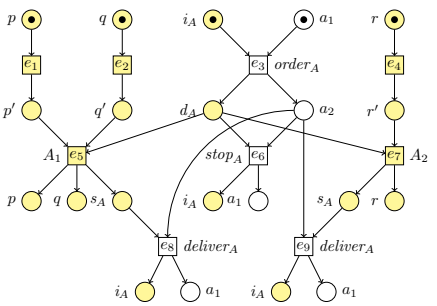
The set of all branching processes of a Petri net pn, finite and infinite, is defined by closing the finite branching processes under countable unions (see Esparza and Heljanko (2008)). In particular, the union of all finite branching processes is called the *unfolding* of pn, denoted un(pn).

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 Figure 2. It depicts a finite branching process of the net from Figure 1 restricted to component A, i.e. producer A and its interface X. The labelling λ is given next to the conditions and events. $\#_\mathsf{A}$ provides the value $\theta(e)(\mathsf{A})$ for every event e (ditto for B).

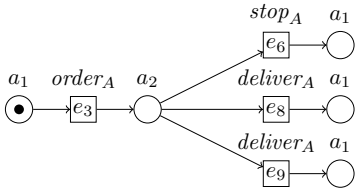


E	$\langle \#_\mathsf{A}, \#_\mathsf{B} \rangle$
e_1	$\langle 1, 0 \rangle$
e_2	$\langle 1, 0 \rangle$
e_3	$\langle 1, 1 \rangle$
e_4	$\langle 1, 0 \rangle$
e_5	$\langle 4, 1 \rangle$
e_6	$\langle 2, 2 \rangle$
e_7	$\langle 3, 1 \rangle$
e_8	$\langle 5, 2 \rangle$
e_9	$\langle 4, 2 \rangle$

Fig. 2. A branching process of π_A (pn), where pn is the net from Figure 1, and associated time stamps.

In the following, we consider projections of timed or untimed unfoldings to components $\mathbb{C}' \subseteq \mathbb{C}$. In this context, notice that $\overline{\pi}_{\mathbb{C}'}(\overline{\text{un}}(\text{pn}))$ is not the same as $\overline{\text{un}}(\pi_{\mathbb{C}'}(\text{pn}))$; 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 B. Only the conditions of X and the events in which it participates are preserved.



E	$\langle \#_\mathsf{A}, \#_\mathsf{B} \rangle$
e_3	$\langle 1, 1 \rangle$
e_6	$\langle 2, 2 \rangle$
e_8	$\langle 5, 2 \rangle$
e_9	$\langle 4, 2 \rangle$

Fig. 3. Projection of Figure 2 to component B.

5. DISTRIBUTED UNFOLDING

There exists a straightforward, centralized algorithm for computing (a finite prefix of) the unfolding of a net. It applies the inductive characterization of a branching processes, adding one event at a time (see Esparza and Heljanko (2008)). The result can be equipped with time stamps and used to produce distributed testers (Ponce de León et al. (2014)). However, the intermediate unfolding prefix can be much larger than those testers.

Here, we look at the approach of Madalinski and Fabre (2009). It computes unfolding prefixes component by component. We briefly describe a general version of this algorithm. This uses smaller prefixes, but adding time stamps is not straightforward (Section 6).

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 (required by the distributed unfolding approach of Madalinski and Fabre (2009)), (2) every (non-empty) interface in pn is an automaton (this is due to timing information, and based on Esparza et al. (2013)).

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 unfolding relies on two basic operations: the projection $\pi_{\mathbb{C}'}$ and a composition operation $\|$. 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. 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.

The *synchronous product* (step 1) 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 $\gamma(\langle t_1, t_2\rangle) = \{c, c'\}$.

Let $\mathcal{A}$ have a (c, c')-gateway X and $Z = X \parallel Y$ for some Y. The *replacement* (step 2) 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 $T_{\mathcal{A}}' = (T_{\mathcal{A}} \setminus T_X) \cup T_Z$; $F' = (F_{\mathcal{A}} \cap N_{\mathcal{A}}'' \times N_{\mathcal{A}}'') \cup F_Z \;\cup \{\, \langle n, \langle n_{\mathcal{A}}, n_B\rangle\rangle \in N_{\mathcal{A}}'' \times N_Z \mid \langle n, n_{\mathcal{A}}\rangle \in F_{\mathcal{A}} \,\} \cup \{\, \langle\langle n_{\mathcal{A}}, n_B\rangle, n\rangle \in N_Z \times N_{\mathcal{A}}'' \mid \langle n_{\mathcal{A}}, n\rangle \in F_{\mathcal{A}} \,\}$ $M_0' = (M_{0,\mathcal{A}} \setminus M_{0,X}) \cup M_{0,Z}$; 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 $Z = Y \| X$ is analogous.

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* (step 3) 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.

From now on, for simplicity of presentation, we shall consider a compound system pn with three components $\mathsf{A}, \mathsf{B}, \mathsf{C}$, 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 $\mathrm{pn} = \mathcal{A} \parallel \mathcal{B} \parallel \mathcal{C}$. All the results given below for three components extend to tree-like nets (Madalinski and Fabre (2009)).

The distributed unfolding of Madalinski and Fabre (2009) is based on the following factorability properties:

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

$$= \pi_{\mathsf{A}}(\mathrm{un}(\mathrm{pn})) \parallel \pi_{\mathsf{B}}(\mathrm{un}(\mathrm{pn})) \parallel \pi_{\mathsf{C}}(\mathrm{un}(\mathrm{pn}))\,. \tag{2}$$

Their approach consists in computing, without computing un(pn), the factors $\pi_{\mathsf{A}}(\mathrm{un}(\mathrm{pn}))$, $\pi_{\mathsf{B}}(\mathrm{un}(\mathrm{pn}))$, and $\pi_{\mathsf{C}}(\mathrm{un}(\mathrm{pn}))$. The idea for this distributed computing of the unfolding factors comes from the recursive application of equation (1) in (2):

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

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

This approach is not yet constructive: computing one unfolding factor requires the (in general infinite) unfolding of other components. One has to replace them by a finite object representing the behaviour of the interface of this system: 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 $\mathrm{sum}_X(\mathcal{A}) = \langle P, T, F, M_0, \ell, \gamma\rangle$ satisying 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 $\pi_c(\mathrm{un}(\mathcal{A}))$ starting from its initial state and having the same label sequence w. Notice that this implies, in particular, that all nodes of $\mathrm{sum}_X(\mathcal{A})$ belong to c.

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

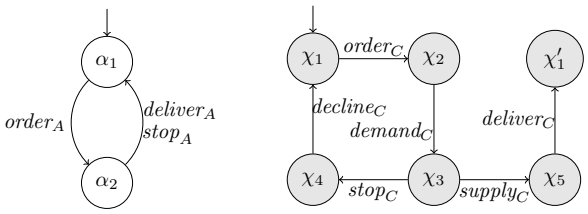


Fig. 4. Summary of $\mathcal{A}$ w.r.t. its interface X (left); summary of $\mathcal{B}$ w.r.t. Y (right).

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

$$\begin{aligned}\pi_{\mathsf{A}}(\mathrm{un}(\mathrm{pn})) &= \mathrm{un}(\mathcal{A} \parallel \pi_{\mathsf{A}}(\mathrm{un}(\mathcal{B} \parallel \pi_{\mathsf{B}}(\mathrm{un}(\mathcal{C})))))\\ &= \mathrm{un}(\mathcal{A} \parallel \mathrm{sum}_X(\mathcal{B} \parallel \mathrm{sum}_Y(\mathcal{C})))\\ \pi_{\mathsf{B}}(\mathrm{un}(\mathrm{pn})) &= \mathrm{un}(\mathrm{sum}_X(\mathcal{A}) \parallel \mathcal{B} \parallel \mathrm{sum}_Y(\mathcal{C}))\\ \pi_{\mathsf{C}}(\mathrm{un}(\mathrm{pn})) &= \mathrm{un}(\mathrm{sum}_Y(\mathrm{sum}_X(\mathcal{A}) \parallel \mathcal{B}) \parallel \mathcal{C})\end{aligned}$$

Example. We illustrate the computation of $\pi_{\mathsf{C}}(\mathrm{un}(\mathrm{pn}))$ on the net of Figure 1. As previously mentioned, $\mathrm{sum}_X(\mathcal{A})$ is identical to X, therefore, $\mathrm{sum}_X(\mathcal{A}) \parallel \mathcal{B}$ is the same as $\mathcal{B}$. In turn, $\mathrm{sum}_Y(\mathcal{B})$ is shown in Figure 4 (right). The result of unfolding the composition of it with $\mathcal{C}$ is shown in Figure 5 (places representing the summary are in grey).

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, firing one action from a summary may imply to fire multiple actions in several components. To represent that, one can use vector clocks. In this sens, vector clocks represent *time increments*.

Let $\overline{\mathrm{pn}} = \langle \mathrm{pn}, \theta\rangle$ be a timed net. A *timed branching process* of $\overline{\mathrm{pn}}$ is a pair $\langle \mathrm{pn}', \theta'\rangle$, where pn' is a branching process

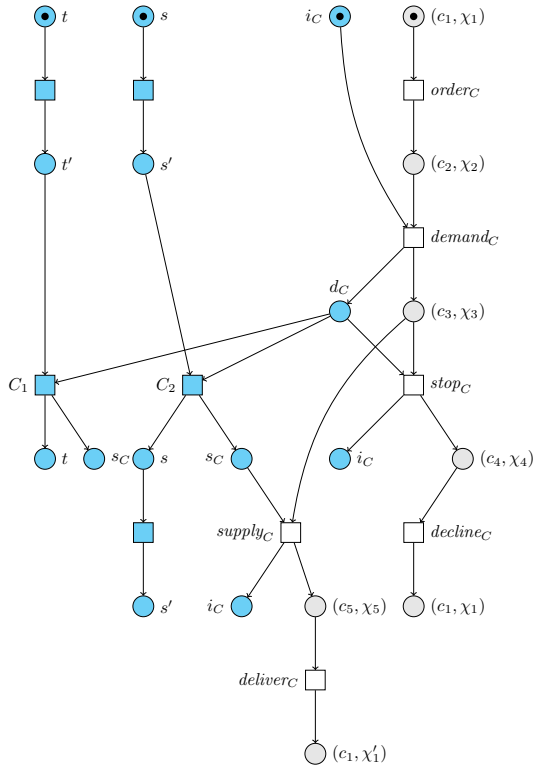


Fig. 5. Unfolding of the composition of $\mathcal{C}$ with the summary of $\mathcal{B}$ w.r.t. Y.

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 $\langle \text{pn}, \theta_{\mathbf{1}} \rangle$, where $\theta_{\mathbf{1}}(t)(c) = 1$ if $c \in \lambda(t)$ and 0 otherwise. We therefore denote $\text{pn}_{\mathbf{1}} := \langle \text{pn}, \theta_{\mathbf{1}} \rangle$.

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.

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. $\mathcal{B}$).

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''.

Example. Consider the component $\mathcal{A}$ of the net in Figure 1 and its interface X. Figure 6 shows a timed summary of $\mathcal{A}_{\mathbf{1}}$ w.r.t. X. 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 together. 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 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 gives 1.

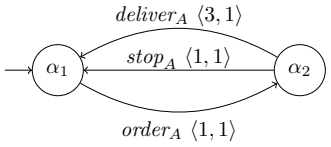


Fig. 6. Timed interface summary of $\mathcal{A}$ w.r.t. interface X.

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:

$$\overline{\pi}_{\mathsf{A}}\left(\overline{\text{un}}(\text{pn})\right) = \overline{\text{un}}(\mathcal{A}_{\mathbf{1}} \parallel \overline{\text{sum}}_X(\mathcal{B}_{\mathbf{1}} \parallel \overline{\text{sum}}_Y(\mathcal{C}_{\mathbf{1}})))$$
$$\overline{\pi}_{\mathsf{B}}\left(\overline{\text{un}}(\text{pn})\right) = \overline{\text{un}}(\overline{\text{sum}}_X(\mathcal{A}_{\mathbf{1}}) \parallel \mathcal{B}_{\mathbf{1}} \parallel \overline{\text{sum}}_Y(\mathcal{C}_{\mathbf{1}}))$$
$$\overline{\pi}_{\mathsf{C}}\left(\overline{\text{un}}(\text{pn})\right) = \overline{\text{un}}(\overline{\text{sum}}_Y(\overline{\text{sum}}_X(\mathcal{A}_{\mathbf{1}}) \parallel \mathcal{B}_{\mathbf{1}}) \parallel \mathcal{C}_{\mathbf{1}})$$

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.Our method is a modification of the – untimed – summary construction from Esparza et al. (2013).

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 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).

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 $Xco_{\mathrm{pn}'}(e)$ 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 $Xco_{\mathrm{pn}'}(e) \subseteq Xco_{\mathrm{pn}'}(e')$. 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 pn′ be the branching process of A with no events
 2: let co = ∅ and coc = ∅
 3: While Ext(pn′, co, coc) ≠ ∅ do
 4:     choose an event e in Ext(pn′, co, coc)
 5:     If e is a cut-off event then let co = co ∪ {e}
 6:     Elseif e is a cut-off candidate of pn′ then
 7:         let coc = coc ∪ {e}
 8:     Else for every e′ ∈ coc do
 9:         If e frees e′ then coc = coc \ {e′}
10:     extend pn′ with e
11: let aut := π_{c′}(pn′)
12: For every e ∈ co with companion e′ do
13:     fuse M(e)_X with M(e′)_X in aut
14: Return aut
```

In the first step of Algorithm 1, $Ext(\mathrm{pn}', co, coc)$ 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 Esparza and Heljanko (2008) for example). The second step 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, $\pi_{c'}(\mathrm{pn}')$ is an acyclic finite automaton. In fact, each terminal node of $\pi_{c'}(\mathrm{pn})$ 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.

Now, let $\langle \mathcal{A}, \theta_A \rangle$ be a timed net with $\mathcal{A}$ (still having gateway X). 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 $Req(e)$ 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: $\forall c, \theta_X(e)(c) = \theta_A(\lambda(e))(c) + \sum_{e' \in Req(e)} \theta_A(\lambda(e'))(c)$. 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 pointwise minimum of the time increments of the fused transitions.

Example. For Figure 1, 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 $\langle \mathrm{aut}, \theta_X \rangle$, as computed by the above modification of Algorithm 1, is a timed interface summary of $\langle \mathcal{A}, \theta_A \rangle$.

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 Ponce de León et al. (2014). 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.

This work also heavily relies on previous theoretical works from the authors. We extend them by bringing a notion of logical time to Petri nets unfolding, with small additional computational cost.

REFERENCES

Athanasiou, K., Ponce de León, H., and Schwoon, S. (2015). Test case generation for concurrent systems using event structures. In *TAP*, 19–37.

Esparza, J. and Heljanko, K. (2008). *Unfoldings – A Partial-Order Approach to Model Checking.* Springer.

Esparza, J., Jezequel, L., and Schwoon, S. (2013). Computation of summaries using net unfoldings. In *FSTTCS*, 225–236.

Fidge, C.J. (1988). Timestamps in message-passing systems that preserve the partial ordering. *Australian Computer Science Communications*, 10(1), 56–66.

Jezequel, L., Madalinski, A., and Schwoon, S. (2018). Distributed computation of vector clocks in petri nets unfolding for test selection. hal-01735406.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *CACM*, 21(7), 558–565.

Madalinski, A. and Fabre, E. (2009). Modular construction of finite and complete prefixes of Petri net unfoldings. *Fundamenta Informaticae*, 95(1), 219–244.

Mattern, F. (1988). Virtual time and global states of distributed systems. In *International Workshop on Parallel and Distributed Algorithms*, 215–226.

Ponce de León, H., Haar, S., and Longuet, D. (2013). Unfolding-based test selection for concurrent conformance. In *ICTSS*, 98–113.

Ponce de León, H., Haar, S., and Longuet, D. (2014). Distributed testing of concurrent systems: Vector clocks to the rescue. In *ICTAC*, 369–387.

Tretmans, J. (1999). Testing concurrent systems: A formal approach. In *CONCUR*, 46–65.