



HAL
open science

Diagnosis of Asynchronous Discrete Event Systems: A Net Unfolding Approach.

Albert Benveniste, Eric Fabre, Stefan Haar, Claude Jard

► **To cite this version:**

Albert Benveniste, Eric Fabre, Stefan Haar, Claude Jard. Diagnosis of Asynchronous Discrete Event Systems: A Net Unfolding Approach.. IEEE Transactions on Automatic Control, 2003, 48 (5), pp.714-727. 10.1109/TAC.2003.811249 . inria-00638224

HAL Id: inria-00638224

<https://inria.hal.science/inria-00638224v1>

Submitted on 4 Nov 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Diagnosis of asynchronous discrete event systems, a net unfolding approach

Albert Benveniste, *Fellow, IEEE*, Eric Fabre, Stefan Haar, and Claude Jard

Abstract—In this paper we consider the diagnosis of asynchronous discrete event systems. We follow a so-called true concurrency approach, in which no global state and no global time is available. Instead, we use only local states in combination with a partial order model of time. Our basic mathematical tool is that of *net unfoldings* originating from the Petri net research area. This study was motivated by the problem of event correlation in telecommunications network management.

Keywords: *diagnosis, asynchronous diagnosis, discrete event systems, Petri nets, unfoldings, alarm correlation.*

I. INTRODUCTION

In this paper we study the diagnosis of truly asynchronous systems. Typical examples are networked systems, such as shown in Fig. 1. In this figure, the sensor system is distributed,

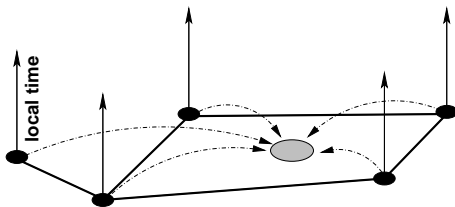


Fig. 1. Supervision of a networked system.

it involves several local sensors, attached to some nodes of the network (shown in black). Each sensor has only a partial view of the overall system. The different sensors possess their own local time, but they are not synchronized. Alarms are reported to the global supervisor (depicted in grey) asynchronously, and this supervisor performs diagnosis. This is the typical architecture in telecommunications network management systems today, our motivating application¹. Events may be correctly ordered by each individual sensor, but communicating alarm events via the network causes a loss of synchronization, and results in a *non-deterministic* and possibly *unbounded* interleaving at the supervisor. Hence, the right picture, for what the supervisor collects, is not a sequence of alarms, but rather a *partially ordered* set of alarms.

Fault diagnosis in discrete event systems has attracted a significant attention, see the work of Lafortune and co-authors [35][10] for an overview of the literature and introduction to the subject. Decentralized diagnosis is analyzed in [10], including both algorithms and their diagnosability properties; the solu-

tion is formulated in terms of a precomputed decentralized *diagnoser*, consisting of a set of communicating machines that have their states labeled by sets of faults and react to alarm observations and communications; the language oriented framework of Wonham and Ramadge (see [9]) is used, and the systems architecture is that of communicating automata, with a synchronous communication based on a global time, as revealed by the assumption “A6” in [10]. The work [10] has been extended by the same authors in [11] toward considering the effect of (bounded) communication delays in decentralized diagnosis. Difficulties resulting from communications in diagnosis are also investigated by Sengupta in [36]. Finally, the recent work of Tripakis [37] discusses issues of undecidability for a certain type of decentralized observability, this issue has again some relation with asynchrony. Baroni et al. [4] propose a different approach, more in the form of a simulation guided by the observed alarms, for a model of communicating automata. The solution proposed offers a first attempt to handle the problem of state explosion which results from the interleaving of events involving different components.

Diagnosis in the framework of Petri net models has also been investigated by some authors. Hadjicostis and Vergheze [22] consider faults in Petri nets in the form of losses or duplications of tokens; this is different from using Petri nets as an asynchronous machine model, for diagnosis. Valette and co-authors [34] use Petri nets to model the normal behavior of systems, and consider as faults the occurrence of events that do not match firing conditions properly. The work closest to ours is that of Giua and co-authors [23][24], it considers the estimation of the current marking from observations.

Event correlation in network management is the subject of a considerable literature, and a number of commercial products are available. We refer the reader to Gardner [21] for a survey. There are two main frameworks for most methods developed in this area. The first one relates to rule-based or case-based reasoning, an approach very different from the one we study here. The second one uses a causal model, in which the relation between faulty states and alarm events is modelled. The articles by Bouloutas et al. [7][8][27] belong to this family, as well as Rozé and Cordier [32] which relies on the diagnoser approach of [35]. The case of event correlation in network management also motivated the series of papers by Fabre et al. [6][2][3], on which the present paper relies.

As said before, our present approach was motivated by the problem of fault management in telecommunications networks, so it is worth discussing how this context motivated some of our choices. As seen from our bibliographical discussion, two classes of approaches were available, to handle the diagnosis of asynchronous systems.

This work was supported by the RNRT project MAGDA, funded by the Ministère de la Recherche; other partners of the project are France Telecom R&D, Alcatel, Ilog, and Paris-Nord University.

IRISA, Campus de Beaulieu, 35042 Rennes cedex, France. A.B., E.F., S.H. are with Inria, C.J. is with CNRS. Corresponding author Albert.Benveniste@irisa.fr

¹See [28] and the url <http://magda.elibel.tm.fr/> for a presentation of the MAGDA project on fault management in telecommunications networks.

A possible approach would consist in constructing a diagnoser in the form of a Petri net, having certain places labeled by faults, and transitions labeled by alarms. Received alarms trigger the net, and visiting a faulty place would indicate that some fault occurred in the original net for monitoring. Another approach would consist in estimating the current marking of the Petri net for monitoring, as in [23][24].

For our application, we needed to support distributed faults and event propagation and distributed sensor setups, from which wrong interleaving can result. Hence we feel it important, that robustness against a wrong interleaving should be addressed. However, the available approaches typically assume that alarms are received in sequence and that this sequence is an actual firing sequence of the net, an assumption not acceptable in our context.

Also, for our application in fault management in telecommunications networks (where faults are typically transient), providing explanations in the form of *scenarios*, not just snapshots, was essential. Finally, returning all scenarios compatible with the observations, was the requirement from operators in network management. They did not ask for a more elaborated information such as fault identification, or isolation.

In this paper, we propose an approach to handle unbounded asynchrony in discrete event systems diagnosis by using net *unfoldings*, originally proposed by M. Nielsen, G. Plotkin, and G. Winskel [30]. Unfoldings were used by Mc Millan [29] for model checking in verification. They were subsequently developed by Engelfriet [13], Rozenberg and Engelfriet [33], Esparza, and Römer [14][15][16]. Net unfoldings are branching structures suitable to represent the set of executions of a Petri net using an asynchronous semantics with local states and partially ordered time. In this structure, common prefixes of executions are shared, and executions differing only in the interleaving of their transition firings are represented only once. Our motivation, for using Petri nets and their unfoldings, is to have an elegant model of asynchronous finite state machines, therefore we restrict ourselves to safe Petri nets throughout this paper. Net unfoldings are not well-known in the control community, they have been however used for supervisory control in [25][26].

The paper is organized as follows. Section II is devoted to the problem setting. Subsection II-A collects the needed background material on Petri nets and their unfoldings. Subsection II-B introduces our first example. And our problem setting for asynchronous diagnosis is formalized in subsection II-C, which constitutes per se our first contribution.

In asynchronous diagnosis, some recorded alarm sequences differ only via the interleaving of concurrent alarms, hence it is desirable not to distinguish such alarm sequences. Similarly, it is desirable not to distinguish diagnoses which only differ in the interleaving of concurrent faults. *Diagnosis nets* are introduced to this end in section III, they express the solution of asynchronous diagnosis by using suitable unfoldings, and constitute the main contribution of this paper. Diagnosis nets return diagnosis as scenarios recording the whole history of failed/unfailed status and type of failure. Corresponding algorithms are given in section IV. These algorithms have the form of pattern matching rules and apply asynchronously.

II. ASYNCHRONOUS DIAGNOSIS: PROBLEM SETTING

In this section we first introduce the background we need on Petri nets and their unfoldings. Then we introduce informally asynchronous diagnosis on an example. And finally we formally define asynchronous diagnosis.

A. Background notions on Petri nets and their unfoldings

Basic references are [31][9][12]. Homomorphisms, conflict, concurrency, and unfoldings, are the essential concepts on which a true concurrency and fully asynchronous view of Petri nets is based. In order to introduce these notions, it will be convenient to consider general “nets” in the sequel.

Nets and homomorphisms. A net is a triple $\mathcal{P} = (P, T, \rightarrow)$, where P and T are disjoint sets of *places* and *transitions*, and $\rightarrow \subset (P \times T) \cup (T \times P)$ is the *flow relation*. The reflexive transitive closure of the flow relation \rightarrow is denoted by \preceq , and its irreflexive transitive closure is denoted by \prec . Places and transitions are called *nodes*, generically denoted by x . For $x \in P \cup T$, we denote by $\bullet x = \{y : y \rightarrow x\}$ the *preset* of node x , and by $x^\bullet = \{y : x \rightarrow y\}$ its *postset*. For $X \subset P \cup T$, we write $\bullet X = \bigcup_{x \in X} \bullet x$ and $X^\bullet = \bigcup_{x \in X} x^\bullet$. An *homomorphism* from a net \mathcal{P} to a net \mathcal{P}' is a map $\varphi : P \cup T \mapsto P' \cup T'$ such that: 1/ $\varphi(P) \subseteq P'$, $\varphi(T) \subseteq T'$, and 2/ for every node x of \mathcal{P} , the restriction of φ to $\bullet x$ is a bijection between $\bullet x$ and $\bullet \varphi(x)$, and the restriction of φ to x^\bullet is a bijection between x^\bullet and $\varphi(x)^\bullet$.

Occurrence nets. Two nodes x, x' of a net \mathcal{P} are *in conflict*, written $x \# x'$, if there exist distinct transitions $t, t' \in T$, such that $\bullet t \cap \bullet t' \neq \emptyset$ and $t \preceq x$, $t' \preceq x'$. A node x is in *self-conflict* if $x \# x$. An *occurrence net* is a net $O = (B, E, \rightarrow)$, satisfying the following additional properties:

$\forall x \in B \cup E : \neg[x \# x]$	no node is in self-conflict
$\forall x \in B \cup E : \neg[x \prec x]$	\preceq is a partial order
$\forall x \in B \cup E : \{y : y \prec x\} < \infty$	\preceq is well founded
$\forall b \in B : \bullet b \leq 1$	each place has at most one input transition

We will assume that the set of minimal nodes of O is contained in B , and we denote by $\min(B)$ or $\min(O)$ this minimal set. Specific terms are used to distinguish occurrence nets from general nets. B is the set of *conditions*, E is the set of *events*, \prec is the *causality* relation.

Nodes x and x' are *concurrent*, written $x \perp x'$, if neither $x \preceq x'$, nor $x' \preceq x$, nor $x \# x'$ hold. A *co-set* is a set X of concurrent conditions. A maximal (for set inclusion) co-set is called a *cut*. A *configuration* κ is a sub-net of O , which is *conflict-free* (no two nodes are in conflict), and *causally closed* (if $x' \preceq x$ and $x \in \kappa$, then $x' \in \kappa$).

Occurrence nets are useful to represent executions of Petri nets. They are a subclass of nets, in which essential properties are visible via the topological structure of the bipartite graph.

Petri nets. For \mathcal{P} a net, a *marking* of \mathcal{P} is a multiset M of places, i.e., a map $M : P \mapsto \{0, 1, 2, \dots\}$. A *Petri net* is a pair $\mathcal{P} = (\mathcal{P}, M_0)$, where \mathcal{P} is a net having finite sets of places and transitions, and M_0 is an *initial* marking. A transition $t \in T$ is *enabled* at marking M if $M(p) > 0$ for every $p \in \bullet t$. Such a

transition can *fire*, leading to a new marking $M' = M - \bullet t + t \bullet$, we denote this by $M[t]M'$. The set of *reachable* markings of \mathcal{P} is the smallest (w.r.t. set inclusion) set $M_0[\cdot]$ containing M_0 and such that $M \in M_0[\cdot]$ and $M[t]M'$ together imply $M' \in M_0[\cdot]$. Petri net \mathcal{P} is *safe* if $M(P) \subseteq \{0, 1\}$ for every reachable marking M . Throughout this paper, we consider only safe Petri nets, hence marking M can be regarded as a subset of places. A finite occurrence net \mathcal{B} can be regarded as a Petri net, where the initial marking is $M_0 = \min(\mathcal{B})$.

Branching processes and unfoldings. A *branching process* of Petri net \mathcal{P} is a pair $\mathcal{B} = (O, \varphi)$, where O is an occurrence net, and φ is an homomorphism from O to \mathcal{P} regarded as nets, such that: 1/ the restriction of φ to $\min(O)$ is a bijection between $\min(O)$ and M_0 (the set of initially marked places), and 2/ for all $e, e' \in E$, $\bullet e = \bullet e'$ and $\varphi(e) = \varphi(e')$ together imply $e = e'$. By abuse of notation, we shall sometimes write $\min(\mathcal{B})$ instead of $\min(O)$.

The set of all branching processes of Petri net \mathcal{P} is uniquely defined, up to an isomorphism (i.e., a renaming of the conditions and events), and we shall not distinguish isomorphic branching processes. For $\mathcal{B}, \mathcal{B}'$ two branching processes, \mathcal{B}' is a *prefix* of \mathcal{B} , written $\mathcal{B}' \sqsubseteq \mathcal{B}$, if there exists an injective homomorphism ψ from \mathcal{B}' into \mathcal{B} , such that $\psi(\min(\mathcal{B}')) = \min(\mathcal{B})$, and the composition $\varphi \circ \psi$ coincides with φ' , where \circ denotes the composition of maps. By theorem 23 of [13], there exists (up to an isomorphism) a unique maximum branching process according to \sqsubseteq , we call it the *unfolding* of \mathcal{P} , and denote it by $\mathcal{U}_{\mathcal{P}}$. The unfolding of \mathcal{P} possesses the following universal property: for every occurrence net O , and every homomorphism $\phi : O \mapsto \mathcal{P}$, there exists an *injective* homomorphism $\iota : O \mapsto \mathcal{U}_{\mathcal{P}}$, such that:

$$\phi = \varphi \circ \iota, \quad (1)$$

where φ denotes the homomorphism associated to $\mathcal{U}_{\mathcal{P}}$. Decomposition (1) expresses that $\mathcal{U}_{\mathcal{P}}$ “maximally unfolds” \mathcal{P} . If \mathcal{P} is itself an occurrence net and $M_0 = \min(\mathcal{P})$ holds, then $\mathcal{U}_{\mathcal{P}}$ identifies with \mathcal{P} .

Configurations of the unfolding $\mathcal{U}_{\mathcal{P}}$ are adequate representations of the firing sequences of \mathcal{P} . Let M_0, M_1, M_2, \dots be a maximal firing sequence of \mathcal{P} , and let $M_{k-1}[t_k]M_k$ be the associated sequence of fired transitions. Then there exists a unique maximal (for set inclusion) configuration κ of \mathcal{P} having the following properties: κ is the union of a sequence e_1, e_2, \dots of events and a sequence c_0, c_1, c_2, \dots of cuts, such that, for each $k > 0$, $\varphi(c_k) = M_k$, $\varphi(e_k) = t_k$, and $c_{k-1} \supseteq \bullet e_k, e_k \bullet \subseteq c_k$. Conversely, each maximal configuration of \mathcal{P} defines a maximal firing sequence, which is unique up to the interleaving of structurally concurrent transitions—transitions t and t' are structurally concurrent iff $\bullet t' \cap (\bullet t \cup t \bullet) = \emptyset$ and $t \bullet \cap (t' \bullet \cup t \bullet) = \emptyset$.

Example 1. Fig. 2 shows the example we will use throughout this paper. A Petri net \mathcal{P} is shown on the top left. Its places are 1, 2, 3, 4, 5, 6, 7, and its transitions are i, ii, iii, iv, v, vi . Places constituting the initial marking are encircled in thick.

A branching process $\mathcal{B} = (O, \varphi)$ of \mathcal{P} is shown on the bottom. Its conditions are depicted by circles, and its events are figured by boxes. Each condition b of \mathcal{B} is labeled by $\varphi(b)$, a place of \mathcal{P} . Each event e of \mathcal{B} is labeled by $\varphi(e)$, a transition of \mathcal{P} . A configuration of Petri net \mathcal{P} is shown in grey. Note that the

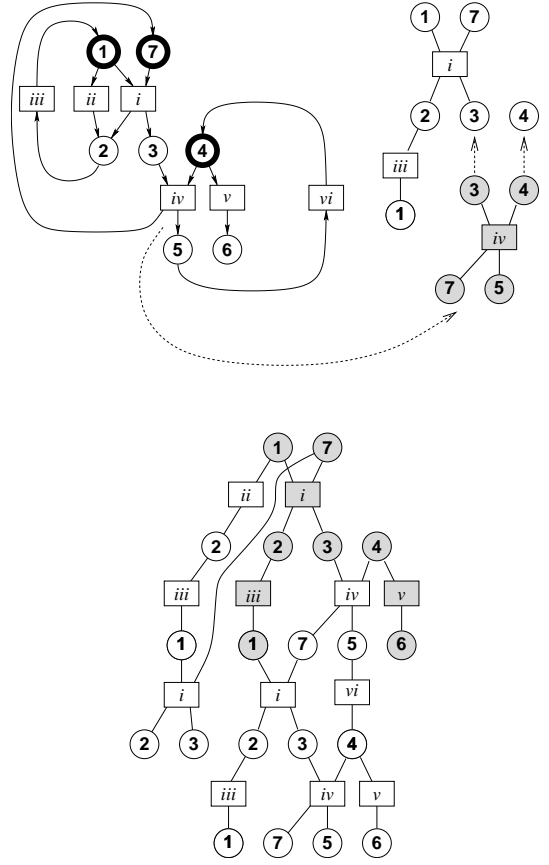


Fig. 2. Example 1 (top left), a configuration (top right), and a branching process (bottom). For this and subsequent examples, we take the following convention for drawing Petri nets and occurrence nets. In Petri nets, the flow relation is depicted using *directed arrows*. In occurrence nets, since no cycle occurs, the flow relation *progresses downwards*, and therefore there is no need to figure them via directed arrows, standard solid lines are used instead.

minimal condition labeled by 7 is branching in \mathcal{B} , although it is not branching in \mathcal{P} itself. The reason is that, in \mathcal{P} , the token can freely move along the circuit $1 \rightarrow ii \rightarrow 2 \rightarrow iii \rightarrow 1$, and resynchronize afterwards with the token sitting in 7.

The mechanism for constructing the unfolding of Petri net \mathcal{P} is illustrated in the top right, it is informally explained as follows. Put the three conditions labeled by the initial marking of \mathcal{P} , this is the minimal branching process of \mathcal{P} . Then, for each constructed branching process \mathcal{B} , select a co-set X of \mathcal{B} , which is labeled by the preset $\bullet t$ of some transition t of \mathcal{P} , and has no event labeled by t in its postset within \mathcal{B} . Append to X a net isomorphic to $\bullet t \rightarrow t \rightarrow t \bullet$ (recall that $\varphi(\bullet t) = X$), and label its additional nodes by t and $t \bullet$, respectively. Performing this recursively yields all possible finite branching processes of \mathcal{P} . Their union is the unfolding $\mathcal{U}_{\mathcal{P}}$.

Labeled nets and their products. For $\mathcal{P} = (P, T, \rightarrow)$ a net, a *labeling* is a map $\lambda : T \mapsto A$, where A is some finite alphabet. A net $\mathcal{P} = (P, T, \rightarrow, \lambda)$ equipped with a labeling λ is called a *labeled net*. For $\mathcal{P}_i = \{P_i, T_i, \rightarrow_i, \lambda_i\}$, $i = 1, 2$, two labeled nets, their product $\mathcal{P}_1 \times \mathcal{P}_2$ is the labeled net defined as follows:

$$\mathcal{P}_1 \times \mathcal{P}_2 = (P, T, \rightarrow, \lambda). \quad (2)$$

In (2), $P = P_1 \uplus P_2$, where \uplus denotes the disjoint union, and:

$$T = \begin{aligned} & \cup \{t_1 \in T_1 : \lambda_1(t_1) \in A_1 \setminus A_2\} & (i) \\ & \cup \{(t_1, t_2) \in T_1 \times T_2 : \lambda_1(t_1) = \lambda_2(t_2)\} & (ii) \\ & \cup \{t_2 \in T_2 : \lambda_2(t_2) \in A_2 \setminus A_1\}, & (iii) \end{aligned}$$

$$p \rightarrow t \quad \text{iff} \quad \begin{aligned} & p \in P_1 \text{ and } p \rightarrow_1 t_1 & \text{for case (ii) or (i)} \\ & p \in P_2 \text{ and } p \rightarrow_2 t_2 & \text{for case (ii) or (iii)} \end{aligned}$$

$$t \rightarrow p \quad \text{iff} \quad \begin{aligned} & p \in P_1 \text{ and } t_1 \rightarrow_1 p & \text{for case (ii) or (i)} \\ & p \in P_2 \text{ and } t_2 \rightarrow_2 p & \text{for case (ii) or (iii)} \end{aligned}$$

In cases (i,iii) only one net fires a transition and this transition has a private label, while the two nets synchronize on transitions with identical labels in case (ii). Petri nets and occurrence nets inherit the above notions of labeling and product.

B. Discussing asynchronous diagnosis on example 1

A labeled Petri net model. Our first example of Fig. 2 is redrawn slightly differently in Fig. 3, in the form of a labeled Petri net.

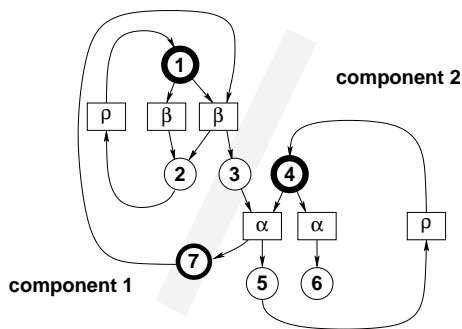


Fig. 3. Example 1, two interacting components modelled as a labeled Petri net.

The example is now interpreted as two interacting components, numbered 1 and 2. Component 2 uses the services of component 1 for its functioning, and therefore it may fail to deliver its service when component 1 is faulty.

Component 1 has two states: nominal, figured by place 1, and faulty, figured by place 2. When getting in faulty state, the component 1 emits an alarm β , which is associated to transition (i) and (ii) (cf. Fig. 2) as a label. The fault of component 1 is temporary, and therefore self-repair can occur, this is figured by the label ρ associated to transition (iii) (cf. Fig. 2).

Component 2 has three states, figured by places 4, 5, 6. State 4 is nominal, state 6 indicates that component 2 is faulty, and state 5 indicates that component 2 fails to deliver its service, due to the failure of component 1. Fault 6 is permanent and cannot be repaired.

The fact that component 2 may fail to deliver its service due to a fault of component 1, is modelled by the shared place 3. The monitoring system of component 2 only detects that this component fails to deliver its service, it does not distinguish between the different reasons for this. Hence the *same* alarm α is attached to the two transitions (iv,v) as a label (cf. Fig. 2). Since fault 2 of component 1 is temporary, self-repair can also occur for component 2, when in faulty state 5. This self-repair is not synchronized with that of component 1, but is still assumed

to be manifested by the same label ρ . Finally, place 7 guarantees that fault propagation, from component 1 to component 2, occurs only when the latter is in nominal state.

The grey area indicates where interaction occurs between the two components. The initial marking consists of the three (nominal) states 1, 4, 7. Labels (alarms α, β or self-repair ρ) attached to the different transitions or events, are generically referred to as *alarms* in the sequel.

The different setups considered, for diagnosis. Three different setups can be considered:

Setup S₁: The successive alarms are recorded *in sequence* by a single supervisor, in charge of fault monitoring. The sensor and communication infrastructure guarantees that *causality is respected*: for any two alarms that are causally related (α causes α'), then α is recorded before α' .

Setup S₂: Each sensor records its local alarms in sequence, by respecting causality. The different sensors perform independently and asynchronously, and there is a single supervisor which collects the records from the different sensors. *No assumption is made on the communication infrastructure.* Thus any interleaving of the records from different sensors can occur, and possible causalities relating alarms collected at different sensors are lost.

Setup S₃: The fault monitoring is performed in a *distributed* way, by different supervisors cooperating asynchronously. Each supervisor is attached to a component, it records its local alarms in sequence, and can exchange supervision messages with the other supervisor, asynchronously. No assumption is made on the communication infrastructure.

In this paper we consider S₁ and S₂ (and generalizations of them), but not distributed diagnosis S₃ (for distributed diagnosis, the reader is referred to [17][18]). Note that Internet cannot be used as a communication infrastructure for setup S₁, but it can be used for setup S₂.

The different setups are illustrated in Fig. 4, which is a combination of Fig. 2 and Fig. 3. The labeled Petri net of Fig. 3 is redrawn, on the top, with the topology used in Fig. 2. In the bottom left, we redraw the configuration shown in grey in Fig. 2-right, call it κ , and we relabel its events by their associated alarms. Configuration κ expresses that component 1 went into its faulty state 2, and then was repaired; concurrently, component 2 moved to its faulty state 6, where self-repair cannot occur. Note that the transmission of the fault of component 1 to component 2, via place 3, is preempted, due to the fatal failure of component 2.

How alarms are recorded is modelled by the two occurrence nets shown in the third and fourth diagrams, we call them *alarm patterns*. In the third diagram, we assume the first setup, in which a single sensor is available to collect the alarms. Hence configuration κ produces the alarms β, α, ρ , recorded in sequence. This record is modelled by the linear alarm pattern shown in the third diagram. This alarm pattern has its events labeled by alarms, but its conditions are “blind”, i.e., they have no label. This manifests the fact that the different places of the Petri net, which are traversed while producing the alarms β, α , or ρ , are not observed.

Now, in the last diagram, we show the case of the second setup, in which β, ρ are collected by the first sensor, and α is col-

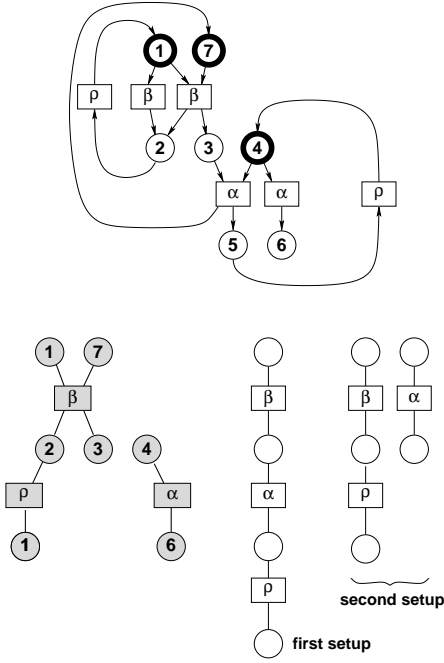


Fig. 4. Example 1, a scenario involving a single sensor, and two independent sensors.

lected by the second one, independently. The result is an alarm pattern composed of two concurrent parts, corresponding to the records collected by each sensor. When collected by the supervisor, these concurrent parts can interleave arbitrarily—this manifests asynchrony.

Asynchronous diagnosis. Alarm patterns are generically denoted by the symbol \mathcal{A} . Note that each sensor delivers, as an alarm pattern, some linear extension of the partial order of events it sees. But the causality relations involving pairs of events seen by different sensors, are lost. In general, *observations may add some causalities, may lose other ones, but they never reverse any of them.* Therefore, the only valid invariant between alarm pattern \mathcal{A} and the configuration κ that produced it, is that \mathcal{A} and κ possess a common linear extension. With this definition, we encompass setups \mathbf{S}_1 and \mathbf{S}_2 in a common framework. From the above discussion, we must accept as plausible explanations of an alarm pattern \mathcal{A} any configuration κ such that \mathcal{A} and κ possess a common linear extension. Such κ are said to *explain* \mathcal{A} . We are now ready to formalize our problem setting.

C. Asynchronous diagnosis: formal problem setting

Now, we formalize what an alarm pattern \mathcal{A} is, and what it means, for \mathcal{A} , to be associated with some configuration κ . We are given the following objects, where the different notions have been introduced in subsection II-A:

- A labeled Petri net $\mathcal{P} = (P, T, \rightarrow, M_0, \lambda)$, where the range of the labeling map λ is the alphabet of possible *alarms*, denoted by A , and
- its unfolding $\mathcal{U}_{\mathcal{P}} = (B, E, \rightarrow, \varphi)$.

Note the following chain of labeling maps:

$$\underbrace{E}_{\text{events}} \xrightarrow{\varphi} \underbrace{T}_{\text{transitions}} \xrightarrow{\lambda} \underbrace{A}_{\text{alarms}}$$

$$e \mapsto \varphi(e) \mapsto \lambda(\varphi(e)) \stackrel{\Delta}{=} \Lambda(e), \quad (3)$$

which defines the *alarm label* of event e , we denote it by $\Lambda(e)$ —we call it also “alarm”, for short, when no confusion can occur.

An *extension* of a net $\mathcal{P} = (P, T, \rightarrow)$ is any net obtained by adding places and flow relations but not transitions. Occurrence nets inherit this notion. An occurrence net induces a labeled partial order on the set of its events, extending this occurrence net induces an extension of this labeled partial order².

Two labeled occurrence nets $O = (B, E, \rightarrow, \Lambda)$ and $O' = (B', E', \rightarrow', \Lambda')$ are called *alarm-isomorphic* if there exists an isomorphism ψ , from (B, E, \rightarrow) onto (B', E', \rightarrow') , seen as directed graphs, which preserves the alarm labels, i.e., such that $\forall e \in E : \Lambda'(\psi(e)) = \Lambda(e)$. Thus, two alarm-isomorphic occurrence nets can be regarded as identical if we are interested only in causalities and alarm labels.

Definition 1 (alarm pattern) Consider \mathcal{P} , $\mathcal{U}_{\mathcal{P}}$, and Λ , as in (3). A labeled occurrence net $\mathcal{A} = (B_{\mathcal{A}}, E_{\mathcal{A}}, \rightarrow_{\mathcal{A}}, \lambda_{\mathcal{A}})$ is an alarm pattern of \mathcal{P} iff:

1. Its labeling map $\lambda_{\mathcal{A}}$ takes its value in the alphabet A of alarms,
2. \mathcal{A} is itself a configuration (it is conflict free), its set of conditions $B_{\mathcal{A}}$ is disjoint from that of $\mathcal{U}_{\mathcal{P}}$, and
3. There exists a configuration κ of $\mathcal{U}_{\mathcal{P}}$, such that \mathcal{A} and κ possess extensions that are alarm-isomorphic.

Assuming, for \mathcal{A} , a set of places disjoint from that of $\mathcal{U}_{\mathcal{P}}$, aims at reflecting that alarm patterns vehicle no information regarding hidden states of the original net. This justifies condition 2. Concerning condition 3 the allowed discrepancy between κ and \mathcal{A} formalizes the possible loss of some causalities (e.g., due to independent and non synchronized sensors), and the possible adding of other ones (e.g., when sensors record their alarms in sequence). The key fact is that the information about the concurrency of events produced by the system cannot be observed by the supervisor. For instance, if the supervisor receives two alarm events α, β that are not causally related, then the net \mathcal{P} may have produced $\alpha \perp \beta$, or $\alpha \preceq \beta$, or $\alpha \succeq \beta$.

To refer to our context of diagnosis, we say that the configuration κ *can explain* \mathcal{A} . For \mathcal{A} a given alarm pattern of \mathcal{P} , we denote by

$$\mathbf{diag}(\mathcal{A}) \quad (4)$$

the set of configurations κ of $\mathcal{U}_{\mathcal{P}}$ satisfying conditions 1,2,3 of definition 1. Due to asynchrony, ambiguity frequently occurs so that the set $\mathbf{diag}(\mathcal{A})$ is far from being a singleton. Therefore the issue of how to compute and represent this solution set efficiently is of great importance when large scale applications are considered. In the next subsection, we propose an adequate data structure to represent and manipulate the set $\mathbf{diag}(\mathcal{A})$ efficiently, we call it a *diagnosis net*.

III. DIAGNOSIS NETS: EXPRESSING ASYNCHRONOUS DIAGNOSIS BY MEANS OF UNFOLDINGS

In this section, we provide explicit formulas for the solution of asynchronous diagnosis, in the form of suitable unfoldings.

² Recall that the labeled partial order (X, \preceq) is an *extension* of labeled partial order (X', \preceq') if labeled sets X and X' are isomorphic, and $\preceq \supseteq \preceq'$ holds. When (X, \preceq) is a total order, we call it a *linear extension* of (X', \preceq') .

A first natural idea is to represent $\mathbf{diag}(\mathcal{A})$ by the minimal subnet of unfolding $\mathcal{U}_{\mathcal{P}}$ which contains all configurations belonging to $\mathbf{diag}(\mathcal{A})$, we denote it by $\mathcal{U}_{\mathcal{P}}(\mathcal{A})$. Subnet $\mathcal{U}_{\mathcal{P}}(\mathcal{A})$ inherits canonically by restriction, the causality, conflict, and concurrency relations defined on $\mathcal{U}_{\mathcal{P}}$. Net $\mathcal{U}_{\mathcal{P}}(\mathcal{A})$ contains all configurations belonging to $\mathbf{diag}(\mathcal{A})$, but unfortunately it also contains undesirable maximal configurations *not* belonging to $\mathbf{diag}(\mathcal{A})$, as Fig. 5 reveals.

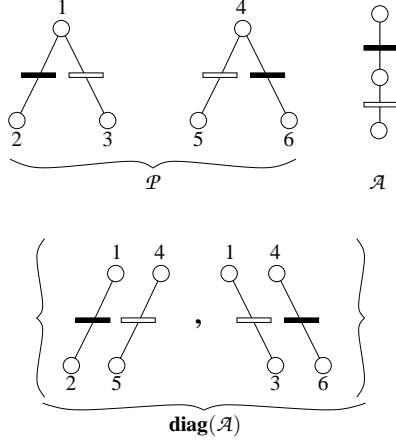


Fig. 5. Example 2. Showing \mathcal{P} , \mathcal{A} , and $\mathbf{diag}(\mathcal{A})$. Note that $\mathcal{U}_{\mathcal{P}}(\mathcal{A}) = \mathcal{P}$.

In this figure, we show, on the top left, a Petri net \mathcal{P} having the set of places $\{1,4\}$ as initial marking, note that \mathcal{P} is an occurrence net. In the top right, we show a possible associated alarm pattern \mathcal{A} . Alarm labels are figured by colors (black and white). The set $\mathbf{diag}(\mathcal{A})$ is shown on the bottom, it comprises two configurations. Unfortunately the minimal subnet $\mathcal{U}_{\mathcal{P}}(\mathcal{A})$ of the original unfolding \mathcal{P} which contains $\mathbf{diag}(\mathcal{A})$, is indeed identical to \mathcal{P} ! Undesirable configurations are $\{(1, t_{12}, 2), (4, t_{46}, 6)\}$ and $\{(1, t_{13}, 3), (4, t_{45}, 5)\}$ (in these statements, t_{12} denotes the transition separating states 1 and 2). But configuration $\{(1, t_{12}, 2), (4, t_{46}, 6)\}$ is such that its two transitions t_{12}, t_{46} explain the *same* alarm event in \mathcal{A} , and therefore this configuration cannot explain \mathcal{A} . And the same holds for the other undesirable configuration.

Fig. 6 suggests an alternative solution, using the product $\mathcal{P} \times \mathcal{A}$ of \mathcal{P} and \mathcal{A} , seen as labeled nets with respective labels λ and $\lambda_{\mathcal{A}}$ (see subsection II-C for these notations). The unfolding $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ is shown. The projection, on the set of nodes labelled by nodes from \mathcal{P} , is depicted using larger arrows. The reader can verify that the corresponding set of maximal configurations coincides with $\mathbf{diag}(\mathcal{A})$. This suggests that $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ is an appropriate representation of $\mathbf{diag}(\mathcal{A})$. We formalize this in the theorem to follow. We use the notations from subsections II-A and II-C, and we need a few more notations.

For $\mathcal{P} = (P, T, \rightarrow)$ a net and X a subset of its nodes, $\mathcal{P}|_X$ denotes the *restriction* of \mathcal{P} to X , defined as

$$\mathcal{P}|_X \triangleq (P \cap X, T \cap X, \rightarrow|_X),$$

where the flow relation $\rightarrow|_X$ is defined as the restriction, to $X \times X$, of the flow relation $\rightarrow \subseteq (P \times T) \cup (T \times P)$ given on \mathcal{P} . Be careful that we restrict the flow relation, not its transitive closure.

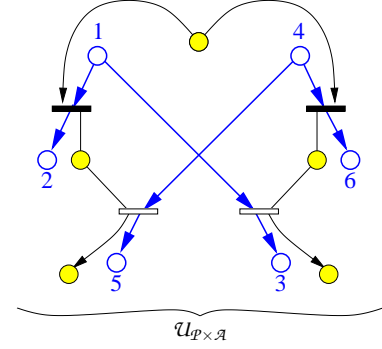
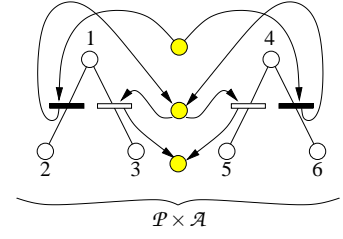


Fig. 6. Example 2. Representing $\mathbf{diag}(\mathcal{A})$ by $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$.

Let $\mathcal{P} = (P, T, \rightarrow, M_0, \lambda)$ and $\mathcal{Q} = (Q, S, \rightarrow, N_0, \mu)$ be two labeled Petri nets, and $\mathcal{U} = (B, E, \rightarrow, \varphi)$ a sub-net of the unfolding $\mathcal{U}_{\mathcal{P} \times \mathcal{Q}}$. Define the labeled occurrence net $\mathbf{proj}_{\mathcal{P}}(\mathcal{U})$, the *projection* of \mathcal{U} on \mathcal{P} , as follows: 1/ restrict \mathcal{U} to its subset of nodes labelled by nodes from \mathcal{P} , and 2/ project, onto T , the labels consisting of synchronized pairs of transitions belonging to $T \times S$. Let us formalize this construction. The set E of events of \mathcal{U} decomposes as $E = E_{\mathcal{P}} \cup E_{\mathcal{P}, \mathcal{Q}} \cup E_{\mathcal{Q}}$, where $E_{\mathcal{P}}$ is the set of events labeled by transitions $t \in T$, $E_{\mathcal{Q}}$ is the set of events labeled by transitions $s \in S$, and $E_{\mathcal{P}, \mathcal{Q}}$ is the set of events labeled by pairs of synchronized transitions $(t, s) \in T \times S$. Then we define:

$$\mathbf{proj}_{\mathcal{P}}(\mathcal{U}) \triangleq (\mathcal{U}|_{\varphi^{-1}(\mathcal{P}) \cup E_{\mathcal{P}, \mathcal{Q}}}, \phi) \quad (5)$$

where the labeling map ϕ is defined as follows: if $b \in B$, then $\phi(b) = \varphi(b)$; if $e \in E_{\mathcal{P}}$, then $\phi(e) = \varphi(e)$; if $e \in E_{\mathcal{P}, \mathcal{Q}}$ is such that $\varphi(e) = (t, s)$, then $\phi(e) = t$. Hence $\mathbf{proj}_{\mathcal{P}}(\mathcal{U})$ has $P \cup T$, the set of nodes of \mathcal{P} , as its label set.

Finally, for \mathcal{O} an occurrence net, we denote by $\mathbf{config}(\mathcal{O})$ the set of all its configurations.

Theorem 1: Let $\mathcal{U}_{\mathcal{P}}$ be the unfolding of some Petri net \mathcal{P} , \mathcal{A} an associated alarm pattern, and let $\mathbf{diag}(\mathcal{A})$ be defined as in (4). Consider the unfolding $\mathcal{U}_{\mathcal{P} \times \mathcal{A}} \triangleq (\bar{B}, \bar{E}, \rightarrow, \bar{\varphi})$, and its associated projections $\mathbf{proj}_{\mathcal{P}}(\cdot)$ and $\mathbf{proj}_{\mathcal{A}}(\cdot)$. Then, $\kappa \in \mathbf{diag}(\mathcal{A})$ iff there exists $\bar{\kappa} \in \mathbf{config}(\mathcal{U}_{\mathcal{P} \times \mathcal{A}})$, such that:

$$\mathbf{proj}_{\mathcal{P}}(\bar{\kappa}) = \kappa \quad \text{and} \quad \mathbf{proj}_{\mathcal{A}}(\bar{\kappa}) = \mathcal{A}. \quad (6)$$

Note that the product $\mathcal{P} \times \mathcal{A}$ involves only synchronized transitions. Note also that every $\bar{\kappa}$ satisfying (6) must be a maximal configuration of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$. Theorem 1 expresses that $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ is an adequate representation of $\mathbf{diag}(\mathcal{A})$, we call it a *diagnosis net*.

Proof: We first prove the *if* part. Let $\bar{\kappa}$ be a configuration of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ such that $\mathbf{proj}_{\mathcal{A}}(\bar{\kappa}) = \mathcal{A}$, and define $\kappa = \mathbf{proj}_{\mathcal{P}}(\bar{\kappa})$. By definition of net extensions (cf. definition 1 and above), $\bar{\kappa}$ is an

extension of both κ and \mathcal{A} . Hence, by definition 1, $\kappa \in \mathbf{diag}(\mathcal{A})$. This was the easy part.

Now we prove the *only if* part. Select an arbitrary $\kappa \in \mathbf{diag}(\mathcal{A})$. We need to show the existence of a $\bar{\kappa}$ satisfying (6). Since $\kappa \in \mathbf{diag}(\mathcal{A})$, then κ and \mathcal{A} possess two respective extensions, κ^e and \mathcal{A}^e , that are alarm isomorphic, let ψ be the corresponding isomorphism, from $E_{\mathcal{A}}$ (the set of events of \mathcal{A}), onto the set of events of κ . Note that κ^e possesses additional dummy conditions that are not labeled by places from \mathcal{P} , and \mathcal{A}^e possesses conditions that do not belong to \mathcal{A} .

Consider the following configuration $\bar{\kappa}^e$, obtained as follows. Its set of events is the set of pairs $(\psi(e), e)$, where e ranges over $E_{\mathcal{A}}$. Then its set of conditions as well as its flow relation is defined by:

$$\text{flow relation of } \bar{\kappa}^e : \begin{cases} \bullet(\psi(e), e) &= \bullet\psi(e) \cup \bullet e \\ (\psi(e), e)^\bullet &= \psi(e)^\bullet \cup e^\bullet \end{cases} \quad (7)$$

where the pre- and postset operations occurring on the right hand sides of the two equalities are taken from the *extensions* κ^e and \mathcal{A}^e . Informally speaking, $\bar{\kappa}^e$ is obtained by glueing together κ^e and \mathcal{A}^e at their events associated via ψ . Note that $\bar{\kappa}^e$ is circuit free.

Now, erase, in $\bar{\kappa}^e$, the conditions that are neither labeled by places from \mathcal{P} , nor belong to \mathcal{A} (such places originate from having extended κ and \mathcal{A} into κ^e and \mathcal{A}^e , respectively). Call $\bar{\kappa}$ the so obtained configuration. By construction:

$$\mathbf{proj}_{\mathcal{P}}(\bar{\kappa}) = \kappa \quad \text{and} \quad \mathbf{proj}_{\mathcal{A}}(\bar{\kappa}) = \mathcal{A}.$$

Thus it remains to show that $\bar{\kappa} \in \mathbf{config}(\mathcal{U}_{\mathcal{P} \times \mathcal{A}})$. On the one hand, $\bar{\kappa}^e$ was circuit/conflict free and causally closed, then so is $\bar{\kappa}$, thus $\bar{\kappa}$ is a configuration. On the other hand, the flow relation and nodes of $\bar{\kappa}$ are also defined by formula (7), provided that the pre- and postset operations occurring on the right hand sides of the two equalities are taken from the *original* configurations κ and \mathcal{A} . By keeping in mind (7), we define the following labeling map $\bar{\phi}$ on $\bar{\kappa}$. For each event e :

$$\bar{\phi}(\psi(e), e) = (\phi(\psi(e)), e),$$

and, for each $\bar{b} \in \bullet(\psi(e), e) \cup (\psi(e), e)^\bullet$:

$$\begin{cases} \bar{b} \in \bullet e \cup e^\bullet & \Rightarrow \bar{\phi}(\bar{b}) = \bar{b} \\ \bar{b} \in \bullet\psi(e) \cup \psi(e)^\bullet & \Rightarrow \bar{\phi}(\bar{b}) = \phi(\bar{b}) \end{cases}$$

Hence, $\bar{\phi}$ is an homomorphism from $\bar{\kappa}$ into $\mathcal{P} \times \mathcal{A}$. By using the universal property (1), there exists an injective homomorphism from $\bar{\kappa}$ into $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$. This, and the fact that $\bar{\kappa}$ was already proved to be a configuration, shows that $\bar{\kappa} \in \mathbf{config}(\mathcal{U}_{\mathcal{P} \times \mathcal{A}})$, this finishes the proof of the theorem. \diamond

Remark. Theorem 1 assumes the knowledge of the initial marking M_0 for Petri net \mathcal{P} . When only a set \mathcal{M}_0 of possible initial markings is known instead, simply augment $(\mathcal{P}, T, \rightarrow)$ as follows. Add some additional place p_0 not belonging to \mathcal{P} , for each possible initial marking $M_0 \in \mathcal{M}_0$ add one transition t_{M_0} to T with label α_0 not belonging to \mathcal{A} , and add the branches $p_0 \rightarrow t_{M_0} \rightarrow M_0$ to the flow relation. To account for this additional places and transitions, add to \mathcal{A} a dummy prefix of the form $b_0 \rightarrow e_0 \rightarrow \min(\mathcal{A})$, where event e_0 has label α_0 . Then, apply theorem 1 to the so augmented Petri net.

Example 1, illustration of diagnosis nets, and comparison with the use of the marking graph. Fig. 7 shows an illustration of the-

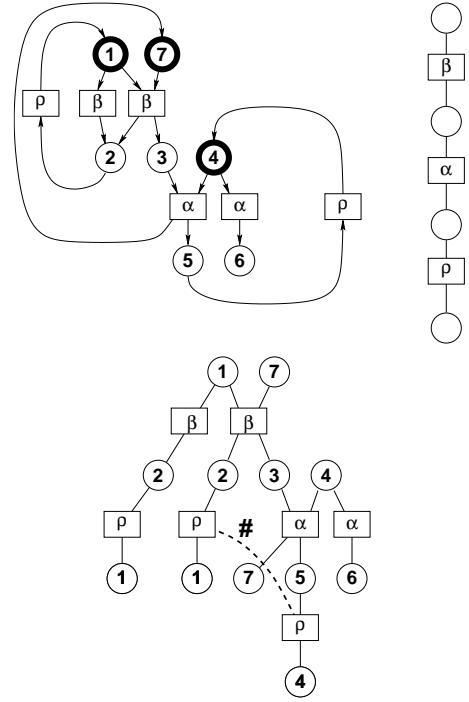


Fig. 7. Example 1, diagnosis net, an illustration of theorem 1.

orem 1. In this figure we show the Petri net \mathcal{P} of Fig. 3 (top left), an associated alarm pattern \mathcal{A} (top right), and the net $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, restricted to its nodes labeled by nodes from \mathcal{P} (bottom). We show in dashed-thick the additional conflict relation between the two otherwise concurrent events labeled by the same alarm ρ . This conflict is inherited from the sharing of a common condition, not shown, belonging to \mathcal{A} . It is easily checked that $\mathbf{diag}(\mathcal{A})$ is adequately represented by this diagram³.

Four alternative explanations are delivered by this diagnosis, this reflects the ambiguity resulting from asynchrony in this example. Explanation 1: component 1 gets in its faulty state without causing damage to component 2, and then gets self-repaired; independently, component 2 gets into its fatal faulty state 6; thus, for this scenario, $(\beta \prec \rho) \perp \alpha$ holds. Explanation 2: component 1 gets in its faulty state while causing damage to component 2, and then gets self-repaired; independently, component 2 gets into its fatal faulty state 6; again, for this scenario, $(\beta \prec \rho) \perp \alpha$ holds. Explanation 3: component 1 gets in its faulty state while causing damage to component 2; consequently, component 2 fails to deliver its service and gets into its state 5, where it subsequently gets self-repaired; for this scenario, we have $\beta \prec \rho \prec \alpha$. Explanation 4: component 1 gets in its faulty state while causing damage to component 2; consequently, component 2 fails to deliver its service; independently component 1 gets self-repaired; thus $\beta \prec (\rho \perp \alpha)$ holds for this scenario.

Fig. 8 compares diagnosis nets with the use of marking graphs. The reader is referred again to our running example 1 (shown in Fig. 7), call it \mathcal{P} . In the first diagram we show the

³The restriction, to its events, of this data structure, is an *event structure* according to Winskel's definition [30][38].

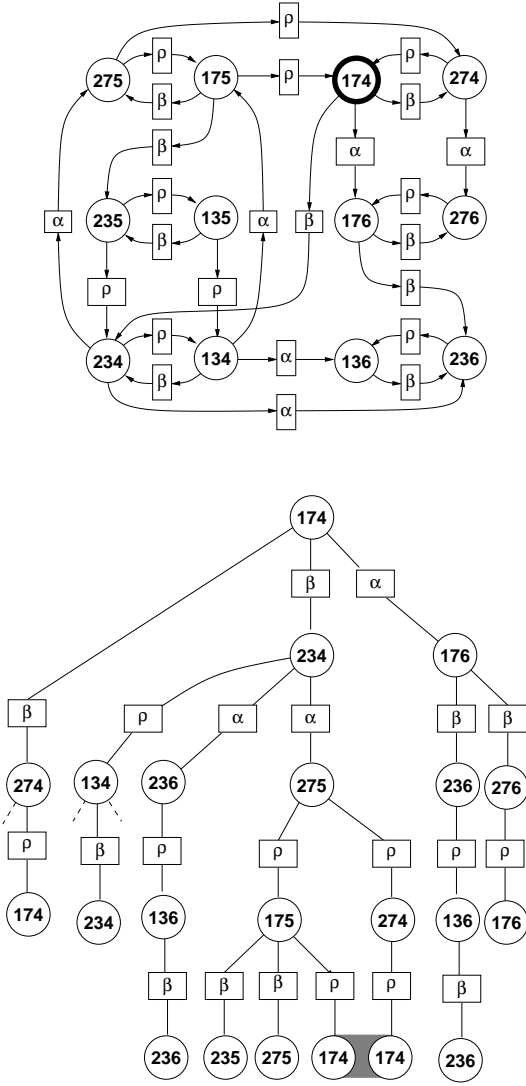


Fig. 8. Marking graph of example 1 (top), and unfolding (bottom).

marking graph of \mathcal{P} , denoted by $\mathcal{M}(\mathcal{P})$. It is a labeled Petri net whose places are labeled by the reachable markings of \mathcal{P} , shown by the combination of the places composing the different markings. We identify the places of the marking graph $\mathcal{M}(\mathcal{P})$ with the markings of \mathcal{P} . Then, $M[t]M'$ in \mathcal{P} iff $M \rightarrow \tau \rightarrow M'$ in $\mathcal{M}(\mathcal{P})$. Transition τ of $\mathcal{M}(\mathcal{P})$ is then labeled by transition t of \mathcal{P} . In Fig. 8 we have labeled instead the transitions τ of $\mathcal{M}(\mathcal{P})$ by the alarm labels $\lambda(t) = \alpha, \beta, \rho$ of the associated transitions t from \mathcal{P} .

The pre/postset of each transition of $\mathcal{M}(\mathcal{P})$ is a singleton, hence there is no concurrency, and $\mathcal{M}(\mathcal{P})$ represents an automaton. Note the diamond composed of the two branches $275 \rightarrow \rho \rightarrow 175 \rightarrow \rho \rightarrow 174$ and $275 \rightarrow \rho \rightarrow 274 \rightarrow \rho \rightarrow 174$, it represents the two possible interleavings of the concurrent transitions labeled by ρ in \mathcal{P} .

We can still regard $\mathcal{M}(\mathcal{P})$ as a Petri net, and consider its unfolding $\mathcal{U}_{\mathcal{M}(\mathcal{P})}$, shown in part in the second diagram (some flow relations are sketched in dashed, to save space). Now, we can safely merge the two conditions labeled by 174 in the bottom of this diagram. The reasons for this are the following: 1/ they are

both labeled by the same state (namely 174), hence they possess identical continuations, and, 2/ their causal closures are labeled by the same alarm sequence $\beta, \alpha, \rho, \rho$, i.e., explain the same sequences of alarms. Merging the two conditions labeled by 174 in the bottom of the second diagram yields a *lattice*, i.e., a labeled net with branching and joining conditions but no circuit, we denote it by $\mathcal{L}_{\mathcal{M}(\mathcal{P})}$. Lattices are not new, they are the data structures used when applying the Viterbi algorithm for maximum likelihood estimation of hidden state sequences in stochastic automata.

Being linear and not branching any more, $\mathcal{L}_{\mathcal{M}(\mathcal{P})}$ is a more compact data structure than the unfolding $\mathcal{U}_{\mathcal{M}(\mathcal{P})}$. The reason for merging the two places labeled by 174 in $\mathcal{U}_{\mathcal{M}(\mathcal{P})}$ is the diamond occurring in $\mathcal{M}(\mathcal{P})$. But this diamond manifests the concurrency of the two self-repairing transitions, and the unfolding $\mathcal{U}_{\mathcal{P}}$ of \mathcal{P} , shown in Fig. 2, already handles this properly: the marking 174 is not duplicated in $\mathcal{U}_{\mathcal{P}}$, unlike in $\mathcal{U}_{\mathcal{M}(\mathcal{P})}$. In fact, this lattice corresponds to a prefix of the unfolding shown in Fig. 2. The unfolding of Fig. 2 is more compact, but in turn, building co-sets requires some processing, whereas this requires no processing for the unfolding of Fig. 8, since co-sets are just places. Therefore, for applications in which memory constraints prevail over processing speed, unfoldings should be preferred. Still, the generalization of lattices to Petri nets is of interest, and their definition and use for diagnosis is investigated in the full paper [1].

IV. ALGORITHMS

In this section, we detail the algorithms for the construction of diagnosis nets. In subsection IV-B we consider the general case considered in theorem 1, this encompasses setups \mathbf{S}_1 and \mathbf{S}_2 of subsection II-B. Then, in subsection IV-C, we focus on \mathbf{S}_1 , for which we give an improved algorithm. In subsection IV-A we first describe the framework we need for the description of these algorithms.

A. Algorithms as pattern matching rules

As a prerequisite, we formally state an inductive construction of the unfolding $\mathcal{U}_{\mathcal{P}}$ of a Petri net \mathcal{P} in the form of a nested family of branching processes \mathcal{B} —this construction is borrowed from [16], it was illustrated in Fig. 2.

We use the following notations. The conditions of $\mathcal{U}_{\mathcal{P}}$ have the form (e, p) , where e is an event of $\mathcal{U}_{\mathcal{P}}$ and p a place of \mathcal{P} . Similarly, events of $\mathcal{U}_{\mathcal{P}}$ have the form (X, t) , where X is a co-set of conditions belonging to $\mathcal{U}_{\mathcal{P}}$, and t is a transition of \mathcal{P} . The homomorphism φ , from $\mathcal{U}_{\mathcal{P}}$ to \mathcal{P} , is given by $\varphi(e, p) = p$, and $\varphi(X, t) = t$. And the flow relation on $\mathcal{U}_{\mathcal{P}}$ is given by $\bullet(e, p) = e$, and $\bullet(X, t) = X$. Conditions (nil, p) are those having no input event, i.e., the distinguished symbol *nil* is used for the minimal conditions of $\mathcal{U}_{\mathcal{P}}$. Hence, we represent a branching process as a pair $\mathcal{B} = (B, E)$ of conditions and events, and the flow relation and homomorphism are derived implicitly, using the above convention. The term \mathbf{co}_B denotes the set of co-sets of B .

The set of *branching processes* of $\mathcal{P} = (P, T, \rightarrow, M_0)$ can be inductively constructed as follows:

- $(\{(nil, p), p \in M_0\}, \emptyset)$ is a branching process of \mathcal{P} .

- For (B, E) a branching process, $t \in T$, and $X \in \mathbf{co}_B$ such that $\varphi(X) = \bullet t$, then the following term is also a branching process of \mathcal{P} :

$$(B \cup \{(e, p) \mid p \in t^\bullet\}, E \cup \{e\}), \text{ where } e = (X, t). \quad (8)$$

If $e \notin E$ we call e a *possible extension* of (B, E) , and we call the corresponding extended branching process a *continuation* of (B, E) by e . The inductive construction (8) can be expressed in the form of a pattern matching rule:

$$\frac{\text{if precondition } \{\dots\} \text{ holds,}}{\text{then, possible extension } \{\dots\} \text{ and its postset } \{\dots\} \text{ result.}} : \frac{X \in \mathbf{co}_B, \varphi(X) = \bullet t}{e = (X, t)} \quad (9)$$

$$e^\bullet = \{(e, p) \mid p \in t^\bullet\}$$

where the three $\{\dots\}$ denote the corresponding three statements shown on the right hand side. In rule (9), it is understood that $e = (X, t)$ is a *possible extension of the current branching process, meaning that $e \notin E$ (the current event set)*, this will not be repeated in the sequel. Most importantly, **rule (9) applies asynchronously**, meaning that the continuation can be performed in any order, from the different co-sets which have possible extensions in the current branching process.

B. Asynchronous diagnosis

The raw rules: computing $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$. We first provide the rules for the computation of the unfolding $\mathcal{U}_{\mathcal{P}_1 \times \mathcal{P}_2}$, for two nets \mathcal{P}_1 and \mathcal{P}_2 . For two nets \mathcal{P}_1 and \mathcal{P}_2 , p_i (resp. t_i) denotes generically a place (resp. transition) of net \mathcal{P}_i , and the labeling map is denoted by λ_i . The homomorphism of the unfolding under construction is denoted by φ . Using these notations, we have the following rules for inductively constructing the branching processes (B, E) of $\mathcal{P}_1 \times \mathcal{P}_2$, whose union forms the unfolding $\mathcal{U}_{\mathcal{P}_1 \times \mathcal{P}_2}$, cf. (2)—in these rules, when the generic index i is used, the universal quantification $\forall i = 1, 2$ is understood:

$$\frac{X_i \in \mathbf{co}_{B_i}, \lambda_i(t_i) \text{ is private, and } \varphi(X_i) = \bullet t_i}{e_i = (X_i, t_i)} \quad (10)$$

$$e_i^\bullet = \{(e_i, p_i) \mid p_i \in t_i^\bullet\}$$

$$\frac{X_1 \cup X_2 \in \mathbf{co}_B, \lambda_1(t_1) = \lambda_2(t_2) : \varphi(X_i) = \bullet t_i}{e = (X_1 \cup X_2, (t_1, t_2))} \quad (11)$$

$$e^\bullet = \{(e, p) \mid p \in t_1^\bullet \cup t_2^\bullet\}$$

Rule (10) performs a local continuation involving a single component, whereas rule (11) performs a synchronized continuation. Thanks to theorem 1, the above rules (10,11) can be specialized to implement the inductive computation of the branching processes (B, E) of $\mathcal{P} \times \mathcal{A}$: simply discard rule (10) since no private label is involved. This yields:

$$\frac{X \cup X_{\mathcal{A}} \in \mathbf{co}_B, \lambda(t) = \lambda_{\mathcal{A}}(e^{\mathcal{A}}), \varphi(X) = \bullet t, \varphi(X_{\mathcal{A}}) = \bullet e^{\mathcal{A}}}{e = (X \cup X_{\mathcal{A}}, (t, e^{\mathcal{A}}))} \quad (12)$$

$$e^\bullet = \{(e, p) \mid p \in t^\bullet \cup e^{\mathcal{A}\bullet}\}$$

where $e^{\mathcal{A}}$ denotes a generic event of \mathcal{A} . Since the presence of the term $X \cup X_{\mathcal{A}}$ in the preset of the extension e always requires the corresponding precondition $X \cup X_{\mathcal{A}} \in \mathbf{co}_B$, we shall omit the

term $X \cup X_{\mathcal{A}} \in \mathbf{co}_B$ in the precondition of the rules in the sequel. Thus rule (12) will be simply written:

$$\frac{\lambda(t) = \lambda_{\mathcal{A}}(e^{\mathcal{A}}), \varphi(X) = \bullet t, \varphi(X_{\mathcal{A}}) = \bullet e^{\mathcal{A}}}{e = (X \cup X_{\mathcal{A}}, (t, e^{\mathcal{A}}))}.$$

$$e^\bullet = \{(e, p) \mid p \in t^\bullet \cup e^{\mathcal{A}\bullet}\}$$

Since continuations can occur from any co-set of the current branching process, the whole branching process must be continuously maintained, for possible continuation, along the construction of the unfolding. Of course this data structure is of rapidly increasing complexity, and this makes the general algorithm based on rule (12) quite cumbersome. Also, in this general case, explanations of an alarm can occur with an arbitrary long delay, but this is the unavoidable price to pay for handling asynchrony with no restriction on the allowed sensor setup.

Refining the rules. Let us investigate how to refine rule (12). Theorem 1 states that $\mathbf{diag}(\mathcal{A})$ is in one-to-one correspondence with the set of $\bar{\kappa}$'s satisfying (6). Thus, only a subnet of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ has to be computed, not all of it. We investigate this issue next. Consider the unfolding $\mathcal{U}_{\mathcal{P} \times \mathcal{A}} = (B, E)$. For \mathcal{A} fixed, and \mathcal{A}' a subnet of \mathcal{A} , denote by

$$\mathbf{explain} \mathcal{A}' \quad (13)$$

the maximal subnet \mathcal{U} of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, such that: 1/ all events of \mathcal{U} are labeled by events of \mathcal{A}' , and 2/ $\min(\mathcal{U})$ and $\max(\mathcal{U})$ are conditions. The subnet $\mathbf{explain} \mathcal{A}'$ collects all events of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ which can explain some alarm belonging to \mathcal{A}' . For $\mathcal{W} \subseteq \mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ a branching process of $\mathcal{P} \times \mathcal{A}$, set:

$$\mathcal{A}_{\mathcal{W}}^{\text{term}} = \max \{ \mathcal{A}' \subseteq \mathcal{A} \mid (\mathbf{explain} \mathcal{A}') \subseteq \mathcal{W} \}. \quad (14)$$

Then, $\mathcal{A}_{\mathcal{W}}^{\text{term}}$ is the *terminated* prefix of \mathcal{A} , i.e., no further continuation of \mathcal{W} will provide a new explanation for $\mathcal{A}_{\mathcal{W}}^{\text{term}}$. Symmetrically, keeping in mind that $\mathcal{U}_{\mathcal{P} \times \mathcal{A}} = (B, E)$, set:

$$\mathcal{A}_{\mathcal{W}}^{\text{fut}} = \max \{ \mathcal{A}' \subseteq \mathcal{A} \mid (\mathbf{explain} \mathcal{A}') \cap \mathcal{W} \cap E = \emptyset \} \quad (15)$$

Then, $\mathcal{A}_{\mathcal{W}}^{\text{fut}}$ collects the *future* alarms, that have not yet been considered at all in \mathcal{W} . Note the subset symbol \subseteq in (15), indicating that $\mathcal{A}_{\mathcal{W}}^{\text{fut}}$ is not a prefix of \mathcal{A} (it is in fact a postfix of \mathcal{A}). Also, the two “max” in (14) and (15) are well defined, since the corresponding sets of \mathcal{A}' 's are stable under union. In general, the set $\mathcal{A}_{\mathcal{W}}^{\text{act}}$ of *active alarms* satisfies:

$$\mathcal{A}_{\mathcal{W}}^{\text{act}} \triangleq \mathcal{A} \setminus (\mathcal{A}_{\mathcal{W}}^{\text{term}} \cup \mathcal{A}_{\mathcal{W}}^{\text{fut}}) \neq \emptyset, \quad (16)$$

and $\mathcal{A}_{\mathcal{W}}^{\text{act}}$ can even have cardinality greater than 1. This means that alarms cannot be processed in sequence, i.e., there is no on-line algorithm. We shall see, however, that $\mathcal{A}_{\mathcal{W}}^{\text{act}} = \emptyset$ holds for a certain increasing chain of \mathcal{W} 's, for setup \mathbf{S}_1 . In general, refined rules must maintain the triple $(\mathcal{A}_{\mathcal{W}}^{\text{term}}, \mathcal{A}_{\mathcal{W}}^{\text{act}}, \mathcal{A}_{\mathcal{W}}^{\text{fut}})$.

For \mathcal{A}' a prefix of \mathcal{A} such that no node of \mathcal{A}' is maximal in \mathcal{A} , denote by

$$\mathbf{stop}(\mathcal{A}') \quad (17)$$

the maximal subnet of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}'}$ possessing no continuation in $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, note that $\mathbf{stop}(\mathcal{A}')$ is a postfix of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}'}$, and no continuation of it will explain alarms belonging to $\mathcal{A} \setminus \mathcal{A}'$. Hence $\mathbf{stop}(\mathcal{A}')$ should be pruned prior to further performing continuations of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}'}$.

For an arbitrary branching process \mathcal{W} of $\mathcal{P} \times \mathcal{A}$, we must prune $\mathcal{W}^{term} \triangleq \mathbf{stop}(\mathcal{A}_{\mathcal{W}}^{term})$, where $\mathcal{A}_{\mathcal{W}}^{term}$ is defined in (14), and keep only $\mathcal{W}^{act} \triangleq \mathcal{W} \setminus \mathcal{W}^{term}$, from which continuation can proceed.

By maintaining the above objects along the steps of the algorithm, refined versions of rule (12) can be derived. Using this technique, in the next subsection we focus on setup \mathbf{S}_1 and provide for it an improved algorithm.

C. Asynchronous diagnosis, an improved algorithm for setup \mathbf{S}_1

Here we investigate the computation of the diagnosis net $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, and then of $\mathbf{diag}(\mathcal{A})$, for the case of \mathcal{A} being a totally ordered alarm pattern of \mathcal{P} —this corresponds to setup \mathbf{S}_1 .

The reader may think of a very simple solution for this particular case, since \mathcal{A} is strictly a firing sequence. Try to fire this sequence in the Petri net from the initial marking. Each time an ambiguity occurs (two transitions may be fired explaining the next event in \mathcal{A}). Then, a new copy of the trial is instanciated (a new Petri net instance), to follow the additional firing sequence. Each time no transition can be fired in a trial to explain a new event, the trial is abandoned. Then, at the end of \mathcal{A} , all the behaviours explaining \mathcal{A} have been obtained. The remaining issue is to find a compact data structure to represent all these behaviours: again, unfoldings are the adequate answer. In fact, this is what our approach below directly provides, by specializing the general case.

We start with some lemmas. The first lemma establishes that on-line algorithms exist for setup \mathbf{S}_1 .

Lemma 1: Let $\mathcal{A}' \sqsubseteq \mathcal{A}$ be a prefix of \mathcal{A} . Then:

$$\mathcal{U}_{\mathcal{P} \times \mathcal{A}'} \sqsubseteq \mathbf{explain} \mathcal{A}'. \quad (18)$$

If \mathcal{A} is totally ordered, then we have:

$$\mathcal{U}_{\mathcal{P} \times \mathcal{A}'} = \mathbf{explain} \mathcal{A}'. \quad (19)$$

Formula (19) says that, if \mathcal{A} is totally ordered, then $\mathcal{U}_{\mathcal{P} \times \mathcal{A}'}$ contains *all* explanations of \mathcal{A}' . In this case, as soon as $\mathcal{U}_{\mathcal{P} \times \mathcal{A}'}$ has been constructed, we can forget \mathcal{A}' , this justifies the consideration of on-line algorithms—we insist that this does not hold in general, cf. (16)!

Proof: Inclusion (18) is obvious, so we need only to prove equality (19) under the assumption that \mathcal{A} is totally ordered. This is the result of interest. It is trivial if $\mathcal{A}' = \emptyset$, so we can assume that this is not the case.

In the sequel of the proof, symbols $\bar{e}\bar{f}$ denote events of the unfolding $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, hence, using the notations of (12), \bar{e} has the form:

$$\bar{e} = (X \cup \bar{X}, (t, e^{\mathcal{A}})),$$

and $\phi_{\mathcal{A}}(\mathbf{proj}_{\mathcal{A}}(\bar{e})) = \bar{e}$, where $\phi_{\mathcal{A}}$ denotes the labeling map of $\mathbf{proj}_{\mathcal{A}}(\mathcal{U}_{\mathcal{P} \times \mathcal{A}})$ (cf. (5)). Also, $\bar{\prec}(\bar{e})$ denotes the configuration

spanned by the causal closure of \bar{e} in $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$. Similar notations and remarks hold for \bar{f} .

Pick $\bar{e} \in \mathbf{explain} \mathcal{A}$, and set $\bar{\kappa} = \bar{\prec}(\bar{e})$. Pick $\bar{f} \in \bar{\kappa}$. Since $\bar{f} \bar{\prec} \bar{e}$, then either $\bar{e} \perp \bar{f}$ or $\bar{f} \bar{\prec} \bar{e}$ holds ($\bar{e} \bar{\prec} \bar{f}$ is impossible, by definition of alarm patterns). But, since \mathcal{A} is totally ordered, then only $\bar{f} \bar{\prec} \bar{e}$ can hold, thus \bar{f} must belong to \mathcal{A}' , since $\bar{e} \in \mathcal{A}'$. Since this holds $\forall \bar{f} \in \bar{\kappa}$, then $\bar{\kappa} \subseteq \mathcal{U}_{\mathcal{P} \times \mathcal{A}'}$, therefore $\mathbf{explain} \mathcal{A}' \subseteq \mathcal{U}_{\mathcal{P} \times \mathcal{A}'}$, this proves the lemma. \diamond

For the following results, we assume \mathcal{A} totally ordered. For \mathcal{U} a subnet of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, denote by $\#(\mathcal{U})$ the subnet of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ comprising the nodes that are in conflict with *every* node of \mathcal{U} . The following theorem indicates how the pruning introduced in subsection IV-B should be performed, it refines theorem 1 for the case in which \mathcal{A} is totally ordered:

Theorem 2: Assume that \mathcal{A} is totally ordered.

1. If \mathcal{A}' is the maximal strict prefix of \mathcal{A} , then $\mathbf{diag}(\mathcal{A})$ coincides with the set of all maximal configurations of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}} \setminus \mathbf{stop}(\mathcal{A}')$.
2. Consider a chain $\mathcal{A}' \sqsubset \mathcal{A}'' \sqsubseteq \mathcal{A}$ of prefixes of \mathcal{A} . Then:

$$\mathbf{stop}(\mathcal{A}') = \#(\mathcal{V}), \text{ where } \mathcal{V} \triangleq \mathcal{U}_{\mathcal{P} \times \mathcal{A}''} \setminus \mathcal{U}_{\mathcal{P} \times \mathcal{A}'}. \quad (20)$$

Proof: Point 1 follows from the definition (17) of $\mathbf{stop}(\mathcal{A}')$. Then, by lemma 1, we have $\mathcal{V} = (\mathbf{explain} \mathcal{A}'' \setminus \mathbf{explain} \mathcal{A}')$, from which point 2 follows. \diamond

The following lemma is of lesser importance, but it will be useful for further optimizing our algorithm, by restricting the set of co-sets than can serve for possible continuation. For \mathcal{U} a subnet of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, denote by $\perp(\mathcal{U})$ the subnet of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ consisting of the nodes that are concurrent with *some* node of \mathcal{U} —note the difference with the definition of $\#(\mathcal{U})$.

Lemma 2: Let \mathcal{A}' , \mathcal{A}'' , and \mathcal{V} be as in point 2 of theorem 2. Set

$$\mathbf{ext}(\mathcal{A}'') \triangleq \mathcal{V} \cup \perp(\mathcal{V}) \quad (20)$$

Then all possible extensions of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}''}$ have their preset contained in $\mathbf{ext}(\mathcal{A}'')$.

Proof: Assume this is not the case. Hence there exists a co-set X not contained in the subnet sitting on the right hand side of (20), and some event $e \in (\mathcal{U}_{\mathcal{P} \times \mathcal{A}} \setminus \mathcal{U}_{\mathcal{P} \times \mathcal{A}''})$, such that $\bullet e = X$. Hence we must have:

$$X \cap (\prec(\mathcal{V}) \cup \#(\mathcal{V})) \neq \emptyset,$$

which implies that $e \in \#(\mathcal{V})$, a contradiction with theorem 2. \diamond

Using the above results, successive optimizations of the generic rule (12) are performed in several steps.

1. *On-line computation of the successive branching processes, (B, E) , of unfolding $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$. Write $\mathcal{A} = (B_{\mathcal{A}}, E_{\mathcal{A}}, \rightarrow_{\mathcal{A}}, \lambda_{\mathcal{A}})$, with:*

$$\begin{aligned} B_{\mathcal{A}} &= \{b_0, b_1, b_2, \dots, b_n, \dots\}, \\ E_{\mathcal{A}} &= \{e_1, e_2, \dots, e_n, \dots\}, \lambda_{\mathcal{A}}(e_k) \in A, \\ \forall n > 0 &: b_{n-1} \rightarrow_{\mathcal{A}} e_n \rightarrow_{\mathcal{A}} b_n, \end{aligned} \quad (21)$$

where we recall that A is the alphabet of possible alarms—the superscript \mathcal{A} has been removed from the events and conditions

of \mathcal{A} , for the sake of clarity. In other words, the flow relation is obtained by interleaving alternatively one condition from $B_{\mathcal{A}}$ and one event from $E_{\mathcal{A}}$. Using these notations, rule (12) rewrites as follows:

$$\frac{\lambda(t) = \lambda_{\mathcal{A}}(e_n), \varphi(X) = \bullet t, \varphi(X_{\mathcal{A}}) = b_{n-1}}{e = (X \cup X_{\mathcal{A}}, (t, e_n))} \quad (22)$$

$$e^\bullet = \{(e, p) \mid p \in t^\bullet\} \cup \{(e, b_n)\}$$

Denote by $\mathcal{A}_n = (b_0 \rightarrow_{\mathcal{A}} e_1 \rightarrow_{\mathcal{A}} b_1 \rightarrow_{\mathcal{A}} \dots \rightarrow_{\mathcal{A}} b_n)$ the prefix of length n of \mathcal{A} , and apply lemma 1 with $\mathcal{A}' := \mathcal{A}_n$, we get:

$$(\hat{B}_n, \hat{E}_n) \stackrel{\Delta}{=} \mathbf{explain} \mathcal{A}_n = \mathcal{U}_{\mathcal{P} \times \mathcal{A}_n} \quad (23)$$

(recall that we represent branching processes as a pair of condition and event sets). Formula (23) expresses that we can forget about \mathcal{A}_n as soon as $\mathcal{U}_{\mathcal{P} \times \mathcal{A}_n}$ has been computed, and implement an *on-line* computation of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}_n}$:

$$(\hat{B}_n, \hat{E}_n) = \mathbf{R}_n^{(22)}(\hat{B}_{n-1}, \hat{E}_{n-1}), \text{ where} \quad (24)$$

$$\mathbf{R}_n^{(22)} \stackrel{\Delta}{=} [\forall t : \mathbf{R}_{n,t}^{(22)}],$$

and $\mathbf{R}_{n,t}^{(22)}$ denotes rule (22), for n and t seen as parameters. Formula (24) defines our *on-line* algorithm.

2. *Computing the set* $\{\bar{\kappa} \in \mathbf{config}(\mathcal{U}_{\mathcal{P} \times \mathcal{A}}) : \mathbf{proj}_{\mathcal{A}}(\bar{\kappa}) = \mathcal{A}\}$. Theorem 1 indicates that we need only to compute those configurations $\bar{\kappa}$ of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, such that $\mathbf{proj}_{\mathcal{A}}(\bar{\kappa}) = \mathcal{A}$. In $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, some configurations, while being maximal (for set inclusion) in $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, explain only a strict prefix of \mathcal{A} . To represent $\mathbf{diag}(\mathcal{A})$ exactly, such configurations must be removed. For this we use theorem 2. Consider

$$\delta \hat{E}_n \stackrel{\Delta}{=} \hat{E}_n \setminus \hat{E}_{n-1},$$

the set of events added to \hat{E}_{n-1} by applying the rule $\mathbf{R}_n^{(22)}$, and let $\#(\delta \hat{E}_n)$ denote the set of nodes belonging to (\hat{B}_n, \hat{E}_n) that are in conflict with every node of $\delta \hat{E}_n$. Apply theorem 2 with $\mathcal{A}' := \mathcal{A}_{n-1}, \mathcal{A}'' := \mathcal{A}_n$, we deduce that $\mathbf{stop}(\mathcal{A}_{n-1}) = \#(\delta \hat{E}_n)$, i.e.,

$$\text{the nodes belonging to } \#(\delta \hat{E}_n) \text{ cannot belong} \quad (25)$$

$$\text{to a configuration explaining alarm } n.$$

Thus, for computing the set $\{\bar{\kappa}_n \in \mathbf{config}(\mathcal{U}_{\mathcal{P} \times \mathcal{A}}) : \mathbf{proj}_{\mathcal{A}}(\bar{\kappa}_n) = \mathcal{A}_n\}$ we must prune the nodes belonging to $\#(\delta \hat{E}_n)$.

This pruning can then be interleaved with the successive application of the on-line rule (24). Performing this yields the *desired sequence* (B_n, E_n) of branching processes, and all maximal configurations, $\bar{\kappa}_n$, of (B_n, E_n) , are such that $\mathbf{proj}_{\mathcal{A}}(\bar{\kappa}_n) = \mathcal{A}_n$. Therefore, the following post-processing is applied after rule (24):

$$\mathbf{pruneR}_n : \text{remove } \#(\delta E_n) \text{ from } (B_n, E_n), \quad (26)$$

and rule (24) is modified as follows:

$$\mathbf{R}_n^{(22)}; \mathbf{pruneR}_n. \quad (27)$$

The pruning mechanism (26) is illustrated in Fig. 9, the reader

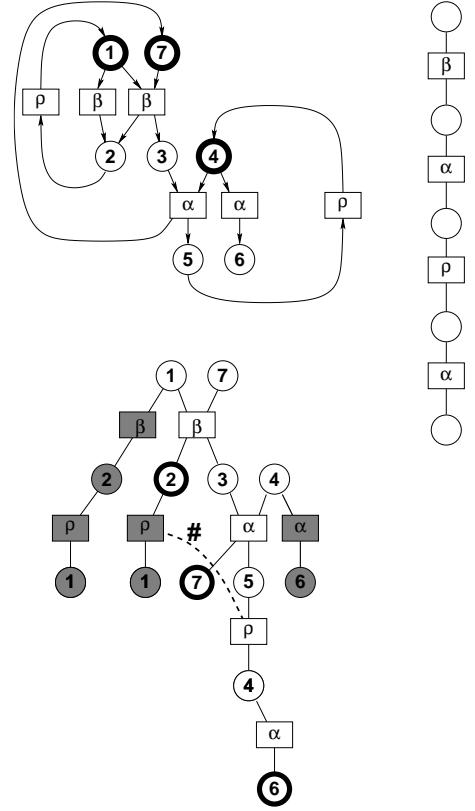


Fig. 9. The pruning mechanism (26), and optimization (28).

should compare with Fig. 7. In this figure, we have extended the \mathcal{A} shown in Fig. 7 by adding one more alarm labeled α (top right). The branching process shown in the bottom is a continuation of the one shown in Fig. 7. The result of the pruning mechanism (26) is depicted in dark grey: corresponding nodes are pruned from the updated $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, which is therefore the white part of the diagram on the bottom. Note that the ambiguity has been removed, since the remaining net is now itself a single configuration. In fact, this figure shows directly this pruning mechanism on the restriction of diagnosis net $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, to the subset of its nodes that are labeled by nodes from \mathcal{P} .

3. *Optimizing.* We can still optimize rule (27) by noting that, in the term (B_n, E_n) resulting from applying this rule, not all places from B_n can serve for future continuations of (B_n, E_n) —compare this situation with the general one discussed at the end of subsection IV-B. For this, we use lemma 2. Denote by $\perp(\delta E_n)$ the set of places belonging to (B_n, E_n) that are concurrent with *some* event belonging to δE_n —note the difference with the former definition of $\#(\delta E_n)$. Then, by lemma 2 we know that

$$\text{only the nodes belonging to } \delta E_n^\bullet \cup \perp(\delta E_n) \quad (28)$$

$$\text{can serve for future continuations of } (B_n, E_n).$$

Using claim (28), rule (22) rewrites as follows, note the modification of the pre-condition:

$$\mathbf{R}_{n,t}^{(29)} : \frac{\lambda(t) = \lambda_{\mathcal{A}}(e_n), \varphi(X) = \bullet t, \varphi(X_{\mathcal{A}}) = b_{n-1}}{e = (X \cup X_{\mathcal{A}}, (t, e_n))} \quad (29)$$

$$e^\bullet = \{(e, p) \mid p \in t^\bullet\} \cup \{(e, b_n)\}$$

The additional precondition is then updated as follows, prior to handling the $n + 1$ st alarm:

$$\delta E_n^\bullet \cup \perp(\delta E_n) = \delta E_n^\bullet \cup (\perp(\delta E_{n-1}) \setminus \bullet \delta E_n), \quad (30)$$

we call \mathbf{optimR}_n the so defined rule. Then, post-processing (26) applies after rule (29) as well, hence the optimized rule becomes:

$$\mathbf{R}_n^{(29)}; \mathbf{pruneR}_n; \mathbf{optimR}_n. \quad (31)$$

This optimization is illustrated in Fig. 9. In this figure, the set $\delta E_n^\bullet \cup \perp(\delta E_n)$ consists of the three conditions encircled in thick, in the diagnosis net shown on the right hand side. Hence co-sets not contained in this set need not be tested, for possible continuation. Note that, unlike the pruning, this optimization does not modify the constructed branching process.

4. Maintaining co-sets. So far we have ignored the need for testing the condition $X \in \mathbf{co}_{B_{n-1}}$, see the remark at the end of subsection IV-A. But the optimization $\mathbf{optimR}(n)$ applies, hence only a postfix of the whole branching process is explored for performing the continuation. Therefore we should avoid exploring backward the entire branching process under construction, in order to check the co-set property. So we need to maintain and update *explicitly* the co-set property, restricted to $\delta E_n^\bullet \cup \perp(\delta E_n)$. We discuss this now.

Focus on rule (29). The following formula holds, to update the co-set property while processing the n th alarm:

$$\mathbf{co}_B := \mathbf{co}_B \cup \left(\bigcup_{Y \in \mathbf{co}_B, Y \supseteq X} Y[X \leftarrow e^\bullet] \right), \quad (32)$$

where we denote by $Y[X \leftarrow e^\bullet]$ the co-set Y in which X has been substituted with e^\bullet . Formula (32) possesses the following initialization and termination conditions:

$$\begin{aligned} \text{initialization} & : \mathbf{co}_B := \mathbf{co}_{B_{n-1}}, \\ \text{termination} & : \mathbf{co}_{B_n} := \mathbf{co}_B. \end{aligned}$$

Finally, rule $\mathbf{R}_{n,t}^{(29)}$ refines as:

$$\mathbf{R}_{n,t}^{(33)} : \frac{X \cup X_{\mathcal{A}} \subseteq (\delta E_{n-1}^\bullet \cup \perp(\delta E_{n-1}))}{\begin{aligned} \lambda(t) &= \lambda_{\mathcal{A}}(e_n), \varphi(X) = \bullet t, \varphi(X_{\mathcal{A}}) = b_{n-1} \\ e &= (X \cup X_{\mathcal{A}}, (t, e_n)) \\ e^\bullet &= \{(e, p) \mid p \in t^\bullet\} \cup \{(e, b_n)\} \\ \mathbf{co}_B &:= \mathbf{co}_B \cup \left(\bigcup_{Y \in \mathbf{co}_B, Y \supseteq X} Y[X \leftarrow e^\bullet] \right) \end{aligned}} \quad (33)$$

And the refined on-line algorithm is obtained by substituting $\mathbf{R}_{n,t}^{(33)}$ for $\mathbf{R}_{n,t}^{(29)}$ in (31). Note that maintaining on-line the concurrency relation is of low cost in this case, in contrast to the general case where continuations can be performed from far in the interior of the net under construction.

V. DISCUSSION

A net unfolding approach to on-line asynchronous diagnosis was presented. This true concurrency approach is suited to distributed and asynchronous systems in which no global state and no global time is available, and therefore a partial order model of time is considered. In the present paper, our basic tool was

the net *unfolding*, a branching structure representing the set of configurations of a Petri net, with asynchronous semantics, local states, and partially ordered time. Diagnosis nets were introduced as a way to encode all solutions of a diagnosis problem. They avoid the well-known state explosion problem, that typically results from having concurrent components in a distributed system interacting asynchronously. Whereas state explosion is kept under control, the computing cost of performing the diagnosis on-line increases (due to the need to compute co-sets); but this is typically a preferred tradeoff for the diagnosis of complex asynchronous systems involving significant concurrency.

It is worth saying what this paper does *not* consider. We do not follow a diagnoser approach. One can view a diagnoser as a “compiled” algorithm for diagnosis. It consists in pre-computing a finite state machine which accepts alarm events, and has states labeled by, e.g., visited faults. In contrast, our approach can be seen as an “interpreted” one, since our diagnosis nets are computed, on-line, by using only the original Petri net structure. Also, we did not investigate issues of diagnosability. Diagnosers for unbounded asynchronous diagnosis and related diagnosability issues have not been considered in the literature, at least to our knowledge. We believe this could be performed by using so-called *complete prefixes* of the unfolding, see [15][16].

Complexity issues have not been addressed. However, the following pragmatic argument can be given to justify the use of unfoldings. Complete prefixes of unfoldings have been used for model checking, an area in which practical complexity is of paramount importance [29][14][15][16].

Various extensions of this work are under progress. The algorithms developed in this paper return all explanations as a diagnosis. Our target application—fault management in telecommunications networks—typically exhibits a great deal of ambiguity. Hence it is of interest to return (the) most likely explanation(s). Probabilistic versions of the present work have been developed for this purpose, see [2][3][19], [20], and [5] for a theory of corresponding stochastic processes.

Since our algorithm is interpreted, not compiled, it can be extended to asynchronous systems subject to *dynamic* changes in their structure—this is typically a situation encountered in network management. This feature favors the use of diagnosis nets instead of the pre-computed diagnosers.

Then, this study is clearly an intermediate step toward *distributed* diagnosis, in which diagnosis is performed jointly by a network of supervisors communicating asynchronously. Papers [17][18][19], [20] are a first attempt toward distributed diagnosis. These references concentrate on how to compute the set of configurations $\mathbf{diag}(\mathcal{A})$ in a distributed way, but they do not consider the issue of how to represent $\mathbf{diag}(\mathcal{A})$ efficiently via unfoldings. The techniques of the present paper therefore should be combined with those of [17][18][19], [20] for getting an efficient solution for the distributed case, this topic will be the subject of a forthcoming paper.

The robustness of algorithms against alarm losses or communications failures needs to be investigated. Also, due to the systems complexity, there is little hope indeed, that an exact model can be provided, hence we need to develop diagnosis methods that work based on an incomplete model, i.e., a model not able to explain all observed behaviours.

Last but not least, getting the system model itself is a bottleneck, for complex distributed systems such as, e.g., telecommunications network management systems. The issue of how to partially automatize the model construction is investigated in [28].

ACKNOWLEDGEMENT. *The authors gratefully acknowledge the reviewers, whose remarks contributed to improving a first version of this paper.*

REFERENCES

- [1] Full version of this paper
http://www.irisa.fr/sigma2/benveniste/pub/B_al_asdiag_2001.html
- [2] A. Aghasaryan, E. Fabre, A. Benveniste, R. Boubour, C. Jard. A Petri net approach to fault detection and diagnosis in distributed systems. Part II: extending Viterbi algorithm and HMM techniques to Petri nets. *CDC'97 Proceedings*, San Diego, December 1997.
- [3] A. Aghasaryan, E. Fabre, A. Benveniste, R. Boubour, C. Jard. Fault Detection and Diagnosis in Distributed Systems : an Approach by Partially Stochastic Petri nets, *Discrete Event Dynamic Systems: theory and application*, special issue on Hybrid Systems, vol. 8, pp. 203-231, June 98.
- [4] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella. Diagnosis of large active systems. *Artificial Intelligence* 110: 135-183, 1999.
- [5] A. Benveniste, E. Fabre, and S. Haar. "Markov nets: probabilistic models for distributed and concurrent systems". Irisa Research Report 1415, September 2001, submitted for publication, <http://www.irisa.fr/sigma2/benveniste/pub/BIGFH2000.html>
- [6] R. Boubour, C. Jard, A. Aghasaryan, E. Fabre, A. Benveniste. A Petri net approach to fault detection and diagnosis in distributed systems. Part I: application to telecommunication networks, motivations and modeling. *CDC'97 Proceedings*, San Diego, December 1997.
- [7] A.T. Bouloutas, G. Hart, and M. Schwartz. Two extensions of the Viterbi algorithm. *IEEE Trans. on Information Theory*, 37(2):430-436, March 1991.
- [8] A.T. Bouloutas, S. Calo, and A. Finkel. Alarm correlation and fault identification in communication networks. *IEEE Trans. on Communications*, 42(2/3/4), 1994.
- [9] C. Cassandras and S. Laforune. *Introduction to discrete event systems*. Kluwer Academic Publishers, 1999.
- [10] R. Debouk, S. Laforune, and D. Teneketzis. Coordinated decentralized protocols for failure diagnosis of discrete event systems. *Discrete Event Dynamic Systems: theory and application*. 10(1/2), 33-86, 2000.
- [11] R. Debouk, S. Laforune, and D. Teneketzis. On the effect of communication delays in failure diagnosis of decentralized discrete event systems. Control group report CGR00-04, Univ. of Michigan at Ann Arbor, submitted for publication, 2001.
- [12] J. Desel, and J. Esparza. *Free Choice Petri Nets*. Cambridge University Press, 1995.
- [13] J. Engelfriet. *Branching Processes of Petri Nets*. Acta Informatica 28, 1991, pp 575-591.
- [14] J. Esparza. Model Checking Using Net Unfoldings. *Sci. of Comp. Prog.* 23:151-195, 1994.
- [15] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In T. Margaria and B. Steffen Eds., *Proc. of TACACS'96*, LNCS 1055, 87-106, 1996. Extended version to appear in *Formal Methods in System Design*, 2000.
- [16] J. Esparza, and S. Römer. An unfolding algorithm for synchronous products of transition systems, in *proceedings of CONCUR'99*, LNCS 1664, Springer Verlag, 1999.
- [17] E. Fabre, A. Benveniste, C. Jard, L. Ricker, and M. Smith. Distributed state reconstruction for discrete event systems. Proc. of the 2000 *IEEE Control and Decision Conference (CDC'2000)*, Sydney, Dec. 2000.
- [18] E. Fabre, A. Benveniste, C. Jard. Distributed diagnosis for large discrete event dynamic systems. In *Proc of the IFAC congress*, Jul. 2002.
- [19] E. Fabre. Compositional models of distributed and asynchronous dynamical systems. In *Proc of the 2002 IEEE Conf. on Decision and Control*, 1-6, Dec. 2002, Las Vegas, 2002.
- [20] E. Fabre. Monitoring distributed systems with distributed algorithms. In *Proc of the 2002 IEEE Conf. on Decision and Control*, 411-416, Dec. 2002, Las Vegas, 2002.
- [21] R.G. Gardner, and D. Harle. Methods and systems for alarm correlation. In *GlobeCom 96*, London, November 1996.
- [22] C.N. Hadjicostis, and G.C. Verghese. Monitoring discrete event systems using Petri net embeddings. in *Proc. of Application and theory of Petri nets 1999*, 188-208.
- [23] A. Giua. —PN state estimators based on event observation. *Proc. 36th Int. Conf. on Decision and Control*, San Diego, Ca, 4-86-4091, 1997.
- [24] A. Giua, and C. Seatzu. Observability of Place/Transition Nets. Preprint, 2001.
- [25] K.X. He and M.D. Lemmon. Liveness verification of discrete-event systems modeled by n -safe Petri nets. in *Proc. of the 21st Int. Conf. on Application and Theory of Petri Nets*, Denmark, June 2000.
- [26] K.X. He and M.D. Lemmon. On the existence of liveness-enforcing supervisory policies of discrete-event systems modeled by n -safe Petri nets. in *Proc. of IFAC'2000 Conf. on Control Systems Design*, special session on Petri nets, Slovakia, June 2000.
- [27] I. Katsela, A.T. Bouloutas, and S. Calo. Centralized vs distributed fault localisation. *Integrated Network Management IV*, A.S. Sethi, Y. Raynaud, and F. Faure-Vincent, Eds. Chapman and Hall, 251-261, 1995.
- [28] A. Aghasaryan, C. Dousson, E. Fabre, Y. Pencolé, A. Osmani. Modeling Fault Propagation in Telecommunications Networks for Diagnosis Purposes. XVIII World Telecommunications Congress 22-27 September 2002 - Paris, France. Available: http://www.irisa.fr/sigma2/benveniste/pub/topic_distribdiag.html
- [29] K. McMillan. Using Unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: *4th Workshop on Computer Aided Verification*, pp. 164-174, 1992.
- [30] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures, and domains. Part I. *Theoretical Computer Science* 13:85-108, 1981.
- [31] W. Reisig. *Petri nets*. Springer Verlag, 1985.
- [32] L. Rozé and M-O. Cordier. Diagnosing discrete event systems: extending the diagnoser approach to deal with telecommunications networks. *Discrete Event Dynamic Systems: theory and application*, special issue on WODES'98, 12(1), 43-82, Jan. 2002.
- [33] G. Rozenberg and J. Engelfriet. Elementary Net Systems. In: *Lectures on Petri Nets I: Basic Models*. LNCS 1491, pp. 12-121, Springer, 1998.
- [34] A. Sahraoui, H. Atabakhche, M. Courvoisier, and R. Valette. Joining Petri nets and knowledge-based systems for monitoring purposes. *Proc. of the IEEE Int. Conf. on Robotics Automation*, 1160-1165, 1987.
- [35] M. Sampath, R. Sengupta, S. Laforune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Trans. Autom. Control* 40(9), 1555-1575, 1995.
- [36] R. Sengupta. Diagnosis and communications in distributed systems. In *Proc. of WODES 1998, international Workshop On Discrete Event Systems*, 144-151, IEE, London, England, 1998.
- [37] S. Tripakis. Undecidable problems of decentralized observation and control. In *Proc. of the 40th IEEE Conf. on Decision and Control*, Orlando, Dec. 2001.
- [38] G. Winskel. Event structures. In *Advances in Petri nets*, LNCS vol. 255, 325-392, Springer Verlag, 1987.