Distributed Monitoring of Concurrent and Asynchronous Systems*

Albert Benveniste[1], Stefan Haar[1], Eric Fabre[1], and Claude Jard[2]

[1] Irisa/INRIA, Campus de Beaulieu, 35042 Rennes cedex, France
Albert.Benveniste@irisa.fr
http://www.irisa.fr/sigma2/benveniste/
[2] Irisa/CNRS, Campus de Beaulieu, 35042 Rennes cedex, France

Abstract. Developing applications over a distributed and asynchronous architecture without the need for synchronization services is going to become a central track for distributed computing. This research track will be central for the domain of autonomic computing and self-management. Distributed constraint solving, distributed observation, and distributed optimization, are instances of such applications. This paper is about distributed observation: we investigate the problem of distributed monitoring of concurrent and asynchronous systems, with application to distributed fault management in telecommunications networks.
Our approach combines two techniques: compositional unfoldings to handle concurrency properly, and a variant of graphical algorithms and belief propagation, originating from statistics and information theory.

Keywords: asynchronous, concurrent, distributed, unfoldings, event structures, belief propagation, fault diagnosis, fault management

1 Introduction

Concurrent and distributed systems have been at the heart of computer science and engineering for decades. Distributed algorithms [18, 25] have provided the sound basis for distributed software infrastructures, providing correct communication and synchronization mechanisms, and fault tolerance for distributed applications. Consensus and group membership have become basic services that a safe distributed architecture should provide. Formal models and mathematical theories of concurrent systems have been essential to the development of languages, formalisms, and validation techniques that are needed for a correct design of large distributed applications.

However, the increasing power of distributed computing allows the development of applications in which the distributed underlying architecture, the function to be performed, and the data involved, are tightly interacting. Distributed optimization is a generic example. In this paper, we consider another instance of

* This work is or has been supported in part by the following projects: MAGDA, MAGDA2, funded by the ministery of research. Other partners of these projects were or are: Alcatel, France Telecom R&D, Ilog, Paris-Nord University.

R. Amadio, D. Lugiez (Eds.): CONCUR 2003, LNCS 2761, pp. 1–26, 2003.

those problems, namely the problem of inferring, from measurements, the hidden internal state of a distributed and asynchronous system. Such an inference has to be performed also in a distributed way. An important application is *distributed alarm correlation and fault diagnosis* in telecomunications networks, which motivated this work.

The problem of recovering state histories from observations is pervasive throughout the general area of information technologies. As said before, it is central in the area of distributed algorithms [18, 25], where it consists in searching for globally coherent sets of available local states, to form the global state. In this case the local states are available. Extending this to the case when the local states themselves must be inferred from observations, has been considered in other areas than computer science. For instance, estimating the state trajectory from noisy measurements is central in control engineering, with the Kalman filter as its most popular instance [16]; the same problem is considered in the area of pattern recognition, for stochastic finite state automata, in the theory of Hidden Markov Models [24]. For both cases, however, no extension exists to handle distributed systems. The theory of Bayesian networks in pattern recognition addresses the problem of distributed estimation, by proposing so-called belief propagation algorithms, which are chaotic and asynchronous iterations to perform state estimation from noisy measurements [19, 20, 23]. On the other hand, systems with dynamics (e.g., automata) are not considered in Bayesian networks. Finally, fault diagnosis in discrete event systems (e.g., automata) has been extensively studied [6, 27], but the problem of distributed fault diagnosis for distributed asynchronous systems has not been addressed.

This paper is organized as follows. Our motivating application, namely distributed alarm correlation and fault diagnosis, is discussed in Sect. 2. Its main features are: the concurrent nature of fault effect propagation, and the need for distributed supervision, where each supervisor knows only the restriction, to its own domain, of the global system model. Our goal is to compute *a coherent set of local views for the global status of the system,* for each supervisor. We follow a true concurrency approach. A natural candidate for this are 1-safe Petri nets with branching processes and unfoldings. Within this framework, we discuss in Sect. 3 a toy example in detail. The mathematical background is recalled in Sect. 4.1. As our toy example shows, two non classical operators are needed: a projection (to formalize what a local view is), and a composition (to formalize the cooperation of supervisors for distributed diagnosis). Occurrence nets, branching processes, and unfoldings will be shown to be not closed under projection, and thus inadequate to support these operations. Therefore, projection and composition are introduced for event structures (in fact, for "condition structures", a variant of event structures in which events are re-interpreted as conditions). This material is developed in Sect. 4.2. It allows us to formally express our problem, which is done in Sect. 5.

With complex interaction and several supervisors, the algorithm quickly becomes intractable, with no easy implementation. Fortunately, it is possible to organize the problem into a high-level *orchestration* of low-level primitives, expressed in terms of projections and compositions of condition structures. This

orchestration is obtained by deriving, for our framework, a set of key properties relating condition structures, their projections, and their compositions. Further analysing these key properties shows that they are shared by basic problems such as: distributed combinatorial constraint solving, distributed combinatorial optimization, and Bayesian networks estimation. Fortunately, chaotic distributed and asynchronous iterations to solve these problems have been studied; perhaps the most well-known version of these is that of *belief propagation* in belief networks [19, 20, 23]. Our high-level orchestration is derived from these techniques, it is explained in Sect. 6. In addition, this approach allows us to consider maximum likelihood estimation of fault scenarios, by using a probabilistic model for fault propagation.

Missing details and proofs can be found in the extended version [1].

2 Motivating Application: Fault Management in Telecommunication Networks [3]

Distributed self-management is a key objective in operating large scale infrastructures. Fault management is one of the five classical components of management, and our driving application. Here, we consider a distributed architecture in which each supervisor is in charge of its own domain, and the different supervisors cooperate at constructing a set of *coherent* local views for their repective domains. Of course, the corresponding global view should never be actually computed.

To ensure modularity, network management systems are decomposed into interconnected Network Elements (NE), composed in turn of several Managed Objects (MO). MO's act as peers providing to each other services for overall fault management. Consequently, each MO is equipped with its own fault management function. This involves self-monitoring for possible own internal sources of fault, as well as propagating, to clients of the considered MO, the effect of one of its servers getting disabled.

Because of this modularity, faults propagate throughout the management system: when a primary fault occurs in some MO, that MO emits an alarm to the supervisor, sends a message to its neigbours, and gets disabled. Its neighbouring MOs receive the message, recognize their inability to deliver their service, get disabled, emit alarms, and so on.

Figure 1 shows on the left hand side the SDH/SONET ring in operation in the Paris area (the locations indicated are subsurbs of Paris). A few ports and links are shown. The right diagram is a detailed view of the Montrouge node. The nested light to mid gray rectangles represent the different layers in the SDH hierarchy, with the largest one being the physical layer. The different boxes are the MOs, and the links across the different layers are the paths for upward/downward fault propagation. Each MO can be seen as an automaton

[3] This section has been prepared with the help of with Armen Aghasaryan Alcatel Research & Innovation, Next Generation Network and Service Management Project, Alcatel R&I, Route de Nozay, Marcoussis, 91461 France

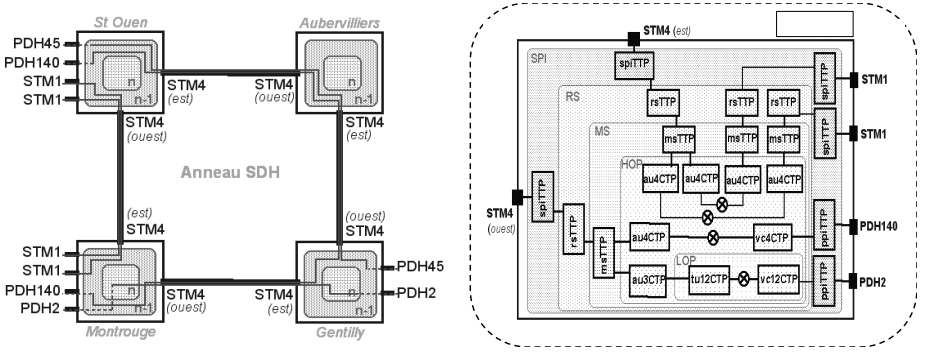

Fig. 1. The Paris area SDH/SONET ring (left), and a detail of the Montrouge node (right). The different levels of the SDH hierarchy are shown: SPI, RS, etc.

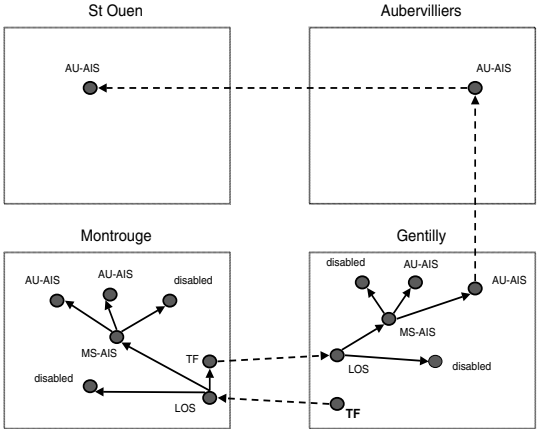

Fig. 2. A fault propagation scenario distributed across the four different sites. The dashed arrows indicate distant propagation. The cryptic names are SDH/SONET fault labels.

reacting to input events/messages, changing its state, and emitting events and alarms to its neighbours, both co-located and distant. Figure 2 shows a realistic example of a fault propagation scenario distributed across the four different sites.

To summarize, the different supervisors are distributed, and different MO's operate concurrently and asynchronously within each supervisor.

3 Informal Discussion of an Example

If all variables involved in the above scenarios possess a finite domain, we can represent these in an enumerative form. This suggests using safe Petri nets with a true concurrency semantics to formalize distributed fault diagnosis.

Presenting the Example, and the Problem. Our example is shown in Fig. 3–1st diagram, in the form of a labeled Petri net, with two interacting components, numbered 1 and 2. Component 2 uses the services of component 1, and therefore may fail to deliver its service when component 1 is faulty. The two components interact via their shared places 3 and 7, represented by the gray zone; note that this Petri net is safe, and that the two places 3 and 7 are complementary.

Component 1 has two private states: safe, represented by place 1, and faulty, represented by place 2. Upon entering its faulty state, component 1 emits an alarm β. The fault of component 1 is temporary, thus self-repair is possible and is represented by the label ρ. Component 2 has three private states, represented by places $4, 5, 6$. State 4 is safe, 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 failure of component 2 caused by a fault of component 1 is modeled by the shared place 3. The monitoring system of component 2 only detects that component 2 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 posterior to 4. 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 bears the same label ρ. Finally, place 7 guarantees that fault propagation, from component 1 to 2, is possible only when the latter is in safe state.

The initial marking consists of the three 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.

Three different setups can be considered for diagnosis, assuming that messages are not lost:

Setup $\mathbf{S_1}$: 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 such that α causes α', α is recorded before α'.

Setup $\mathbf{S_2}$: Each sensor records its local alarms in sequence, while respecting causality. The different sensors perform independently and asynchronously, and a single supervisor collects the records from the different sensors. Thus any interleaving of the records from different sensors is possible, and causalities among alarms from different sensors are lost.

Setup $\mathbf{S_3}$: The fault monitoring is distributed, with different supervisors cooperating asynchronously. Each supervisor is attached to a component, records its local alarms in sequence, and can exchange supervision messages with the other supervisors, asynchronously.

A Simple Solution? For setup $\mathbf{S_1}$, there is a simple solution. Call $\mathcal{A}$ the recorded alarm 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 $\mathcal{A}$), a new copy of the trial (a new Petri net) is instanciated 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. Setup $\mathbf{S}_2$ can be handled similarly, by exploring all interleavings of the two recorded alarm sequences. However, this direct approach does not represent efficiently the set of all solutions to the diagnosis problem.

In addition, this direct approach does not work for Setup $\mathbf{S}_3$. In this case, no supervisor knows the entire net and no global interleaving of the recorded alarm sequences is available. Maintaining a coherent set of causally related local diagnoses becomes a difficult problem for which no straightforward solution works. The approach we propose below addresses both the Setup $\mathbf{S}_3$ and the efficient representation of all solutions, for all setups.

An Efficient Data Structure to Represent All Runs. Figure 3 shows our running example. The Petri net $\mathcal{P}$ is repeated on the 2nd diagram: the labels α, β, ρ have been discarded, and transitions are i, ii, iii, iv, v, vi. Places constituting the initial marking are indicated by thick circles.

The mechanism of constructing a run of $\mathcal{P}$ in the form of a partial order is illustrated in the 2nd and 3rd diagrams. Initialize any run of $\mathcal{P}$ with the three conditions labeled by the initial marking $(1, 7, 4)$. Append to the pair $(1, 7)$ a copy of the transition $(1, 7) \to i \to (2, 3)$. Append to the new place labeled 2 a copy of the transition $(2) \to iii \to (1)$. Append, to the pair $(3, 4)$, a copy of the transition $(3, 4) \to iv \to (7, 5)$ (this is the step shown). We have constructed (the prefix of) a run of $\mathcal{P}$. Now, all runs can be constructed in this way. Different runs can share some prefix.

In the 4th diagram we show (prefixes of) all runs, by superimposing their shared parts. The gray part of this diagram is a copy of the run shown in the

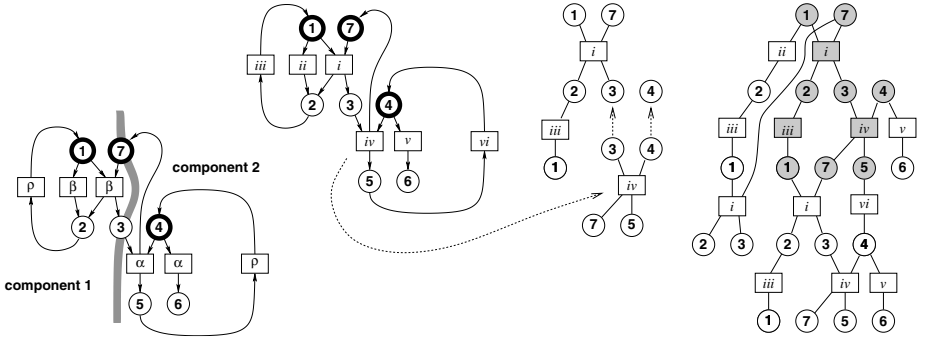

Fig. 3. Running example in the form of a Petri net (left), and representing its runs in a branching process. Petri nets are drawn by using directed arrows; on the other hand, since occurrence nets are acylic, we draw them using nondirected branches which have to be interpreted as implicitly directed toward bottom.

3rd diagram. The alternative run on the extreme left of this diagram (it involves successive transitions labeled ii, iii, i) shares only its initial places with the run in gray. On the other hand, replacing, in the gray run, the transition labeled iv by the one labeled v yields another run which shares with the gray one its transitions respectively labeled by i and by iii. This 4th diagram is a *branching process* of $\mathcal{P}$, we denote it by $\mathcal{U}_{\mathcal{P}}$; it is a net without cycle, in which the preset of any condition contains exactly one event. Nodes of $\mathcal{U}_{\mathcal{P}}$ are labeled by places/transitions of $\mathcal{P}$ in such a way that the two replicate each other, locally around transitions.

Terminology. When dealing with branching processes, to distinguish from the corresponding concepts in Petri nets, we shall from now on refer to *conditions/events* instead of places/transitions.

Asynchronous Diagnosis with a Single Sensor and Supervisor. Here we consider setup $\mathbf{S}_1$, and our discussion is supported by Fig. 4. The 1st diagram of this figure is the alarm sequence $\beta, \alpha, \rho, \rho, \beta, \alpha$ recorded at the unique sensor. It is represented by a cycle-free, linear Petri net, whose conditions are not labeled—conditions have no particular meaning, their only purpose is to indicate the ordering of alarms. Denote by $\mathcal{A}' = \beta \to \alpha \to \rho$ the shaded prefix of $\mathcal{A}$.

The 2nd diagram shows the net $\mathcal{U}_{\mathcal{A}' \times \mathcal{P}}$, obtained by unfolding the product $\mathcal{A}' \times \mathcal{P}$ using the procedure explained in the figure 3. The net $\mathcal{U}_{\mathcal{A}' \times \mathcal{P}}$ shows how successive transitions of $\mathcal{P}$ synchronize with transitions of $\mathcal{A}'$ having identical label, and therefore explain them.

Since we are not interested in the conditions originating from $\mathcal{A}'$, we remove them. The result is shown on the 3rd diagram. The dashed line labeled # ori-

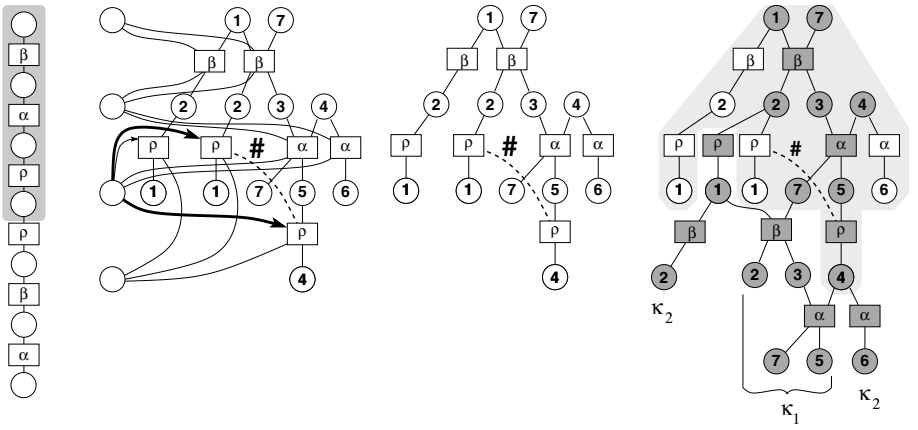

Fig. 4. Asynchronous diagnosis with a single sensor: showing an alarm sequence $\mathcal{A}$ (1st diagram), the explanation of the prefix $\mathcal{A}' = \beta \to \alpha \to \rho$ as the branching process $\mathcal{U}_{\mathcal{A}' \times \mathcal{P}}$ (2nd and 3rd diagrams), and its full explanation in the form of a net $\mathcal{U}_{\mathcal{P},\mathcal{A}}$ (4th diagram). In these diagrams, all branches are directed downwards unless otherwise explicitly stated.

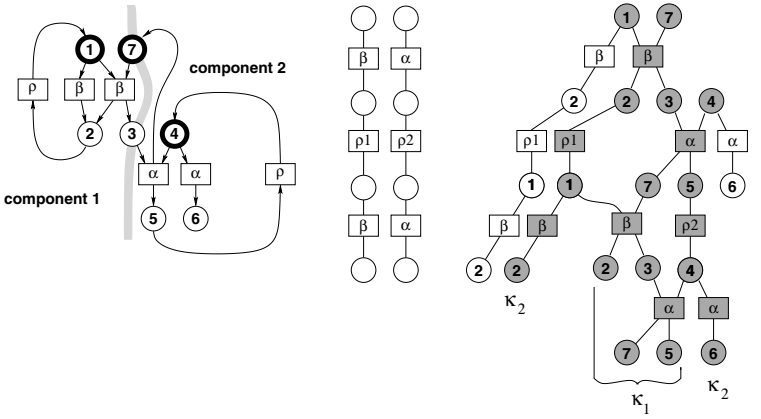

Fig. 5. Asynchronous diagnosis with two independent sensors: showing an alarm pattern $\mathcal{A}$ (middle) consisting of two concurrent alarm sequences, and its explanation in the form of a branching process $\mathcal{U}_{\mathcal{P},\mathcal{A}}$ (right).

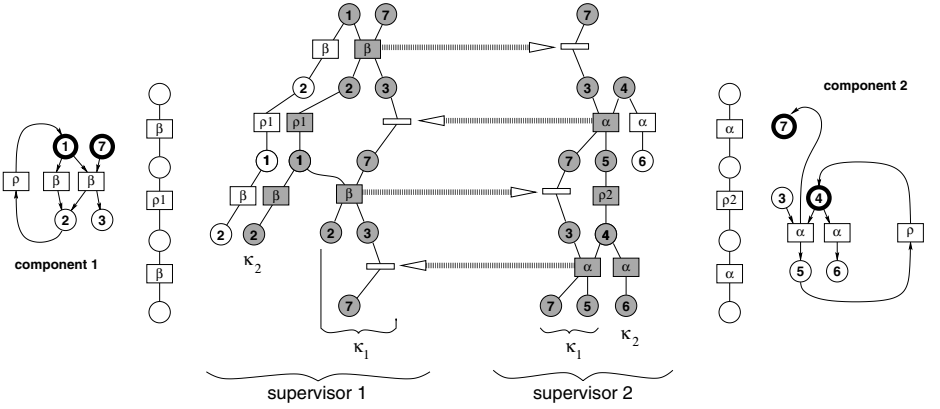

Fig. 6. Distributed diagnosis: constructing two coherent local views of the branching process $\mathcal{U}_{\mathcal{P},\mathcal{A}}$ of Fig. 5 by two supervisors cooperating asynchronously.

ginates from the corresponding conflict in $\mathcal{U}_{\mathcal{A}'\times\mathcal{P}}$ that is due to two different conditions explaining the same alarm ρ. Thus we need to remove, as possible explanations of the prefix, all runs of the 3rd diagram that contain the #-linked pair of events labeled ρ. All remaining runs are valid explanations of the subsequence β, α, ρ.

Finally, the net $\mathcal{U}_{\mathcal{P},\mathcal{A}}$ shown in the 4th diagram contains a prefix consisting of the nodes filled in dark gray. This prefix is the union of the two runs κ_1 and κ_2 of $\mathcal{P}$, that explain $\mathcal{A}$.

Asynchronous Diagnosis with Two Independent Sensors but a Single Supervisor. Focus on setup $\mathbf{S}_2$, in which alarms are recorded by two independent sensors, and then collected at a single supervisor for explanation. Figure 5 shows the same alarm history as in Fig. 4, except that it has been recorded by two independent sensors, respectively attached to each component. The supervisor knows the global model of the system, we recall it in the 1st diagram of Fig. 5.

The two "repair" actions are now distinguished since they are seen by different sensors, this is why we use different labels: ρ_1, ρ_2. This distinction reduces the ambiguity: in Fig. 5 we suppress the white filled path $(2) \to \rho \to (1)$ that occured in Fig. 4. On the other hand, alarms are recorded as two concurrent sequences, one for each sensor, call the whole an *alarm pattern.* Causalities between alarms from different components are lost. This leads to further ambiguity, as shown by the longer branch $(1) \to \beta \to (2) \to \rho_1 \to (1) \to \beta \to (2)$ in Fig. 5, compare with Fig. 4.

The overall result is shown in Fig. 5, and the valid explanations for the entire alarm pattern are the two configurations κ_1 and κ_2 filled in dark gray.

Distributed Diagnosis with Two Concurrent Sensors and Supervisors. Consider setup $\mathbf{S}_3$, in which alarms are recorded by two independent sensors, and processed by two local supervisors which can communicate asynchronously. Figure 6 shows two branching processes, respectively local to each supervisor. For completeness, we have shown the information available to each supervisor. It consists of the local model of the component considered, together with the locally recorded alarm pattern. The process constructed by supervisor 1 involves only events labeled by alarms collected by sensor 1, and places that are either local to component 1 (e.g., $1, 2$) or shared (e.g., $3, 7$); and similarly for the process constructed by supervisor 2.

The 3rd diagram of Fig. 5 can be recovered from Fig. 6 in the following way: glue events sitting at opposite extremities of each thick dashed arrow, identify adjacent conditions, and remove the thick dashed arrows. These dashed arrows indicate a communication between the two supervisors, let us detail the first one. The first event labeled by alarm β belongs to component 1, hence this explanation for β has to be found by supervisor 1. Supervisor 1 sends an abstraction of the path $(1, 7) \to \beta \to (2, 3)$ by removing the local conditions $1, 2$ and the label β since the latter do not concern supervisor 2. Thus supervisor 2 receives the path $(7) \to [] \to (3)$ to which it can append its local event $(3, 4) \to \alpha \to (7, 5)$; and so on.

Discussion. The cooperation between the two supervisors needs only asynchronous communication. Each supervisor can simply "emit and forget." Diagnosis can progress concurrently and asynchronously at each supervisor. For example, supervisor 1 can construct the branch $[1 \to \beta \to 2 \to \rho_1 \to 1 \to \beta \to 2]$ as soon as the corresponding local alarms are collected, without ever synchronizing with supervisor 2. This technique extends to distributed diagnosis with several

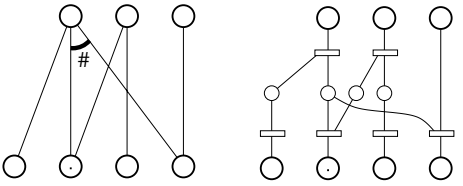

Fig. 7. Illustrating the problem of representating causality, conflict, and concurrency with an occurrence net. The 1st diagram represents a set of conditions together with a causality relation depicted by branches, and a conflict relation whose source is indicated by the symbol #. The 2nd diagram interpolates the 1st one as an occurrence net. (Arcs are assumed directed downwards.)

supervisors. But the algorithm for this general case is indeed very tricky, and its analysis is even more so. Thus, theoretical study is really necessary, and this is the main subject of the remainder of this paper.

As Fig. 4 and Fig. 6 indicate, we need projection and composition operations on branching processes. Focus on projections. Projecting away, from an occurrence net, a subset of conditions and events, can be performed by taking the restriction, to the subset of remaining conditions, of the two causality and conflict relations. An instance of resulting structure is shown in Fig. 7–left. A possible interpolation in the form of an occurrence net is shown in Fig. 7–right. Such an interpolation is not unique. In addition, the introduction of dummy conditions and events becomes problematic when combining projections and compositions.

We need a class of data structures equipped with causality and conflict relations, that is stable under projections. The class of event structures is a candidate. However, since diagnosis must be explained by sets of state histories, we need to handle conditions, not events. For this reason, we rather consider *condition structures,* as introduced in the next section.

4 Mathematical Framework: Nets, Unfoldings, and Condition Structures

4.1 Prerequisites on Petri nets and Their Unfoldings [6, 8, 26]

Petri nets. A *net* is a triple $\mathcal{N} = (P, T, \rightarrow)$, where P and T are disjoint sets of *places* and *transitions*, and $\rightarrow \subseteq (P \times T) \cup (T \times P)$ is the *flow relation.* Reflexive and irreflexive transitive closures of a relation are denoted by the superscripts $(.)^*$ and $(.)^+$, respectively. Define $\preceq = \rightarrow^*$ and $\prec = \rightarrow^+$. 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 $X^\bullet = \bigcup_{x \in X} x^\bullet$. A *homomorphism* from a net $\mathcal{N}$ to a net $\mathcal{N}'$ is a map $\varphi : P \cup T \mapsto P' \cup T'$ such that: (i) $\varphi(P) \subseteq P'$, $\varphi(T) \subseteq T'$, and (ii) for every transition t of $\mathcal{N}$, the restriction of φ to ${}^\bullet t$ is a bijection between ${}^\bullet t$ and ${}^\bullet \varphi(t)$, and the restriction of φ to $t^\bullet$ is a bijection between $t^\bullet$ and $\varphi(t)^\bullet$.

For $\mathcal{N}$ a net, a *marking* of $\mathcal{N}$ is a multiset M of places, i.e., a map $M : P \mapsto \{0, 1, 2, \ldots\}$. A *Petri net* is a pair $\mathcal{P} = (\mathcal{N}, M_0)$, where $\mathcal{N}$ 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 $M' = M - {}^\bullet t + t^\bullet$, denoted by $M[t\rangle M'$. 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.

For $\mathcal{N}_i = \{P_i, T_i, \rightarrow_i\}$, $i = 1, 2$, two nets such that $T_1 \cap T_2 = \emptyset$, their *parallel composition* is the net

$$\mathcal{N}_1 \,\|\, \mathcal{N}_2 =_{\text{def}} (P_1 \cup P_2, T_1 \cup T_2, \rightarrow_1 \cup \rightarrow_2).$$

Petri nets and occurrence nets inherit this notion. For Petri nets, we adopt the convention that the resulting initial marking is equal to $M_{1,0} \cup M_{2,0}$, the union of the two initial markings. We say that $\mathcal{N}_1 \,\|\, \mathcal{N}_2$ *has no distributed conflict* if:

$$\forall p \in P_1 \cap P_2, \exists i \in \{1, 2\} : p^\bullet \subseteq T_i. \tag{1}$$

Note that our example of Fig. 3 satisfies (1). This is a reasonable assumption in our context, since shared places aim at representing the propagation of faults between components. Having distributed conflict would have no meaning in this case. A study of this property in the context of the synthesis of distributed automata via Petri nets is available in [5].

For $\mathcal{N} = (P, T, \rightarrow)$ a net, a *labeling* is a map $\lambda : T \mapsto A$, where A is some finite alphabet. A net $\mathcal{N} = (P, T, \rightarrow, \lambda)$ equipped with a labeling λ is called a *labeled net.* For $\mathcal{N}_i = \{P_i, T_i, \rightarrow_i, \lambda_i\}$, $i = 1, 2$, two labeled nets, their product $\mathcal{N}_1 \times \mathcal{N}_2$ is the labeled net $(P, T, \rightarrow, \lambda)$ defined as follows: $P = P_1 \uplus P_2$, where $\uplus$ denotes the disjoint union, and:

$$T = \begin{cases} \{t =_{\text{def}} t_1 \in T_1 \mid \lambda_1(t_1) \in A_1 \setminus A_2\} & \text{(i)} \\ \cup\, \{t =_{\text{def}} (t_1, t_2) \in T_1 \times T_2 \mid \lambda_1(t_1) = \lambda_2(t_2)\} & \text{(ii)} \\ \cup\, \{t =_{\text{def}} t_2 \in T_2 \mid \lambda_2(t_2) \in A_2 \setminus A_1\}, & \text{(iii)} \end{cases}$$

$$p \rightarrow t \text{ iff } \begin{cases} p \in P_1 \text{ and } p \rightarrow_1 t_1 & \text{for case (i)} \\ \exists i = 1, 2 : p \in P_i \text{ and } p \rightarrow_i t_i & \text{for case (ii)} \\ p \in P_2 \text{ and } p \rightarrow_2 t_2 & \text{for case (iii)} \end{cases}$$

and $t \rightarrow p$ is defined symmetrically. 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.

The *language* $\mathcal{L}_\mathcal{P}$ of labeled Petri net $\mathcal{P}$ is the subset of A^* consisting of the words $\lambda(t_1), \lambda(t_2), \lambda(t_3), \ldots$, where $M_0[t_1\rangle M_1[t_2\rangle M_2[t_3\rangle M_3 \ldots$ ranges over the set of finite firing sequences of $\mathcal{P}$. Note that $\mathcal{L}_\mathcal{P}$ is prefix closed.

Occurrence Nets and Unfoldings. Two nodes x, x' of a net $\mathcal{N}$ 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 $\mathcal{O} = (B, E, \rightarrow)$, satisfying the following additional properties:

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

We will assume that the set of minimal nodes of $\mathcal{O}$ is contained in B, and we denote by $\min(B)$ or $\min(\mathcal{O})$ 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. We say that node x is *causally related* to node x' iff $x \prec x'$. Nodes x and x' are *concurrent,* written $x \perp\!\!\!\perp x'$, if neither $x \preceq x'$, nor $x \preceq x'$, nor $x\#x'$ hold. A *conflict set* is a set $\mathcal{A}$ of pairwise conflicting nodes, i.e. a clique of $\#$; a *co-set* is a set X of pairwise concurrent conditions. A maximal (for set inclusion) co-set is called a *cut.* A *configuration* is a sub-net κ of $\mathcal{O}$, which is *conflict-free* (no two nodes are in conflict), *causally closed* (if $x' \preceq x$ and $x \in \kappa$, then $x' \in \kappa$), and contains $\min(\mathcal{O})$. We take the convention that maximal nodes of configurations shall be conditions.

A *branching process* of Petri net $\mathcal{P}$ is a pair $\mathcal{B} = (\mathcal{O}, \varphi)$, where $\mathcal{O}$ is an occurrence net, and φ is a homomorphism from $\mathcal{O}$ to $\mathcal{P}$ regarded as nets, such that: (i) the restriction of φ to $\min(\mathcal{O})$ is a bijection between $\min(\mathcal{O})$ and M_0 (the set of initially marked places), and (ii) 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 $\min(\mathcal{O})$. 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 [9], 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 $\mathcal{U}_\mathcal{P}$. Maximal configurations of $\mathcal{U}_\mathcal{P}$ are called *runs* of $\mathcal{P}$.

4.2 Condition Structures

Occurrence nets give rise in a natural way to *(prime) event structures* [22]: a prime event structure is a triple $\mathcal{E} = (E, \preceq, \#)$, where $\preceq \subseteq E \times E$ is a partial order such that (i) for all $e \in E$, the set $\{e' \in E \mid e' \preceq e\}$ is finite, and (ii) $\# \subseteq E \times E$ is symmetric and irreflexive, and such that for all $e_1, e_2, e_3 \in E$, $e_1\#e_2$ and $e_2 \preceq e_3$ imply that $e_1\#e_3$. Obviously, "forgetting" the net interpretation of an occurrence net yields an event structure, and even restricting to the event set E does. This is the usual way of associating nets and event structures, and explains the name. Below, we will use event structures whose elements are interpreted as *conditions* in the sense of occurrence nets. To avoid confusion, we will speak of *condition structures*, even if the mathematical properties are invariant under this change of interpretation. Restricting an occurrence net to its set of conditions yields a condition structure.

Condition structures are denoted by $\mathcal{C} = (B, \preceq, \#)$. Denote by $\rightarrow$ the successor relation, i.e., the transitive reduction of the relation $\preceq$. For $b \in B$, we denote by ${}^\bullet b$ and $b^\bullet$ the preset and postset of b in $(B, \rightarrow)$, respectively. For $\mathcal{C}' \subseteq \mathcal{C}$ two condition structures, define the preset ${}^\bullet\mathcal{C}' = \bigcup_{b \in \mathcal{C}'} {}^\bullet b$; the postset $\mathcal{C}'^\bullet$ is defined similarly. The prefix relation on condition structures is denoted by $\sqsubseteq$. If $\mathcal{C}' \sqsubseteq \mathcal{C}$ are two condition structures, we write

$$\mathcal{C}' \rightarrow b \text{ iff } b \in \mathcal{C}'^\bullet \text{ but } b \notin \mathcal{C}', \tag{2}$$

and we say that b is an *extension* of $\mathcal{C}'$. Each condition structure $\mathcal{C} = (B, \preceq, \#)$ induces a *concurrency* relation defined by $b \perp\!\!\!\perp b'$ iff neither $b \preceq b'$ nor $b' \preceq b$ nor $b \# b'$ holds. Two conditions b, b' are called in *immediate conflict* iff $b \# b'$ and $\forall b''$ such that $b'' \prec b$, then $\neg[b'' \# b']$, and symmetrically.

For $\mathcal{C} = (B, \preceq, \#)$ a condition structure, a *labeling* is a map $\varphi : B \mapsto P$, where P is some finite alphabet [4]. We shall not distinguish labeled condition structures that are identical up to a bijection that preserves labels, causalities, and conflicts; such condition structures are called

$$\textit{equivalent,} \text{ denoted by the equality symbol } = \tag{3}$$

For $\mathcal{C} = (B, \preceq, \#, \varphi)$ and $\mathcal{C}' = (B', \preceq', \#', \varphi')$ two labeled condition structures,

$$\text{a } \textit{(partial) morphism } \psi : \mathcal{C} \mapsto \mathcal{C}'$$

is a surjective function $B \supseteq dom(\psi) \mapsto B'$ such that $\psi(\preceq) \supseteq \preceq'$ and $\psi(\#) \supseteq \#'$ (causalities and conflicts can be erased but not created), which is in addition label preserving, i.e., $\forall b \in dom(\psi)$, $\varphi'(\psi(b)) = \varphi(b)$. Note that $\psi(\preceq) \supseteq \preceq'$ is equivalent to $\forall b \in B : {}^\bullet\psi(b) \subseteq \psi({}^\bullet b)$. For $X \subset B$, define, for convenience:

$$\psi(X) =_{\text{def}} \psi(X \cap dom(\psi)), \text{ with the convention } \psi(\emptyset) = nil. \tag{4}$$

$\mathcal{C}$ and $\mathcal{C}'$ are *isomorphic*, written $\mathcal{C} \sim \mathcal{C}'$, if there exist two morphisms $\psi' : \mathcal{C} \mapsto \mathcal{C}'$ and $\psi'' : \mathcal{C}' \mapsto \mathcal{C}$. It is not true in general that $\mathcal{C} \sim \mathcal{C}'$ implies $\mathcal{C} = \mathcal{C}'$ in the sense of (3). However:

Lemma 1. *If $\mathcal{C}$ and $\mathcal{C}'$ are finite, then $\mathcal{C} \sim \mathcal{C}'$ implies $\mathcal{C} = \mathcal{C}'$.*

Be careful that $\mathcal{C} = \mathcal{C}'$ means that $\mathcal{C}$ and $\mathcal{C}'$ are *equivalent,* not "equal" in the naive sense—we will not formulate this warning any more. To be able to use lemma 1, we shall henceforth assume the following, where the *height* of a condition structure is the least upper bound of the set of all lengths of finite causal chains $b_0 \rightarrow b_1 \rightarrow \ldots \rightarrow b_k$:

[4] The reader may be surprised that we reuse the symbols P and φ for objects that are different from the set of places P of some Petri net and the homomorphism φ associated with its unfolding. This notational overloading is indeed intentional. We shall mainly use condition structures obtained by erasing events in unfoldings: restricting the homomorphism $\varphi : B \cup E \mapsto P \cup T$ to the set of conditions B yields a labeling $B \mapsto P$ in the above sense.

Assumption 1 *All condition structures we consider are of finite width, meaning that all prefixes of finite height are finite.*

In this paper, we will consider mainly labeled condition structures satisfying the following property, we call them *trimmed* condition structures:

$$\forall b, b' \in B \ : \ \left.\begin{array}{c} {}^\bullet b = {}^\bullet b' \\ \text{and } \varphi(b) = \varphi(b') \end{array}\right\} \Rightarrow b = b'. \tag{5}$$

Condition (5) is similar to the irreducibility hypothesis found in branching processes. It is important when considering the projection operation below, since projections may destroy irreducibility.

A *trimming procedure* can be applied to any labeled condition structure $\mathcal{C} = (B, \preceq, \#, \varphi)$ as follows: inductively by starting from $\min(\mathcal{C})$, identify all pairs (b, b') of conditions such that both ${}^\bullet b = {}^\bullet b'$ and $\varphi(b) = \varphi(b')$ hold in $\mathcal{C}$. This procedure yields a triple $(\bar{B}, \bar{\preceq}, \bar{\varphi})$, and it remains to define the trimmed conflict relation $\bar{\#}$, or, equivalently, the trimmed concurrency relation $\bar{\perp\!\!\!\perp}$. Define $\bar{b} \ \bar{\perp\!\!\!\perp} \ \bar{b}'$ iff $b \perp\!\!\!\perp b'$ holds for some pair (b, b') of conditions mapped to $(\bar{b}, \bar{b}')$ by the trimming procedure. This defines $\bar{\mathcal{C}} = (\bar{B}, \bar{\preceq}, \bar{\#}, \bar{\varphi})$.

It will be convenient to consider the following *canonical form* for a labeled trimmed condition structure. Its conditions have the special inductive form (X, p), where X is a co-set of $\mathcal{C}$ and $p \in P$. The causality relation $\preceq$ is simply encoded by the preset function ${}^\bullet(X, p) = X$, and the labeling map is $\varphi(X, p) = p$. Conditions with empty preset have the form (nil, p), i.e., the distinguished symbol *nil* is used for the minimal conditions of $\mathcal{C}$. The conflict relation is specified separately. Unless otherwise specified, trimmed condition structures will be assumed in canonical form.

For $\mathcal{C} = (B, \preceq, \#, \varphi)$ a trimmed labeled condition structure, and $Q \subset P$ a subset of its labels, the *projection of* $\mathcal{C}$ *on* Q, denoted by $\Pi_Q(\mathcal{C})$, is defined as follows. Take the restriction of $\mathcal{C}$ to $\varphi^{-1}(Q)$, we denote it by $\mathcal{C}_{\,|\,\varphi^{-1}(Q)}$. By this we mean that we restrict B as well as the two relations $\preceq$ and $\#$. Note that $\mathcal{C}_{\,|\,\varphi^{-1}(Q)}$ is not trimmed any more. Then, applying the trimming procedure to $\mathcal{C}_{\,|\,\varphi^{-1}(Q)}$ yields $\Pi_Q(\mathcal{C})$, which is trimmed and in canonical form. By abuse of notation, denote by $\Pi_Q(b)$ the image of $b \in \mathcal{C}_{\,|\,\varphi^{-1}(Q)}$ under this operation. The projection possesses the following universal property:

$$\forall \psi : \mathcal{C} \mapsto \mathcal{C}', \text{ and } \mathcal{C}' \text{ has label set } Q \implies \exists \psi' : \begin{array}{ccc} \mathcal{C} & \xrightarrow{\Pi_Q} & \Pi_Q(\mathcal{C}) \\ & {\scriptstyle\psi}\searrow & \downarrow{\scriptstyle\psi'} \\ & & \mathcal{C}' \end{array} \tag{6}$$

In (6), symbols ψ, ψ', Π_Q are morphisms, and the diagram commutes.

The *composition* of the two trimmed condition structures $\mathcal{C}_i = (B_i, \preceq_i, \#_i, \varphi_i)$, $i = 1, 2$, where labeling φ_i takes its values in alphabet P_i, is the condition structure

$$\begin{array}{c} \mathcal{C}_1 \wedge \mathcal{C}_2 = (B, \preceq, \#, \varphi), \text{ with two associated morphisms} \\ \psi_i : \mathcal{C}_1 \wedge \mathcal{C}_2 \mapsto \mathcal{C}_i, i = 1, 2, \end{array} \tag{7}$$

satisfying the following universal property:

$$\forall \psi'_1, \psi'_2 : \begin{array}{ccc} & \mathcal{C}' & \\ {\scriptstyle \psi'_1}\swarrow & & \searrow{\scriptstyle \psi'_2} \\ \mathcal{C}_1 & & \mathcal{C}_2 \end{array} \implies \exists \psi' : \begin{array}{ccccc} & & \mathcal{C}' & & \\ & {\scriptstyle \psi'_1}\swarrow & \downarrow{\scriptstyle \psi'} & \searrow{\scriptstyle \psi'_2} & \\ \mathcal{C}_1 & \xleftarrow[\psi_1]{} & \mathcal{C}_1 \wedge \mathcal{C}_2 & \xrightarrow[\psi_2]{} & \mathcal{C}_2 \end{array} \tag{8}$$

In (8), the ψ's denote morphisms, and the second diagram commutes. The composition is inductively defined as follows (we use the canonical form):

1. $\min(\mathcal{C}_1 \wedge \mathcal{C}_2) =_{\text{def}} \min(\mathcal{C}_1) \cup \min(\mathcal{C}_2)$, where we identify $(nil, p) \in \mathcal{C}_1$ and $(nil, p) \in \mathcal{C}_2$ for $p \in P_1 \cap P_2$. The causality relation and labeling map follow from the canonical form, and $\# =_{\text{def}} \#_1 \cup \#_2$. The canonical surjections $\psi_i : \min(\mathcal{C}_1 \wedge \mathcal{C}_2) \mapsto \min(\mathcal{C}_i), i = 1, 2$ are morphisms.
2. Assume $\mathcal{C}' =_{\text{def}} \mathcal{C}'_1 \wedge \mathcal{C}'_2$ together with the morphisms $\psi_i : \mathcal{C}' \mapsto \mathcal{C}'_i$ are defined, for $\mathcal{C}'_i$ a finite prefix of $\mathcal{C}_i$, and $i = 1, 2$. Then, using (4) we define, for all co-sets X of $\mathcal{C}'$:

$$\left.\begin{array}{r} X \subset dom(\psi_1), p \in P_1 \setminus P_2 \\ \mathcal{C}'_1 \to (\psi_1(X), p) \end{array}\right\} \Rightarrow \quad \mathcal{C}' \to (X, p) \qquad \text{(i)}$$

$$\left.\begin{array}{r} X \subset dom(\psi_1), p \in P_1 \cap P_2 \\ \mathcal{C}'_1 \to (\psi_1(X), p) \end{array}\right\} \Rightarrow \quad \mathcal{C}' \to (X, p) \qquad \text{(i')}$$

$$\left.\begin{array}{r} X \subset dom(\psi_2), p \in P_2 \setminus P_1 \\ \mathcal{C}'_2 \to (\psi_2(X), p) \end{array}\right\} \Rightarrow \quad \mathcal{C}' \to (X, p) \qquad \text{(ii)}$$

$$\left.\begin{array}{r} X \subset dom(\psi_2), p \in P_1 \cap P_2 \\ \mathcal{C}'_2 \to (\psi_2(X), p) \end{array}\right\} \Rightarrow \quad \mathcal{C}' \to (X, p) \qquad \text{(ii')}$$

$$\left.\begin{array}{r} p \in P_1 \cap P_2 \\ \mathcal{C}'_1 \to (\psi_1(X), p) \\ \mathcal{C}'_2 \to (\psi_2(X), p) \end{array}\right\} \Rightarrow \quad \mathcal{C}' \to (X, p) \qquad \text{(iii)}$$

Some comments are in order. The above five rules overlap: if rule (iii) applies, then we could have applied as well rule (i') with $Y = X \cap dom(\psi_1)$ in lieu of X, and the same for (ii'). Thus we equip rules (i–iii) with a set of priorities (a rule with priority 2 applies only if no rule with priority 1 is enabled):

$$\text{rules (i,ii,iii) have priority 1, rules (i',ii') have priority 2.} \tag{9}$$

For the five cases (i,i',ii,ii',iii), extend $\psi, i = 1, 2$ as follows:

$$\begin{array}{rl} \psi_1(X, p_1) =_{\text{def}} (\psi_1(X), p_1) & \text{(i, i')} \\ \psi_2(X, p_2) =_{\text{def}} (\psi_2(X), p_2) & \text{(ii, ii')} \\ \text{for } i = 1, 2 : \psi_i(X, p) =_{\text{def}} (\psi_i(X), p) & \text{(iii),} \end{array}$$

where convention (4) is used.

Using the above rules, define the triple $(B, \preceq, \varphi)$ as being the smallest[5] triple containing $\min(\mathcal{C}_1 \wedge \mathcal{C}_2)$, and such that no extension using rules (i,i',ii,ii',iii)

[5] for set inclusion applied to the sets of conditions.

applies. It remains to equip the triple $(B, \preceq, \varphi)$ with the proper conflict relation. The conflict relation $\#$ on $\mathcal{C}_1 \wedge \mathcal{C}_2$ is defined as follows:

$$\# \text{ is the smallest conflict relation containing } \psi_1^{-1}(\#_1) \cup \psi_2^{-1}(\#_2). \quad (10)$$

Comments. The composition of labeled event structures combines features from the product of labeled nets (its use of synchronizing labels), and from unfoldings (its inductive construction). Note that this composition is not strictly synchronizing, due to the special rules (i',ii'), which are essential in ensuring (8).

4.3 Condition Structures Obtained from Unfoldings

Consider a Petri net $\mathcal{P} = (P, T, \rightarrow, M_0)$ with its unfolding $\mathcal{U}_\mathcal{P}$. Let $\mathcal{C}_\mathcal{P} = (B, \preceq, \#, \varphi)$ be the condition structure obtained by erasing the events in $\mathcal{U}_\mathcal{P}$. Such condition structures are of finite width.

Lemma 2. *If $\mathcal{P}$ satisfies the following condition:*

$$\forall p \in P \ : \ t, t' \in {}^\bullet p \text{ and } t \neq t' \ \Rightarrow \ {}^\bullet t \neq {}^\bullet t'. \quad (11)$$

then $\mathcal{C}_\mathcal{P}$ is a trimmed condition structure, i.e., it satisfies (5). Unless otherwise stated, all Petri nets we shall consider satisfy this condition.

Condition (11) can always be enforced by inserting dummy places and transitions, as explained next. Assume that ${}^\bullet t \rightarrow t \rightarrow p$ and ${}^\bullet t' \rightarrow t' \rightarrow p$ with $t' \neq t$ but ${}^\bullet t' = {}^\bullet t$. Then, replace the path ${}^\bullet t \rightarrow t \rightarrow p$ by ${}^\bullet t \rightarrow t \rightarrow q_{t,p} \rightarrow s_{t,p} \rightarrow p$, where $q_{t,p}$ and $s_{t,p}$ are a fresh place and a fresh transition associated to the pair (t, p). Perform the same for the other path ${}^\bullet t' \rightarrow t' \rightarrow p$. This is a mild transformation, of low complexity cost, which does not modify reachability properties.

Important results about condition structures obtained from unfoldings are collected below. In this result, $\mathbf{1} =_{\text{def}} \emptyset$ denotes the empty condition structure, and $\mathcal{C}, \mathcal{C}_1, \mathcal{C}_2$ denote arbitrary condition structures with respective label sets P, P_1, P_2, and label set Q is arbitrary unless otherwise specified.

Theorem 1. *The following properties hold:*

$$\Pi_{P_1}(\mathcal{C}) \wedge \Pi_{P_2}(\mathcal{C}) = \Pi_{P_1 \cup P_2}(\mathcal{C}) \quad \text{(a0)}$$
$$\Pi_{P_1}(\Pi_{P_2}(\mathcal{C})) = \Pi_{P_1 \cap P_2}(\mathcal{C}) \quad \text{(a1)}$$
$$\Pi_P(\mathcal{C}) = \mathcal{C} \quad \text{(a2)}$$
$$\forall Q \supseteq P_1 \cap P_2 : \Pi_Q(\mathcal{C}_1 \wedge \mathcal{C}_2) = \Pi_Q(\mathcal{C}_1) \wedge \Pi_Q(\mathcal{C}_2) \quad \text{(a3)}$$
$$\mathcal{C} \wedge \mathbf{1} = \mathcal{C} \quad \text{(a4)}$$
$$\mathcal{C} \wedge \Pi_Q(\mathcal{C}) = \mathcal{C} \quad \text{(a5)}$$

4.4 Discussion

To our knowledge, compositional theories for unfoldings or event structures have received very little attention so far. The work of Esparza and Römer [10] investigates unfoldings for synchronous products of transition systems. The central

issue of this reference is the construction of finite complete prefixes. However, no product is considered, for the unfoldings themselves. To our knowledge, the work closest to ours is that of J-M. Couvreur et al. [7]–Sect. 4. Their motivations are different from ours, since they aim at constructing complete prefixes in a modular way, for Petri nets that have the form $\mathcal{P} = \mathcal{P}_1 \times \ldots \times \mathcal{P}_k$ (synchronous product). It is required that the considered Petri nets are not *reentrant,* a major limitation due to the technique used.

We are now ready to formally state our distributed diagnosis problem.

5 Distributed Diagnosis: Formal Problem Setting [3]

We are given the following labeled Petri nets:

$\mathcal{P} = (P, T, \rightarrow, M_0, \lambda)$ **:** the underlying "true" system. $\mathcal{P}$ is subject to faults, thus places from $\mathcal{P}$ are labelled by *faults,* taken from some finite alphabet (the non-faulty status is just one particular "fault"). The labeling map λ associates, to each transition of $\mathcal{P}$, a label belonging to some finite alphabet A of *alarm labels.* For its supervision, $\mathcal{P}$ produces so-called *alarm patterns,* i.e., sets of causally related alarms.

$\mathcal{Q} = (P^{\mathcal{Q}}, T^{\mathcal{Q}}, \rightarrow, M_0^{\mathcal{Q}}, \lambda^{\mathcal{Q}})$ **:** $\mathcal{Q}$ represents the faulty behaviour of $\mathcal{P}$, as observed via the sensor system. Thus we require that: (i) The labeling maps of $\mathcal{Q}$ and $\mathcal{P}$ take their values in the same alphabet A of alarm labels, and (ii) $\mathcal{L}_{\mathcal{Q}} \supseteq \mathcal{L}_{\mathcal{P}}$, i.e., the language of $\mathcal{Q}$ contains the language of $\mathcal{P}$. In general, however, $\mathcal{Q} \neq \mathcal{P}$. For example, if a single sensor is assumed, which collects alarms in sequence by preserving causalities (as assumed in [3]), then $\mathcal{Q}$ is the net which produces all linear extensions of runs of $\mathcal{P}$. In contrast, if several independent sensors are used, then the causalities between events collected by different sensors are lost. Configurations of $\mathcal{Q}$ are called *alarm patterns.*

Global Diagnosis. Consider the map: $\mathcal{A} \mapsto \mathcal{U}_{\mathcal{A}\times\mathcal{P}}$, where $\mathcal{A}$ ranges over the set of all finite alarm patterns. This map filters out, during the construction of the unfolding $\mathcal{U}_{\mathcal{P}}$, those configurations which are not compatible with the observed alarm pattern $\mathcal{A}$. Thanks to Lemma 2, we can replace the unfolding $\mathcal{U}_{\mathcal{A}\times\mathcal{P}}$ by the corresponding condition structure $\mathcal{C}_{\mathcal{A}\times\mathcal{P}}$. Then, we can project away, from $\mathcal{C}_{\mathcal{A}\times\mathcal{P}}$, the conditions labeled by places from $\mathcal{A}$ (all this is detailed in [3]–Theorem 1). Thus we can state:

Definition 1. *Global diagnosis is represented by the following map:*

$$\mathcal{A} \longmapsto \Pi_P(\mathcal{C}_{\mathcal{A}\times\mathcal{P}}), \tag{12}$$

where $\mathcal{A}$ ranges over the set of all finite configurations of $\mathcal{Q}$.

Modular Diagnosis. Assume that Petri net $\mathcal{P}$ decomposes as $\mathcal{P} = \|_{i\in I}\mathcal{P}_i$. The different subsystems $\mathcal{P}_i$ interact via some shared places, and their sets of

transitions are pairwise disjoint. In particular, the alphabet A of alarm labels decomposes as $A = \bigcup_{i\in I} A_i$, where the A_i are pairwise disjoint. Next, we assume that each subsystem $\mathcal{P}_i$ possesses its own local sets of sensors, and the local sensor subsystems are independent, i.e., do not interact. Thus $\mathcal{Q}$ also decomposes as $\mathcal{Q} = \|_{i\in I}\mathcal{Q}_i$, and the $\mathcal{Q}_i$ possess pairwise disjoint sets of places. Consequently, in (12), $\mathcal{A}$ decomposes as $\mathcal{A} = \|_{i\in I}\mathcal{A}_i$, where the $\mathcal{A}_i$, the locally recorded alarm patterns, possess pairwise disjoint sets of places too.

Definition 2. *Modular diagnosis is represented by the following map:*

$$\mathcal{A} \;\longmapsto\; [\Pi_{P_i}(\mathcal{C}_{\mathcal{A}\times\mathcal{P}})]_{i\in I}\,, \tag{13}$$

where $\mathcal{A}$ ranges over the set of all finite prefixes of runs of $\mathcal{Q}$.

Note that, thanks to Theorem 1, we know that $\Pi_P(\mathcal{C}_{\mathcal{A}\times\mathcal{P}}) = \bigwedge_{i\in I}[\Pi_{P_i}(\mathcal{C}_{\mathcal{A}\times\mathcal{P}})]$, i.e., fusing the local diagnoses yields global diagnosis. However, we need to compute modular diagnosis without computing global diagnosis. On the other hand, the reader should notice that, in general, $\Pi_{P_i}(\mathcal{C}_{\mathcal{A}\times\mathcal{P}}) \neq \mathcal{C}_{\mathcal{A}_i\times\mathcal{P}_i}$, expressing the fact that the different supervisors must cooperate at establishing a coherent modular diagnosis.

6 Distributed Orchestration of Modular Diagnosis [12, 13]

This section is essential. It provides a framework for the high-level orchestration of the distributed computation of modular diagnosis. We first link our problem with the seemingly different areas of distributed constraint solving and distributed optimization.

6.1 A Link with Distributed Constraint Solving

Consider Theorem 1, and re-interpret, for a while, the involved objects differently. Suppose that our above generic label set P is a set of variables, thus $p \in P$ denotes a variable. Then, suppose that all considered variables possess a finite domain, and that $\mathcal{C}$ generically denotes a *constraint* on the tuple P of variables, i.e., $\mathcal{C}$ is a subset of all possible values for this tuple. For $Q \subset P$, re-interpret $\Pi_Q(\mathcal{C})$ as the projection of $\mathcal{C}$ onto Q. Then, re-interpret $\wedge$ as the conjunction of constraints. Finally, $\mathbf{1}$ is the trivial constraint, having empty set of associated variables. It is easily seen that, whith this re-interpretation, properties (a0–a5) are satisfied.

Modular constraint solving consists in computing $\Pi_{P_i}(\bigwedge_{j\in I}\mathcal{C}_j)$ without computing the global solution $\bigwedge_{j\in I}\mathcal{C}_j$. Distributed constraint solving consists in computing $\Pi_{P_i}(\bigwedge_{j\in I}\mathcal{C}_j)$ in a distributed way. Thus distributed constraint solving is a simpler problem, which is representative of our distributed diagnosis problem, when seen at the proper abstract level.

A Chaotic Algorithm for Distributed Constraint Solving. We build a graph on the index set I as follows: draw a branch (i, j) iff $P_i \cap P_j \neq \emptyset$, i.e., $\mathcal{C}_i$ and $\mathcal{C}_j$ interact directly. Denote by $\mathcal{G}_I$ the resulting *interaction graph.* For $i \in I$, denote by $\mathrm{N}(i)$ the neighbourhood of i, composed of the set of j's such that $(i, j) \in \mathcal{G}_I$. Note that $\mathrm{N}(i)$ contains i. The algorithm assumes the existence of an "initial guess", i.e., a set of $\mathcal{C}_i, i \in I$ such that

$$\mathcal{C} = \bigwedge_{i \in I} \mathcal{C}_i, \tag{14}$$

and it aims at computing $\Pi_{P_i}(\mathcal{C})$, for $i \in I$, in a chaotic way. In the following algorithm, each site i maintains and updates, for each neighbour j, a message $\mathcal{M}_{i,j}$ toward j. Thus there are two messages per edge (i, j) of $\mathcal{G}_I$, one in each direction:

Algorithm 1

1. *Initialization:* for each edge $(i, j) \in \mathcal{G}_I$:

$$\mathcal{M}_{i,j} = \Pi_{P_i \cap P_j}(\mathbf{1}). \tag{15}$$

2. *Chaotic iteration:* until stabilization, select an edge $(i, j) \in \mathcal{G}_I$, and update:

$$\mathcal{M}_{i,j} := \prod_{P_i \cap P_j} \left(\mathcal{C}_i \wedge \left[\bigwedge_{k \in \mathrm{N}(i) \setminus j} \mathcal{M}_{k,i} \right] \right). \tag{16}$$

3. *Termination:* for each $i \in I$, set:

$$\mathcal{C}_i^\star = \mathcal{C}_i \wedge \left[\bigwedge_{k \in \mathrm{N}(i)} \mathcal{M}_{k,i} \right]. \tag{17}$$

The following result belongs to the folklore of smoothing theory in statistics and control. It was recently revitalised in the area of so-called "soft" coding theory. In both cases, the result is stated in the context of distributed constrained optimization. In its present form, it has been proved in [13] and uses only the abstract setting of Sect. 6.1 with properties (a0–a5):

Theorem 2 ([13]). *Assume that $\mathcal{G}_I$ is a tree. Then Algorithm 1 converges in finitely many iterations, and $\mathcal{C}_i^\star = \Pi_{P_i}(\mathcal{C})$.*

An informal argument of why the result is true is illustrated in Fig. 8. Message passing, from the leaves to the thick node, is depicted by directed arrows. Thanks to (16), each directed arrow cumulates the effect, on its sink node, of the constraints associated with its set of ancestor nodes, where "ancestor" refers to the ordering defined by the directed arrows. Now, we provide some elementary calculation to illustrate this mechanism. Apply Algorithm 1 with the particular policy shown in Fig. 8, for selecting the successive branches of $\mathcal{G}_I$. Referring to this figure, select concurrently the branches $(9, 5)$ and $(8, 5)$, and then select the

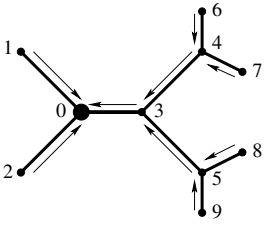

Fig. 8. Illustrating why Algorithm 1 performs well when $\mathcal{G}_I$ is a tree. Messages are generated first at the leaves. They meet at some arbitrarily selected node—the "center", here depicted in thick. Then they travel backward to the leaves.

branch $(5, 3)$. Then, successive applications of formula (16) yield:

$$\begin{array}{rl} \mathcal{M}_{(9,5)} & = \Pi_{P_9 \cap P_5}\Big(\mathcal{C}_9 \wedge \Big[\bigwedge_{k \in \mathrm{N}(9) \setminus 5} \mathcal{M}_{(k,i)}\Big]\Big) \\ & = \Pi_{P_9 \cap P_5}(\mathcal{C}_9) \text{ (since } \mathrm{N}(9) \setminus 5 = \emptyset) \\ \mathcal{M}_{(8,5)} & = \Pi_{P_8 \cap P_5}(\mathcal{C}_8) \\ \mathcal{M}_{(5,3)} & = \Pi_{P_5 \cap P_3}(\mathcal{C}_5 \wedge \mathcal{M}_{(9,5)} \wedge \mathcal{M}_{(8,5)}) \\ & = \Pi_{P_5 \cap P_3}(\mathcal{C}_5 \wedge \mathcal{C}_9 \wedge \mathcal{C}_8) \end{array} \tag{18}$$

The calculations can be further continued. They clearly yield the theorem for the center node $i = 0$, for the particular case where the branches are selected according to the policy shown in Fig. 8. A back-propagation to the leaves yields the result for all nodes. In this explanation, the messages were partially ordered, from the leaves to the thick node, and then vice-versa. Due to the monotonic nature of the algorithm, a chaotic and concurrent emission of the messages yields the same result, possibly at the price of exchanging a larger number of messages. Here is a formal proof borrowed from [13], we omit details.

Proof. Write $\mathcal{C} \leq \mathcal{C}'$ iff $\mathcal{C} = \mathcal{C}' \wedge \mathcal{C}''$ for some $\mathcal{C}''$. Using this notation, note that (16) is monotonic in the following sense: if $\forall k : \mathcal{M}_{k,i} \leq \mathcal{M}'_{k,i}$ in the right hand side of (16), then $\mathcal{M}_{i,j} \leq \mathcal{M}'_{i,j}$ in the left hand side of (16). Next, mark $\mathcal{M}_{i,j}$ in formula (16) with a running subset $J_{i,j} \subseteq I$, initialized with $J_{i,j} = \emptyset$:

$$\begin{array}{rl} \mathcal{M}_{i,j} & := \Pi_{P_i \cap P_j}\Big(\mathcal{C}_i \wedge \Big[\bigwedge_{k \in \mathrm{N}(i) \setminus j} \mathcal{M}_{k,i}\Big]\Big) \\ J_{i,j} & := J_{i,j} \cup \{i\} \cup \Big[\bigcup_{k \in \mathrm{N}(i) \setminus j} J_{k,i}\Big] \end{array} \tag{19}$$

Then, by using properties (a0–a5) and the monotonicity of (16), we get (left as an exercise to the reader):

$$\mathcal{M}_{i,j} = \Pi_{P_i \cap P_j}\Big(\bigwedge_{k \in J_{i,j}} \mathcal{C}_k\Big)\,. \tag{20}$$

Hint: compare with (18). Verify that the assumption that $\mathcal{G}_I$ is a tree is used for applying repeatedly axiom (a3): this assumption guarantees that $(\bigwedge_{k \in J_{i,j}} \mathcal{C}_k)$ and $\mathcal{C}_j$ interact only via $P_i \cap P_j$. From (20) the theorem follows easily. ◇

6.2 A Link with Distributed Constrained Optimization

As announced before, Theorem 2 generalizes to distributed constrained optimization. For this case, $\mathcal{C}$ becomes a pair $\mathcal{C} =$(constraint, cost)$=_{\text{def}} (\mathbf{C}, \mathbf{J})$. Constraints are as before, and costs have the following additive form:

$$\mathbf{J}(x) = \sum_{p \in P} \mathbf{j}_p(x_p), \tag{21}$$

where $x_p \in dom(p)$, the domain of variable p, $x = (x_p)_{p \in P}$, and the local costs $\mathbf{j}_p$ are real-valued, nonnegative cost functions. We require that $\mathbf{J}$ be normalized:

$$\sum_{x \models \mathbf{C}} \exp(\mathbf{J}(x)) = 1, \tag{22}$$

where $x \models \mathbf{C}$ means that x satisfies constraint $\mathbf{C}$, and exp(.) denotes the exponential. In the following, the notation **Cst** will denote generically a normalization factor whose role is to ensure condition (22). Then, define:

$$\Pi_Q(\mathbf{J})(x_Q) =_{\text{def}} \frac{1}{\mathbf{Cst}} \max_{x : \Pi_Q(x) = x_Q} \mathbf{J}(x), \tag{23}$$

In (23), $\Pi_Q(x)$ denotes the projection of x on Q. Then, define $\Pi_Q(\mathbf{C})$ to be the projection of constraint $\mathbf{C}$ on Q, i.e., the elimination from $\mathbf{C}$, by existential quantification, of the variables not belonging to Q. Finally, define $\Pi_Q(\mathcal{C}) =_{\text{def}} (\Pi_Q(\mathbf{C}), \Pi_Q(\mathbf{J}))$. Next, we define the composition $\wedge$. To this end, take for $\mathbf{C}_1 \wedge \mathbf{C}_2$ the conjunction of the considered constraints. And define:

$$(\mathbf{J}_1 \wedge \mathbf{J}_2)(x) =_{\text{def}} \frac{1}{\mathbf{Cst}} \left(\mathbf{J}_1(\Pi_{P_1}(x)) + \mathbf{J}_2(\Pi_{P_2}(x))\right). \tag{24}$$

It is easily checked that the properties (a0–a5) are still satisfied. Thus, Algorithm 1 solves, in a distributed way, the following problem:

$$\max_{x \models \mathbf{C}} \mathbf{J}(x), \tag{25}$$

for the case in which $\mathcal{C} =_{\text{def}} (\mathbf{C}, \mathbf{J})$ decomposes as $\mathcal{C} = \bigwedge_{i \in I} \mathcal{C}_i$.

Problem (25) can also be interpreted as maximum likelihood constraint solving, to resolve nondeterminism. In this case, the cost function $\mathbf{J}$ is interpreted as the logarithm of the likelihood (loglikelihood)—whence the normalization constraint (22). Then, the additive decomposition (21) for the loglikelihood means that the different variables are considered independent with respect to their prior distribution (i.e., when ignoring the constraints). In fact, the so defined systems $\mathcal{C}$ are Markov random fields, for which Algorithm 1 provides distributed maximum likelihood estimation. This viewpoint is closely related to belief propagation in belief networks [19, 20, 23].

6.3 Application to Off-Line Distributed Diagnosis

As said above, our framework of Sects. 4 and 5 satisfies properties (a0–a5). We can therefore apply algorithm 1 to perform off-line diagnosis, i.e., compute $\Pi_{P_i}(\mathcal{C}_{\mathcal{A}\times\mathcal{P}})$, cf. (13), for $\mathcal{A}$ being fixed. Now, we need an initial guess $\mathcal{C}_i$ satisfying (14), i.e., in our case: $\mathcal{C}_{\mathcal{A}\times\mathcal{P}} = \bigwedge_{i\in I}\mathcal{C}_i$. As we said, $\mathcal{C}_{\mathcal{A}_i\times\mathcal{P}_i}$ is not a suitable initial guess, since $\mathcal{C}_{\mathcal{A}\times\mathcal{P}} \neq \bigwedge_{i\in I}\mathcal{C}_{\mathcal{A}_i\times\mathcal{P}_i}$. Thus we need to find another one. Of course, we want this initial guess to be "cheap to compute", meaning at least that its computation is purely local and does not involve any cooperation between supervisors.

This is by no means a trivial task in general. However, in our running example, we can simply complement $\mathcal{P}_1$ by a new path (3) $\to$ [] $\to$ (7), and $\mathcal{P}_2$ by a path (7) $\to$ [] $\to$ (3), and the reader can check that (14) is satisfied by taking $\mathcal{C}_i =_{\text{def}} \mathcal{C}_{\mathcal{A}_i\times\bar{\mathcal{P}}_i}$, where $\bar{\mathcal{P}}_i, i = 1, 2$ denote the above introduced completions. In fact, this trick works for pairs of nets having a simple interaction involving only one pair of complementary places. It is not clear how to generalize this to more complex cases—fortunately, this difficulty disappears for the on-line algorithm, which is our very objective.

Anyway, having a correct initial guess at hand, Algorithm 1 applies, and yields the desired high-level orchestration for off-line distributed diagnosis. Each primitive operation of this orchestration is either a projection or a composition. For both, we have given the detailed definition above. All primitives are local to each site, i.e., involve only its private labels.

Again, since only properties (a0–a5) are required by Algorithm 1, we can also address the maximum likelihood extension discussed before (see [4, 17] for issues of randomizing Petri nets with a full concurrency semantics). This is the problem of maximum likelihood diagnosis that our prototype software solved, in the context described in Sect. 2.

6.4 An On-Line Variant of the Abstract Setting

Handling on-line diagnosis amounts to extending the results of Sect. 6.1 to "time-varying" structures in a certain sense [14]. We shall now complement the set of abstract properties (a0–a5) to prepare for the on-line case. Equip the set of condition structures with the following partial order:

$$\mathcal{C}' \sqsubseteq\!\!\!\sqsubseteq \mathcal{C} \text{ iff } \mathcal{C} \text{ is a prefix of } \mathcal{C}', \tag{26}$$

please note the inversion! To emphasize the analogy with constraint solving, $\mathcal{C}' \sqsubseteq\!\!\!\sqsubseteq \mathcal{C}$ reads: $\mathcal{C}'$ *refines* $\mathcal{C}$. Note that

$$\mathcal{C}' \sqsubseteq\!\!\!\sqsubseteq \mathcal{C} \text{ holds in particular if} : \mathcal{C}' = \mathcal{C}_{\mathcal{A}'\times\mathcal{P}}, \mathcal{C} = \mathcal{C}_{\mathcal{A}\times\mathcal{P}}, \mathcal{A} \sqsubseteq \mathcal{A}', \tag{27}$$

this is the situation encountered in incremental diagnosis. The following result complements Theorem 1:

Theorem 3. *The following properties hold:*

$$\begin{aligned} \mathcal{C} &\sqsubseteq \mathbf{1} && \text{(a6)}\\ \mathcal{C}_1 \sqsubseteq \mathcal{C}_2 \Rightarrow \quad \mathcal{C}_1 \wedge \mathcal{C}_3 &\sqsubseteq \mathcal{C}_2 \wedge \mathcal{C}_3 && \text{(a7)}\\ \mathcal{C}' \sqsubseteq \mathcal{C} \Rightarrow \forall Q : \Pi_Q(\mathcal{C}') &\sqsubseteq \Pi_Q(\mathcal{C}) && \text{(a8)} \end{aligned}$$

Distributed Constraint Solving with Monotonically Varying Constraints. Consider again our re-interpretation as distributed constraint solving. Now, instead of constraint $\mathcal{C}_i$ being given once and for all, we are given a *set of constraints* $\mathbf{C}_i$ *ordered by* $\sqsubseteq$. More precisely, for $\mathcal{C}_i, \mathcal{C}_i' \in \mathbf{C}_i$, write $\mathcal{C}_i \sqsubseteq \mathcal{C}_i'$ iff $\mathcal{C}_i \Rightarrow \mathcal{C}_i'$, i.e., the former refines the latter. We assume that $\mathbf{C}_i$ is a lattice, i.e., that the supremum of two constraints exists in $\mathbf{C}_i$. Since all domains are finite, then $\mathcal{C}_i^\infty = \lim_{\sqsubseteq}(\mathbf{C}_i)$ is well defined. Algorithm 1 is then modified as follows, [14]:

Algorithm 2

1. *Initialization:* for each edge $(i,j) \in \mathcal{G}_I$:
$$\mathcal{M}_{i,j} = \Pi_{P_i \cap P_j}(\mathbf{1}). \tag{28}$$
2. *Chaotic nonterminating iteration:* Choose nondeterministically, in the following steps:
CASE 1: select a node $i \in I$ and update:
$$\text{read } \mathcal{C}_i^{\mathsf{new}} \sqsubseteq \mathcal{C}_i^{\mathsf{cur}}, \text{ and update } \mathcal{C}_i^{\mathsf{cur}} := \mathcal{C}_i^{\mathsf{new}}. \tag{29}$$
CASE 2: select an edge $(i,j) \in \mathcal{G}_I$, and update:
$$\mathcal{M}_{i,j} := \Pi_{P_i \cap P_j}(\mathcal{C}_i^{\mathsf{cur}} \wedge [\textstyle\bigwedge_{k \in \mathrm{N}(i) \setminus j} \mathcal{M}_{k,i}]). . \tag{30}$$
CASE 3: Update subsystems: select $i \in I$, and set:
$$\mathcal{C}_i^\star = \mathcal{C}_i^{\mathsf{cur}} \wedge [\textstyle\bigwedge_{k \in \mathrm{N}(i)} \mathcal{M}_{k,i}]. \tag{31}$$

In step 2, $\mathcal{C}_i^{\mathsf{cur}}$ denotes the $\mathsf{current}$ estimated value for $\mathcal{C}_i$, whereas $\mathcal{C}_i^{\mathsf{new}}$ denotes the new, refined, version. Algorithm 2 is *fairly* executed if it is applied in such a way that every node i of CASE 1 and CASE 3, and every edge (i,j) of CASE 2 is selected infinitely many times.

Theorem 4 ([14]). *Assume that $\mathcal{G}_I$ is a tree, and Algorithm 2 is fairly executed. Then, for any given $\mathcal{C} = \bigwedge_{i \in I} \mathcal{C}_i$, where $\mathcal{C}_i \in \mathbf{C}_i$, after sufficiently many iterations, one has $\forall i \in I : \mathcal{C}_i^\star \sqsubseteq \Pi_{P_i}(\mathcal{C})$.*

Theorem 4 expresses that, modulo a fairness assumption, Algorithm 2 refines, with some delay, the solution $\Pi_{P_i}(\mathcal{C})$ of any given intermediate problem $\mathcal{C}$.

Proof. It refines the proof of Theorem 2. The monotonicity argument applies here with the special order $\sqsubseteq$. Due to our fairness assumption, after sufficiently many iterations, each node i has updated its $\mathcal{C}_i^{\mathsf{cur}}$ in such a way that $\mathcal{C}_i^{\mathsf{cur}} \sqsubseteq \mathcal{C}_i$. Select such a status of Algorithm 2, and then start marking the recursion (30) as in (19). Applying the same reasoning as for the proof of Theorem 2 yields that $\mathcal{M}_{i,j} \sqsubseteq \Pi_{P_i \cap P_j}(\bigwedge_{k \in J_{i,j}} \mathcal{C}_k)$, from which Theorem 4 follows. ◇

How Algorithms 1 and 2 behave when $\mathcal{G}_I$ possesses cycles is discussed in [1, 15].

Application to On-Line Distributed Diagnosis. Here we consider the case in which a possibly infinite alarm pattern is observed incrementally: $\mathcal{A}^\infty = \|_{i\in I}\, \mathcal{A}_i^\infty$, meaning that sensor i receives only finite prefixes $\mathcal{A}_i \sqsubseteq \mathcal{A}_i^\infty$, partially ordered by inclusion. Now, by (27):

$$\forall i \in I : \mathcal{A}_i' \sqsubseteq \mathcal{A}_i \Rightarrow \mathcal{C}_i = \mathcal{C}_{\mathcal{A}_i \times \mathcal{P}_i} \;\sqsubseteq\!\!\sqsubseteq\; \mathcal{C}_{\mathcal{A}_i' \times \mathcal{P}_i} = \mathcal{C}_i'. \tag{32}$$

Thus on-line distributed diagnosis amounts to generalizing Algorithm 1 to the case in which subsystems $\mathcal{C}_i$ are updated on-line while the chaotic algorithm is running. Theorem 4 expresses that, if applied in a fair manner, then Algorithm 2 explains any finite alarm pattern after sufficiently many iterations.

For the off-line case, we mentioned that obtaining an "initial guess" for the $\mathcal{C}_i$ was a difficult issue. Now, since Algorithm 2 progresses incrementally, the issue is to compute the increment from $\mathcal{C}_i^{\mathsf{cur}}$ to $\mathcal{C}_i^{\mathsf{new}}$ in a "cheap" way. As detailed in [1], this can be performed locally, provided that the increment from $\mathcal{A}_i^{\mathsf{cur}}$ to $\mathcal{A}_i^{\mathsf{new}}$ is small enough.

Back to Our Running Example. Here we only relate the different steps of Algorithm 2 to the Fig. 6. Initialization is performed by starting from empty unfoldings on both supervisors. CASE 1 of step (a) consists, e.g., for supervisor 1, in recording the first alarm β ($\mathcal{A}_i^{\mathsf{cur}} = \emptyset$ and $\mathcal{A}_i^{\mathsf{new}} = \{\beta\}$), and then explaining β by the net $(1) \to \beta \to (2) \cup (1,7) \to \beta \to (2,3)$. CASE 2 of step (a) consists, e.g., for supervisor 1, in computing the abstraction of this net, for use by supervisor 2, this is shown by the first thick dashed arrow. Step (b), e.g., consists, for supervisor 2, in receiving the above abstraction and using it to append $(3,4) \to \alpha \to (7,5)$ as an additional explanation for its first alarm α; another explanation is the purely local one $(4) \to \alpha \to (6)$, which does not require the cooperation of supervisor 1.

7 Conclusion

For the context of fault management in SDH/SONET telecommunications networks, a prototype software implementing the method was developed in our laboratory, using Java threads to emulate concurrency. This software was subsequently deployed at Alcatel on a truly distributed experimental management platform. No modification was necessary to perform this deployment.

To ensure that the deployed application be autonomous in terms of synchronization and control, we have relied on techniques from true concurrency. The overall distributed orchestration of the application also required techniques originating from totally different areas related to statistics and information theory, namely belief propagation and distributed algorithms on graphical models. Only by blending those two orthogonal domains was it possible to solve our problem, and our work is a contribution in both domains.

Regarding concurrency theory, we have introduced a new compositional theory of modular event (or condition) structures. These objects form a category

equipped with its morphisms, with a projection and two composition operations; it provides the adequate framework to support the distributed construction of unfoldings or event structures. It opens the way to using unfoldings or event structures as core data structures for distributed and asynchronous applications.

Regarding belief propagation, the work reported here presents an axiomatic form not known before. Also, time-varying extensions are proposed. This abstract framework allowed us to address distributed diagnosis, both off-line and on-line, and to derive types of algorithms not envisioned before in the field of graphical algorithms.

The application area which drives our research raises a number of additional issues for further investigation. Getting the model (the net $\mathcal{P}$) is the major one: building the model manually is simply not acceptable. We are developing appropriate software and models for a small number of generic management objects. These have to be instanciated on the fly at network discovery, by the management platform. This is a research topic in itself. From the theoretical point of view, the biggest challenge is to extend our techniques to dynamically changing systems. This is the subject of future research. Various robustness issues need to be considered: messages or alarms can be lost, the model can be approximate, etc. Probabilistic aspects are also of interest, to resolve nondeterminism by performing maximum likelihood diagnosis. The papers [4, 17] propose two possible mathematical frameworks for this, and a third one is in preparation.

References

1. extended version, available as IRISA report No 1540 http://www.irisa.fr/bibli/publi/pi/2003/1540/1540.html
2. 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
3. A. Benveniste, E. Fabre, C. Jard, and S. Haar. Diagnosis of asynchronous discrete event systems, a net unfolding approach. *IEEE Trans. on Automatic Control,* 48(5), May 2003. Preliminary version available from http://www.irisa.fr/sigma2/benveniste/pub/IEEE_TAC_AsDiag_2003.html
4. A. Benveniste, S. Haar, and E. Fabre. Markov Nets: probabilistic Models for Distributed and Concurrent Systems. INRIA *Report* **4235**, 2001; available electronically at http://www.inria.fr/rrrt/rr-4754.html.
5. B. Caillaud, E. Badouel and Ph. Darondeau. Distributing Finite Automata through Petri Net Synthesis. *Journal on Formal Aspects of Computing,* 13, 447–470, 2002.
6. C. Cassandras and S. Lafortune. *Introduction to discrete event systems.* Kluwer Academic Publishers, 1999.
7. J.-M. Couvreur, S. Grivet, D. Poitrenaud, Unfolding of Products of Symmetrical Petri Nets, 22nd International Conference on Applications and Theory of Petri Nets (ICATPN 2001), Newcastle upon Tyne, UK, June 2001, LNCS 2075, pp. 121–143.
8. J. Desel, and J. Esparza. *Free Choice Petri Nets.* Cambridge University Press, 1995.
9. J. Engelfriet. *Branching Processes of Petri Nets.* Acta Informatica 28, 1991, pp. 575–591.

10. J. Esparza, S. Römer. An unfolding algorithm for synchronous products of transition systems. in proc. of CONCUR'99, LNCS Vol. 1664, Springer Verlag, 1999.
11. E. Fabre, A. Benveniste, C. Jard. Distributed diagnosis for large discrete event dynamic systems. In *Proc of the IFAC congress,* Jul. 2002.
12. 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.
13. 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.
14. E. Fabre. Distributed diagnosis for large discrete event dynamic systems. In preparation.
15. E. Fabre. Convergence of the turbo algorithm for systems defined by local constraints. IRISA Res. Rep. 1510, 2003.
16. G.C. Goodwin and K.S. Sin. *Adaptive Filtrering, Prediction, and Control.* Prentice-Hall, Upper Sadle River, N.J. 1984.
17. S. Haar. Probabilistic Cluster Unfoldings. *Fundamenta Informaticae* 53(3–4), 281–314, 2002.
18. L. Lamport and N. Lynch. Distributed Computing: Models and Methods. in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics,* Jan van Leeuwen, editor, Elsevier (1990), 1157–1199.
19. S.L. Lauritzen. *Graphical Models*, Oxford Statistical Science Series 17, Oxford University Press, 1996.
20. S.L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statistical Society, Series B,* 50(2), 157–224, 1988.
21. R.J. McEliece, D.J.C. MacKay, J.-F. Cheng, Turbo Decoding as an Instance of Pearl's Belief Propagation Algorithm. *IEEE Transactions on Selected Areas in Communication,* 16(2), 140–152, Feb. 1998.
22. M. Nielsen and G. Plotkin and G. Winskel. Petri nets, event structures, and domains. Part I. *Theoretical Computer Science* **13**:85–108, 1981.
23. J. Pearl. Fusion, Propagation, and Structuring in Belief Networks. *Artificial Intelligence,* 29, 241–288, 1986.
24. L.R. Rabiner and B.H. Juang. An introduction to Hidden Markov Models. *IEEE ASSP magazine* **3**, 4–16, 1986.
25. M. Raynal. *Distributed Algorithms and Protocols.* Wiley & Sons, 1988.
26. W. Reisig. *Petri nets.* Springer Verlag, 1985.
27. M. Sampath, R. Sengupta, K. Sinnamohideen, S. Lafortune, and D. Teneketzis. Failure diagnosis using discrete event models. *IEEE Trans. on Systems Technology,* 4(2), 105–124, March 1996.
28. Y. Weiss, W.T. Freeman, On the Optimality of Solutions of the Max-Product Belief-Propagation Algorithm in Arbitrary Graphs. *IEEE Trans. on Information Theory,* 47(2), 723–735, Feb. 2001.