Symbolic Diagnosis of Partially Observable Concurrent Systems [⋆]

Thomas Chatain and Claude Jard

IRISA/ENS Cachan-Bretagne,
Campus de Beaulieu, F-35042 Rennes cedex, France
{Thomas.Chatain, Claude.Jard}@irisa.fr

Abstract. Monitoring large distributed concurrent systems is a challenging task. In this paper we formulate (model-based) diagnosis by means of hidden state history reconstruction, from event (e.g. alarm) observations. We follow a so-called true concurrency approach: the model defines explicitly the causal and concurrency relations between the observable events, produced by the system under supervision on different points of observation. The problem is to compute on-the-fly the different partial order histories, which are the possible explanations of the observable events. In this paper we extend our first method based on Petri nets unfolding to high-level parameterized Petri nets. This allows the designer to model data aspects (even on infinite domains) and non deterministic actions. The observation of such an action gives only partial information and the supervisor has to introduce parameters to represent the hidden aspects of the reached state. This supposes that the possible values for the parameters are symbolically computed and refined during supervision. In practice, non deterministic actions can also be used as an approximation to deal with incomplete information about the system. In this case the refinement of the parameters during supervision improves the knowledge of the model.

1 Introduction

Concurrent and distributed systems have been at the heart of computer science and engineering for decades. 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.

In this paper, we consider another instance of the use of formal models to master the complexity of distributed applications, namely the problem of inferring, from measurements, the hidden internal state of a distributed and asynchronous system. An important application is *distributed alarm correlation and fault diagnosis* in telecommunications networks, which motivated this work.

[⋆] This work was supported by the French RNRT project SWAN, funded by the Ministère de la Recherche; partners of the project are Inria, France Telecom R&D, Alcatel, QosMetrics, and Paris-Nord University.

The problem of recovering state histories from observations is pervasive throughout the general area of information technologies. For instance, estimating the state trajectory from noisy measurements is central in control engineering, with the Kalman filter as its most popular instance [8]; the same problem is considered in the area of pattern recognition, for stochastic finite state automata, in the theory of Hidden Markov Models [13]. For both cases, however, no extension exists to handle distributed systems. Finally, fault diagnosis in discrete event systems (e.g., automata) has been extensively studied [2,15], but the problem of dealing with concurrent model is just starting.

We follow a so-called true concurrency approach: the model defines explicitly the causal and concurrency relations between the observable events, produced by the system under supervision on different points of observation. The problem is to compute on-the-fly the different partial order histories, which are the possible explanations of the observable events. A natural candidate to formalize the approach are 1-safe Petri nets with branching processes and unfoldings. The previous work of our group used this framework to define the histories and a distributed algorithm to build them as a collection of consistent local views [1].

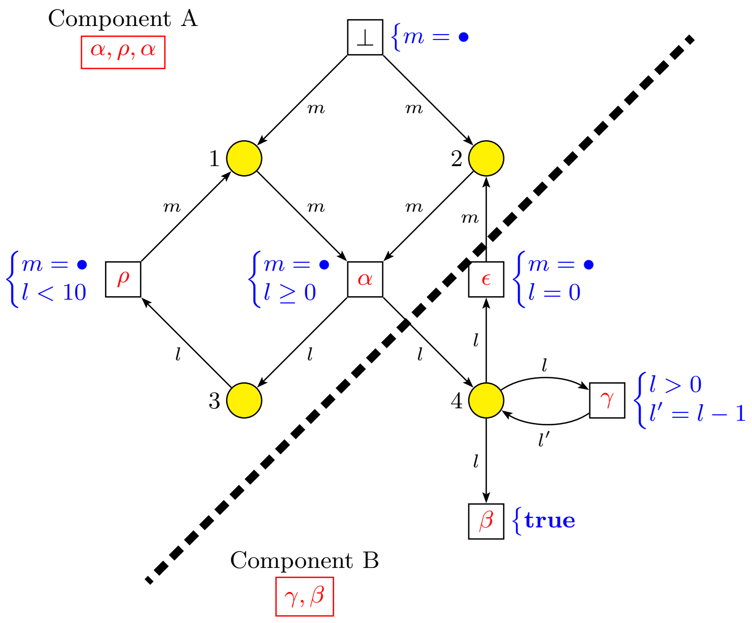
In this paper we extend our method to *high-level parameterized Petri nets*. This allows the designer to model data aspects (even on infinite domains) and non deterministic actions. The observation of such an action gives only partial information and the supervisor has to introduce parameters to represent the hidden aspects of the reached state. This supposes that the possible values for the parameters are symbolically computed and refined during supervision. In practice, non deterministic actions can also be used as an approximation to deal with incomplete information about the system. In this case the refinement of the parameters during supervision improves the knowledge of the model. We think this symbolic approach will be able to deal with more complex distributed systems. At the heart of our scientific contribution is the definition of a symbolic unfolding for high-level Petri nets, which combines the traditional unfolding [10,11] with a kind of α-conversion (λ-calculus) to deal with parameters. Up to our knowledge, this is original. The idea of using an unknown symbolic initial marking has already been addressed in [16], but restricted to the framework of simple Petri nets and their marking graphs.

This paper is organized as follows. We first begin in Section 2 by an informal presentation of the problem on a toy example, illustrating the high-level Petri net model we use, its unfolding and the trajectories we want to compute with respect to a given partially ordered observation. The mathematical background is recalled in Section 3, following the usual notation for Petri nets, as used for instance in [10]. In Section 4, we present an original algorithm to compute a symbolic unfolding. This allows us to formally express the diagnosis problem, which is done in Section 5 using a composition between the observation and the model, which can be then symbolically unfolded. We also show that unfolding can be performed on-the-fly, observable event by observable event. We conclude in Section 6 by presenting different perspectives on the use of the approach to monitor real distributed systems.

2 An example of diagnosis under partial observation

2.1 The parameterized concurrent model

Our parameterized concurrent model is based on the standard high-level Petri net introduced in [9] and augmented with free variables. It is exemplified in Figure 1, which shows two interacting components, named A and B. Component A may fail (observed as α) with a given non observable severity level (parameter l). To be completely repaired, component A must execute a local action (observed as ρ, and possible only if the severity level is less than 10), and wait for the completion of the recovery procedure of component B, which has been informed of the failure. To recover from a failure of severity l, component B must execute l repairs, observed as γ. But, at any time, component B may also fail and stop (observed as β). The initial transition $\perp$ starts the system in feeding the places 1 and 2 with black tokens (transported by the local variable m). Component A has two private states: safe, represented by place 1, and faulty, represented by place 3. Upon entering its faulty state, component A emits an alarm α. The failure of component A causes repairing actions in component B. This causality is modeled by the shared place 4. The monitoring system of component B (sensor B) only detects that component B provides an elementary action of repairing (observed as γ). The last action recovers the fail by putting the system again in state 2, shared with component A. This action is not observable.



Component A
α, ρ, α
$\perp$
$\{m = \bullet$
m
m
1
2
m
m
m
m
$\begin{cases} m = \bullet \\ l < 10 \end{cases}$
ρ
$\begin{cases} m = \bullet \\ l \geq 0 \end{cases}$
α
ϵ
$\begin{cases} m = \bullet \\ l = 0 \end{cases}$
l
l
l
l
l
3
4
γ
$\begin{cases} l > 0 \\ l' = l - 1 \end{cases}$
l'
l
β
$\{\mathbf{true}$
Component B
γ, β

Fig. 1. A concurrent machine with two components, which may fail with an unknown severity level and can be repaired accordingly.

All the observable events are also called *alarms* in the sequel, and represented by a grek letter on the figures. The fact that a transition is not observable is shown by writing ϵ instead of an alarm. It is to be noticed that the exact severity level of the fail l is not observable, and will be inferred during supervision using a kind of symbolic execution of the model.

In order to define the dynamics of such network, we consider that each place can be fed by a multiset of values (often called "colors"). These values are tested and forwarded by the transitions. As we can see, each transition associates a label $(\alpha, \beta, \gamma, \rho, \epsilon)$ and a predicate (printed near the transition in a curly brace, as a conjunction of expressions), called the *guard*. Furthermore, each incident edge is labeled by a local variable. The transition guard is composed with these local free variables. Informally, a transition is fireable if its guard is satisfiable. This means there exist some values to the variables for which the guard is true. One can thus select an instance of these values, which are unified (matched) to the variables. It is required that the values unified to the input arcs variables are present in the input places. The firing of the transition removes these values from the input places. The output places are then filled by the values unified to the output arc variables. In our example, the firing of the transition $\perp$ puts one token in places 1 and 2. The transition labeled by α becomes fireable. When it fires, it removes the tokens from 1 and 2 and puts a token in place 3 and an arbitrary integer l (provided $l \geq 0$) in place 4. The dynamics is formally defined in Section 3.

2.2 Supervision architecture

We consider the following setup for diagnosis, assuming that messages are not lost. Each sensor records its local alarms in sequence, while respecting causality (i.e. the observed sequence cannot contradict the causalities defined in the model). 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. This architecture is illustrated in Figure 2.

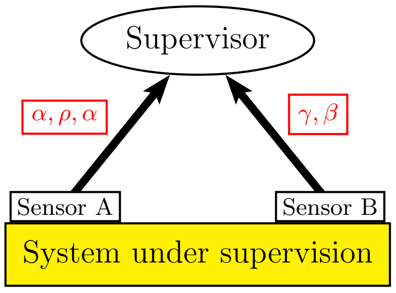


Fig. 2. The considered supervision architecture, composed of several sensors that report alarms asynchronously to a unique diagnoser.

For the development of the example, we consider that the system under supervision produces the sequences $\alpha\rho\alpha$ on sensor A, and $\gamma\beta$ on sensor B.

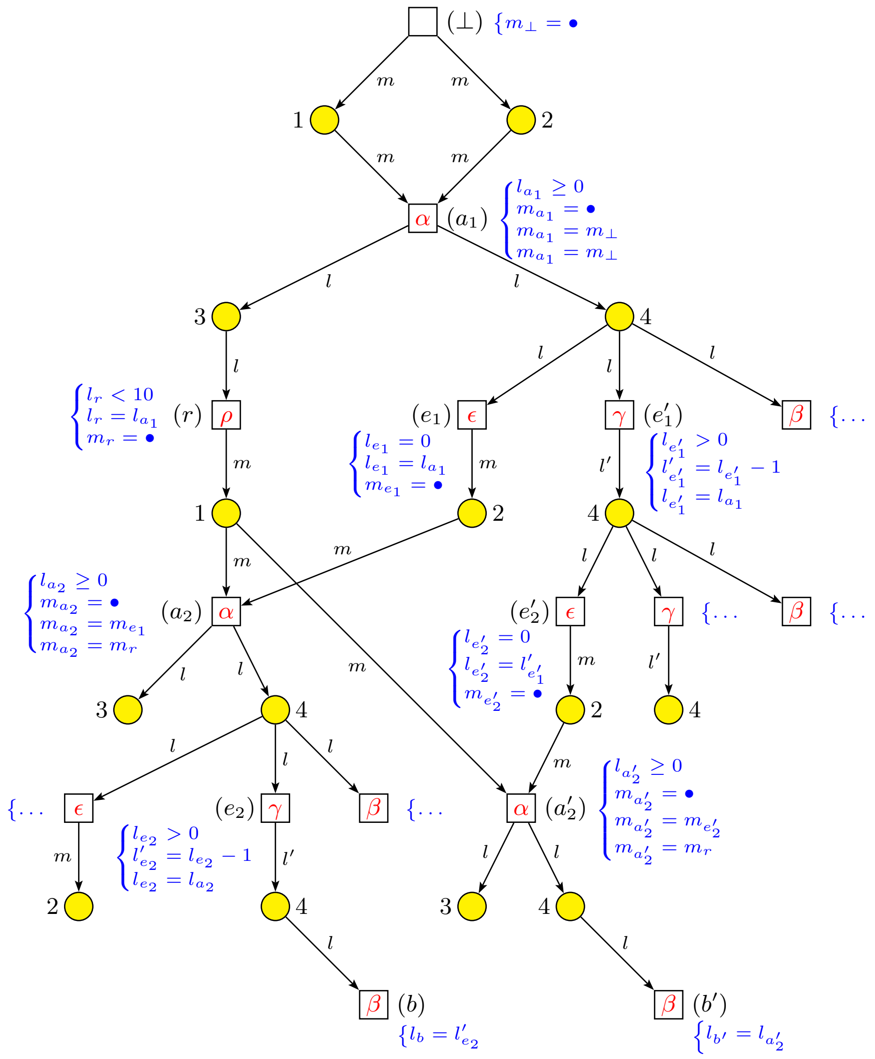
We think such an architecture is the first important step towards a distributed supervision, in which the monitoring is itself distributed, with different supervisors cooperating asynchronously. Each supervisor is attached to a sensor (i.e. a component of the model), records its local alarms in sequence, and can exchange supervision messages with the other supervisors, asynchronously. This aspect is deferred to a subsequent paper.

2.3 Unfoldings: an efficient data structure to represent all runs

The construction of the runs of the high-level parameterized Petri net of Figure 1 is illustrated in Figure 3.

The algorithm is to consider all the transitions of the original Petri net, and to place them, one at a time, if they are possible. Let us start by placing the initial transition $\perp$. Once placed, a transition becomes a unique event (denoted by $\perp$, a_1, e_1 etc.) in the graph. The local variables acquire then the status of global variables and for this purpose are renamed (actually indexed by the event name). An event e, instance of a transition t, is placed only if its preset (the input places) is present in the graph and if the following enabling condition is satisfiable. The enabling condition is formed by the conjunction of the local conditions of the events located in the causal past of e (see below the definition of causality) and of its local condition. The local condition is the guard of the transition t (in which the local variables have been renamed by their global names), augmented with the constraint that the variables of the input arcs have the same values that the variables of the output arcs of the input event of the input places, in order to capture the causal relation. To keep track of this condition, we associate a new predicate with the new event. In the graph of Figure 3, the local condition of each event is printed in a curly brace. This graph is usually infinite. We have drawn only a prefix of it. In the formal description of Section 4, the local condition is the predicate *loc_pred*(e) and the enabling condition is the predicate *pred*(e).

Two events linked by a path of the graph are *causally* related, since there exists a flow of values between them. Two events are *concurrent* if they are causally related and if they are not in *conflict* (i.e. cannot belong to a same run). There are two causes of conflict. The first one, called *structural conflict* is that they have been separated by a choice in the system, represented in the graph by a branching from an ancestor place of these events. The second possibility is specific to the parameterized model: two events are also in conflict (called *non-structural conflict*) if their predicates are not simultaneously satisfiable. We thus show that the symbolic unfolding is an interesting structure to represent the different runs, in which causality and concurrency are explicitly given. The different runs are superimposed in the graph and separated by the notion of conflict. In Figure 3, the event r is a cause of event a_2'; the event e_1 is concurrent with event r; event a_2 is structurally in conflict with event e_1'. A non-structural conflict is also possible between the event r and an event labeled by γ reachable after more than 10 consecutive repairs on component B (not represented in the prefix chosen in the figure).



(⊥) {$m_\perp = \bullet$
1
2
m
(a_1)
$l_{a_1} \geq 0$, $m_{a_1} = \bullet$, $m_{a_1} = m_\perp$, $m_{a_1} = m_\perp$
α
3
4
l
(r)
ρ
$l_r < 10$, $l_r = l_{a_1}$, $m_r = \bullet$
(e_1)
ϵ
$l_{e_1} = 0$, $l_{e_1} = l_{a_1}$, $m_{e_1} = \bullet$
γ
(e'_1)
$l_{e'_1} > 0$, $l'_{e'_1} = l_{e'_1} - 1$, $l_{e'_1} = l_{a_1}$
β
{…
l'
(a_2)
$l_{a_2} \geq 0$, $m_{a_2} = \bullet$, $m_{a_2} = m_{e_1}$, $m_{a_2} = m_r$
(e'_2)
$l_{e'_2} = 0$, $l_{e'_2} = l'_{e'_1}$, $m_{e'_2} = \bullet$
(e_2)
$l_{e_2} > 0$, $l'_{e_2} = l_{e_2} - 1$, $l_{e_2} = l_{a_2}$
(a'_2)
$l_{a'_2} \geq 0$, $m_{a'_2} = \bullet$, $m_{a'_2} = m_{e'_2}$, $m_{a'_2} = m_r$
(b)
{$l_b = l'_{e_2}$
(b')
{$l_{b'} = l_{a'_2}$

Fig. 3. Some runs of the example represented in a branching process.

2.4 Asynchronous diagnosis

Figure 3 showed different runs of the system, represented in a single graph. The question now is to select the runs that are compatible with the observations. In Figure 4, we have projected the graph of Figure 3 by considering that some events are not compatible with the actual observation. This is the case for instance for the first β transitions, which cannot be considered since γ have to be explained before and that the occurrence of β stops the production of γ in the model. The resulting graph shows two possible explanations: the first corresponds to the left part of the graph with the following partial order $\alpha.(\rho \parallel \epsilon).\alpha.\gamma.\beta$; the second is the right part of the graph: $\alpha.(\rho \parallel (\gamma.\epsilon)).\alpha.\beta$. We see that these two possible explanations share a same prefix $\alpha.\rho$ in the graph. Another interesting fact is the refinement of constraints on variables during the unfolding: for instance, at the end of the first explanation, we can infer that the severity level of the first fail α was 0, because of the conjunction of the predicates of the events a_1 and e_1.

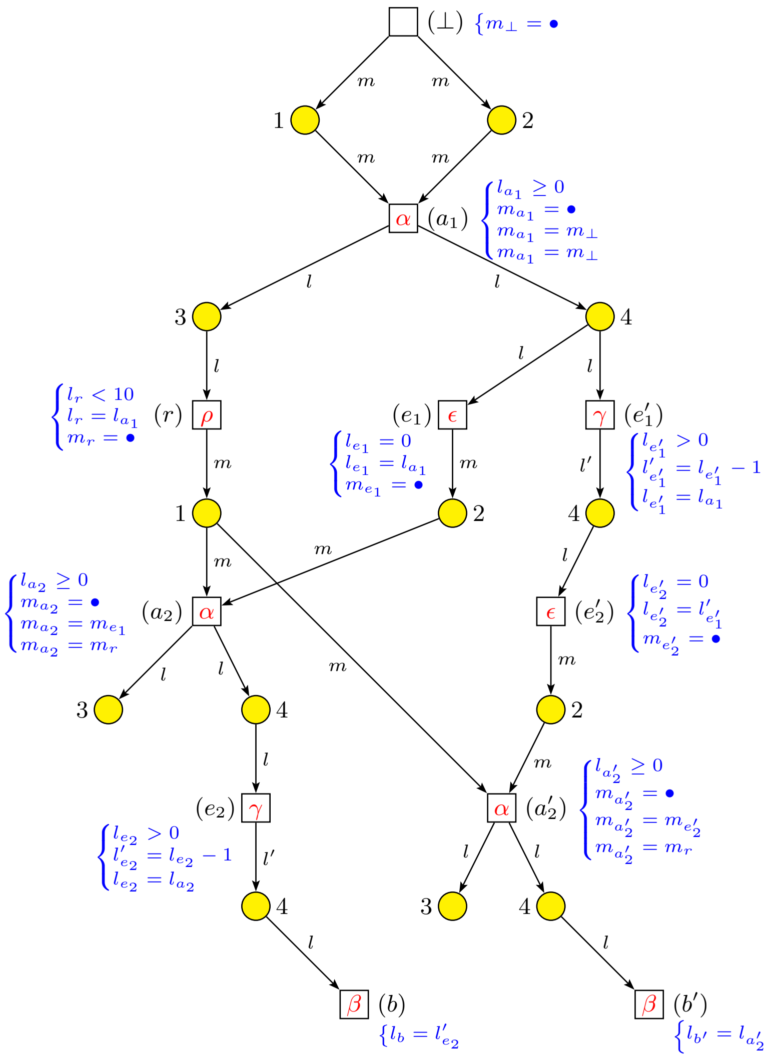

$(\bot)$ $\{m_\bot = \bullet$
(a_1) $l_{a_1} \geq 0$, $m_{a_1} = \bullet$, $m_{a_1} = m_\bot$, $m_{a_1} = m_\bot$
(r) $l_r < 10$, $l_r = l_{a_1}$, $m_r = \bullet$
(e_1) $l_{e_1} = 0$, $l_{e_1} = l_{a_1}$, $m_{e_1} = \bullet$
(e_1') $l_{e_1'} > 0$, $l'_{e_1'} = l_{e_1'} - 1$, $l_{e_1'} = l_{a_1}$
(a_2) $l_{a_2} \geq 0$, $m_{a_2} = \bullet$, $m_{a_2} = m_{e_1}$, $m_{a_2} = m_r$
(e_2') $l_{e_2'} = 0$, $l_{e_2'} = l'_{e_1'}$, $m_{e_2'} = \bullet$
(e_2) $l_{e_2} > 0$, $l'_{e_2} = l_{e_2} - 1$, $l_{e_2} = l_{a_2}$
(a_2') $l_{a_2'} \geq 0$, $m_{a_2'} = \bullet$, $m_{a_2'} = m_{e_2'}$, $m_{a_2'} = m_r$
(b) $\{l_b = l'_{e_2}$
(b') $\{l_{b'} = l_{a_2'}$

Fig. 4. The causal graph resulting of our diagnosis algorithm.

In practice, the desired projection is obtained by synchronizing the system model with the observations. This augmented model is then unfolded. The last phase is to keep only the system part of the unfolding to present the explanations to the user. Figure 5 shows our original model, constrained with the considered observations. The sequencing of local observations are represented as the linear nets at the left and right parts of the figure. The observations constrain the execution of the original model since the treatment of the next local observation requires that a transition with the same label in the model has been fired. This is the role of places A, B, and their complements $\overline{A}$ and $\overline{B}$ in the figure.

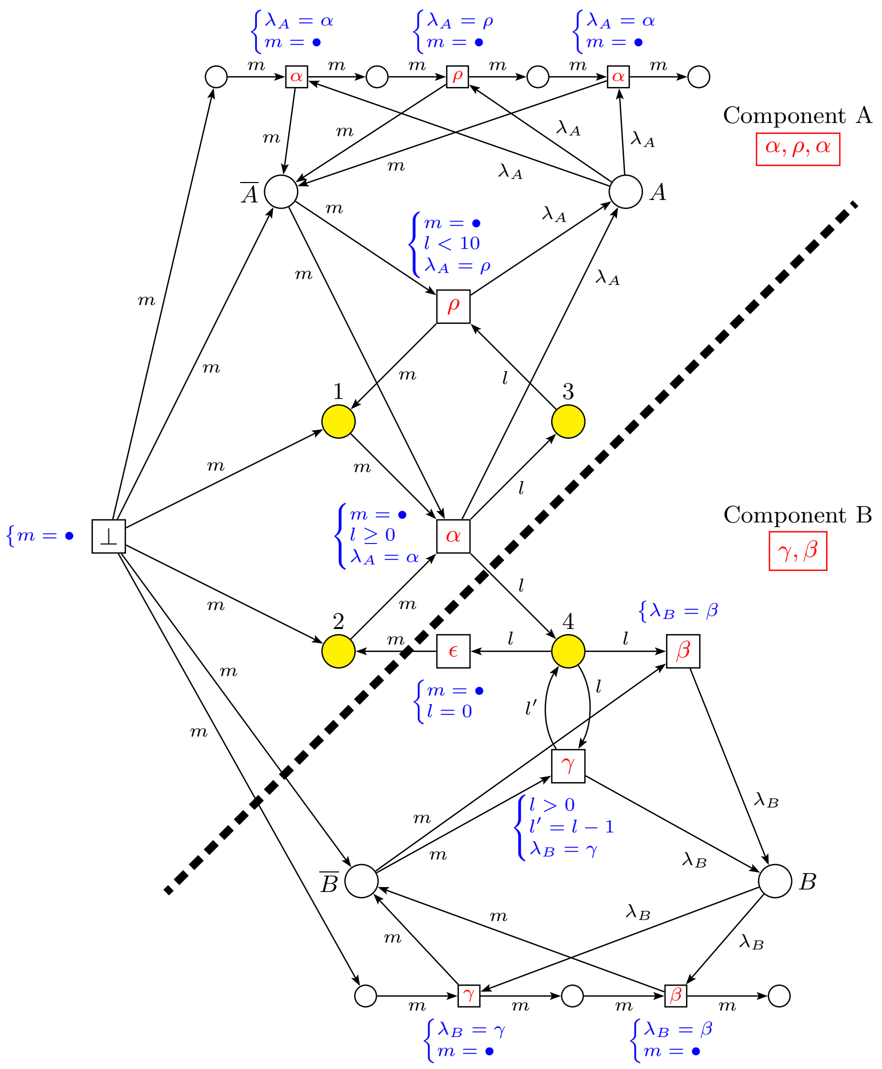

Component A
α, ρ, α
Component B
γ, β
$\{\lambda_A = \alpha, m = \bullet$
$\{\lambda_A = \rho, m = \bullet$
$\{m = \bullet, l < 10, \lambda_A = \rho$
$\{m = \bullet, l \geq 0, \lambda_A = \alpha$
$\{m = \bullet$
$\{\lambda_B = \beta$
$\{m = \bullet, l = 0$
$\{l > 0, l' = l - 1, \lambda_B = \gamma$
$\{\lambda_B = \gamma, m = \bullet$
$\{\lambda_B = \beta, m = \bullet$
$\overline{A}$
A
$\overline{B}$
B
1
2
3
4

Fig. 5. The model of Figure 1, constrained by the observation.

The rest of the paper defines mathematically these different objects and operations. The final contribution is an on-the-fly algorithm, which builds the different possible explanations in the form of an unfolding, increasing step by step at each observation.

3 Mathematical background: high-level Petri nets

Basic references are [2,4,14]. We use the standard notations, adopted from [10].

3.1 Notations

We recall the notations:

- $f : A \longmapsto B$ denotes a mapping f from A to B;
- $A \uplus B$ denotes the disjoint union of the sets A and B;
- $e[n \leftarrow n']$ is the expression e in which all the occurrences of the name n have been replaced by the expression n'.
  $e[n \leftarrow f(n)]_{n \in N}$ is the result of the parallel replacement of each name $n \in N$ by the expression $f(n)$.

A *multiset* over a set X is a mapping $\mu : X \longmapsto \mathbb{N}$. We denote $x \in \mu$ if $\mu(x) > 0$. We define the empty multiset $\emptyset$ as $\emptyset(x) \overset{\text{def}}{=} 0$ for all $x \in X$. We define the union of two multisets μ_1 and μ_2 over X as $(\mu_1 + \mu_2)(x) \overset{\text{def}}{=} \mu_1(x) + \mu_2(x)$ for all $x \in X$. For two multisets μ and μ' over X, we write $\mu \leq \mu'$ if for all $x \in X$, $\mu(x) \leq \mu'(x)$. A multiset μ is finite if $\{x \in X \mid x \in \mu\}$ is finite. In this case we can represent it with $\{\!|\ldots|\!\}$ delimiters. For example $\{\!|a, a, b|\!\}$ will denote the multiset μ defined by $\mu(a) = 2$, $\mu(b) = 1$ and $\mu(x) = 0$ for all $x \in X \setminus \{a, b\}$. For a mapping $h : X \longmapsto Y$, we denote $\{\!|h(x) \mid x \in \mu|\!\}$ or $h(\mu)$ the multiset μ' over Y such that for all $y \in Y$, $\mu'(y) \overset{\text{def}}{=} \sum_{x \in X \wedge h(x) = y} \mu(x)$.

3.2 High-level Petri nets

In this section we present the formal model we use to represent the system we work on and its behavior. The example of Figure 1 illustrates this model.

It is assumed that there exists a (finite or infinite) set *Tok* of elements (or 'colors') [1] and a set *VAR* of variable names, such that $\mathit{Tok} \cap \mathit{VAR} = \emptyset$.

A *high-level Petri net* is a quadruple $N \overset{\text{def}}{=} (P, T, W, \iota)$ such that:

- P and T are disjoint sets of *places* and *transitions* respectively;
- W is a multiset over $(P \times \mathit{VAR} \times T) \cup (T \times \mathit{VAR} \times P)$ of arcs;
- ι maps each $t \in T$ to a predicate $\iota(t)$ on $\mathit{VAR}(t)$, where $\mathit{VAR}(t) \overset{\text{def}}{=} \{v \mid (p, v, t) \in W \vee (t, v, p) \in W\}$. For every $t \in T$, $\iota(t)$ is called the *guard* of t.

[1] We do not mention any type conditions on the colors because this is not essential for our application. Adding types would only be a refinement of the firing conditions of the transitions.

For two nodes $y, y' \in P \cup T$, we denote $y \rightarrow y'$ if there exists a variable v such that $(y, v, y') \in W$. The reflexive and irreflexive transitive closures of $\rightarrow$ are denoted respectively by $\preceq$ and $\prec$. For a transition $t \in T$, let ${}^\bullet t \overset{\text{def}}{=} \{\!|(p, v) \mid (p, v, t) \in W|\!\}$, $t^\bullet \overset{\text{def}}{=} \{\!|(p, v) \mid (t, v, p) \in W|\!\}$.

In figures, places are usually represented by circles and transitions by squares. Labeled arrows between places and transitions represent the arcs. The guards of the transitions are printed in a curly brace.

A *homomorphism* from a high-level Petri net $N = (P, T, W, \iota)$ to a high-level Petri net $N' = (P', T', W', \iota')$ is a mapping $h : P \cup T \longmapsto P' \cup T'$ such that:

- $h(P) \subseteq P'$ and $h(T) \subseteq T'$;
- for all $t \in T$, $\begin{cases} {}^\bullet h(t) = \{\!|(h(p), v) \mid (p, v) \in {}^\bullet t|\!\} \\ h(t)^\bullet = \{\!|(h(p), v) \mid (p, v) \in t^\bullet|\!\} \\ \iota'(h(t)) = \iota(t) \end{cases}$

A *firing mode* of a transition t is a mapping $\sigma : \mathit{VAR}(t) \longmapsto \mathit{Tok}$ such that $\iota(t)$ evaluates to **true** under the substitution given by σ. We denote ${}^\bullet(t, \sigma) \overset{\text{def}}{=} \{\!|(p, \sigma(v)) \mid (p, v) \in {}^\bullet t|\!\}$ and $(t, \sigma)^\bullet \overset{\text{def}}{=} \{\!|(p, \sigma(v)) \mid (p, v) \in t^\bullet|\!\}$.

A *marking* of a net N is a multiset over $P \times \mathit{Tok}$. A transition t is *enabled* at marking M with firing mode σ if ${}^\bullet(t, \sigma) \leq M$. Such a transition can *fire*, leading to a new marking $M' \overset{\text{def}}{=} M - {}^\bullet(t, \sigma) + (t, \sigma)^\bullet$.

A *high-level Petri net system* is a high-level Petri net $\Upsilon \overset{\text{def}}{=} (P, T, W, \iota)$, which has a unique initial transition called $\perp$ such that ${}^\bullet\!\perp = \emptyset$. In the sequel we assume that $\iota(\perp)$ is satisfiable, i.e. $\perp$ has at least one firing mode. $\perp$ fires only once, at the empty marking, to start the system.

Remark: low-level Petri nets can be seen as particular high-level Petri nets, in which all the arcs use the same variable m, and all the guards are $(m = \bullet)$. The drawback with low-level Petri nets is the lack of manipulations of data. In practice, the data aspects have to be enumerated, and thus explode and are limited to finite domains for variables. This is why we consider the extension to the so-called high-level Petri nets.

4 Symbolic unfolding

This section formally defines the structure we use to represent the different runs of a system. Figure 3 shows a symbolic branching process of the system of Figure 1. For each event e, the predicate *loc_pred*(e) is printed near the event.

4.1 High-level occurrence nets

The net $N \overset{\text{def}}{=} (P, T, W, \iota)$ is called *ordinary* if for each pair y, y' of nodes of N, there exists at most one arc connecting y and y' $(\sum_{v \in \mathit{VAR}} W((y, v, y')) \leq 1)$.

Two nodes (places or transitions), y and y', of an ordinary net $N \stackrel{\text{def}}{=} (P, T, W, \iota)$ are in *structural conflict*, denoted by $y \# y'$, if there exist distinct transitions $t, t' \in T$ and a place $p \in P$ such that $p \to t$, $p \to t'$, $t \preceq y$ and $t' \preceq y'$. A node y is in *structural self-conflict* if $y \# y$.

A *high-level occurrence net* is an ordinary net system $ON \stackrel{\text{def}}{=} (B, E, G, \iota)$, where B is a set of *conditions* (places), E is a set of *events* (transitions) and G is a flow relation, satisfying the following conditions:

– for every $b \in B$, there exists a unique pair (e, v) called ${}^\bullet b$ such that $(e, v, b) \in G$;
– for every $y \in B \cup E$, $\begin{cases} \neg(y \# y) \\ \neg(y \prec y) \\ \bot \preceq y \\ \text{there are finitely many } y' \text{ such that } y' \prec y. \end{cases}$

$\prec$ is called the *causality relation*. We say that node y is *causally related* to node y' if $y \prec y'$.

For all $e \in E$ we denote $\lceil e \rceil \stackrel{\text{def}}{=} \{f \in E \mid f \preceq e\}$. For all $F \subseteq E$ we denote $\lceil F \rceil \stackrel{\text{def}}{=} \bigcup_{f \in F} \lceil f \rceil$.

For a high-level occurrence net $ON \stackrel{\text{def}}{=} (B, E, G, \iota)$ we define the mappings *loc_pred* and *pred* which map each $e \in E$ to the predicates

$$\begin{aligned} \mathit{loc\_pred}(e) \stackrel{\text{def}}{=} \quad & \iota(e)[v \leftarrow v_e]_{v \in VAR(e)} \\ & \wedge \bigwedge_{(b,v) \in {}^\bullet e} (v_e = v'_{e'}) \qquad \text{with } {}^\bullet b = (e', v') \\ \mathit{pred}(e) \stackrel{\text{def}}{=} & \bigwedge_{f \preceq e} \mathit{loc\_pred}(f) \end{aligned}$$

4.2 Symbolic branching processes

A *symbolic branching process* of Υ is a pair $\pi \stackrel{\text{def}}{=} (ON, h)$ such that:

– ON is a high-level occurrence net such that for all $e \in E$, *pred*(e) is satisfiable;
– h is a homomorphism from ON to Υ;
– $h(\bot) = \bot$;
– for all $e, f \in E$, if $h(e) = h(f)$ and ${}^\bullet e = {}^\bullet f$, then $e = f$.

4.3 Non structural conflict, concurrency

In branching processes of high-level Petri nets, the causality relation is the same as in branching processes of low-level Petri nets. But there are two different causes of conflict. The *structural conflict* is the equivalent of the conflict relation in branching processes of low-level Petri nets; and we define a *non structural conflict*, that restricts the concurrency relation. This notion of non structural conflict is due to the existence of symbolic parameters.

The events of the set $F \subseteq E$ are in *non structural conflict* if $\bigwedge_{f\in F} \mathit{pred}(f)$ is not satisfiable. We note that for all F in non structural conflict and $F' \subseteq E$, if $\lceil F \rceil \subsetneq \lceil F' \rceil$ then F' is also in non structural conflict.

The events of F are in *minimal non structural conflict* if there does not exist any $F' \subseteq E$ such that $\lceil F' \rceil \subsetneq \lceil F \rceil$ and the events of F' are in non structural conflict.

The events of the set $F \subseteq E$ are *concurrent* if they are not in non structural conflict, and for each $e, e' \in F$, neither $e \prec e'$, nor $e' \prec e$, nor $e \mathbin{\#} e'$ holds.

A *co-set* is a set C of conditions such that the events $\{e \in E \mid \exists b \in C \quad e \to b\}$ are not in (structural or non structural) conflict, and there does not exist any $b, c \in C, e \in E$ such that $b \to e \prec c$.

4.4 Symbolic unfolding

The set of all symbolic branching processes of a high-level Petri net system 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 π, π' two symbolic branching processes, π' is a prefix of π, written $\pi' \sqsubseteq \pi$, if there exists an injective homomorphism ϕ from π' into π, such that $\phi(\perp) = \perp$, and the composition $h \circ \phi$ coincides with h', where $\circ$ denotes the composition of maps.

Thus, the notion of unfolding of a Petri net as the unique maximum branching process up to isomorphism, proved in theorem 23 of [3], can be adapted to symbolic branching processes of high-level Petri nets to define the *symbolic unfolding* $\mathcal{U}_{\Upsilon}$ of a high-level Petri net system Υ.

Branching processes of a (high-level) Petri net represent the different runs. The interest is that the causalities and the concurrency between the transitions figuring in the run are explicitly represented in a graph. This is why, this kind of behavioral semantics for Petri nets is called "true concurrency semantics", and fits particularly well with the kind of trajectories we want to produce as the monitoring activity.

Some applications use the notion of finite complete prefix defined on low-level Petri nets. We do not need this notion in the area of diagnosis because the unfoldings we generate are finite, as the model is constrained by the observation. Furthermore we think that it would not be obvious to define a notion of finite complete prefix for symbolic unfoldings of high-level Petri nets, because of the theoretical power of the model.

4.5 Algorithm

We propose an algorithm to compute the symbolic unfolding of a high-level Petri net. This algorithm needs to decide if the predicates $\mathit{pred}(e)$ are satisfiable. This is possible if the guards of the transitions are expressed in some weak enough language. One possible framework is the use of Presburger arithmetics [12] (arithmetics without multiplication).

The algorithm consists in a non deterministic iteration, after the placement of the initial event $\perp$. In each iteration we choose a transition t and a co-set C to create a new event e. The predicate $\mathit{pred}(e)$ is memorized for each event. The minimal non structural conflicts are memorized in the variable *conflict*, which is used to find the co-sets.

In the area of diagnosis, the net is constrained by the observation as we will see in Section 5. Thus its unfolding is finite and the algorithm terminates, if we except models that contain loops of non observable transitions. But in the general case the unfolding may be infinite, and precautions have to be taken to ensure that all the events of the unfolding are computed. One method is to use the *causal depth* of the events defined as follows: the *causal depth* of an event $e \in E$ is the number of events on the longest path from $\perp$ to e. For all integer n, the number of events at depth n is finite. If the algorithm is forced to compute all the events at depth n before those at depth $n+1$, then all the events will be computed.

Initialization

```
1. initialize the sets $B, E, G$ to $\emptyset$, $h$ and $\mathit{pred}$ to the empty mapping and
   $\mathit{conflict}$ to $\emptyset$;
2. add the event $\perp$ to $E$, and update $h$ with $h(\perp) = \perp$;
3. for each $(p, v) \in \perp^\bullet$, add a new condition $b$ to $B$, add $(\perp, v, b)$ to $G$ and
   update $h$ with $h(b) = p$;
4. extend $\mathit{pred}$ with $\mathit{pred}(\perp) = \iota(\perp)[v \leftarrow v_\perp]_{v \in VAR(\perp)}$;
```

Non deterministic iteration

Repeat until no transition can be chosen:

```
1. choose nondeterministically a transition $t \in T \setminus \{\perp\}$ such that there exist
   a co-set $C$ and a bijection $\mathit{pin}$ from ${}^\bullet t$ to $C$, satisfying:
   – for all $(p, v) \in {}^\bullet t$, $h(\mathit{pin}((p, v))) = p$;
   – the predicate $\mathit{pred\_e} \overset{\text{def}}{=} \mathit{loc\_pred} \wedge \bigwedge_{b \in C} \mathit{pred}(b)$ is satisfiable, where:
      • $\mathit{pred}(b) \overset{\text{def}}{=} \mathit{pred}(e')$   with ${}^\bullet b = (e', v')$
      • $\mathit{loc\_pred} \overset{\text{def}}{=} \iota(t)[v \leftarrow v_e]_{v \in VAR(t)}$
                   $\wedge \bigwedge_{(p,v) \in {}^\bullet t} (v_e = v'_{e'})$   with ${}^\bullet \mathit{pin}((p, v)) = (e', v')$
      • $e$ is a new event.
2. add the event $e$ to $E$, and update $h$ with $h(e) = t$;
3. for each $(p, v) \in {}^\bullet t$, add $(\mathit{pin}((p, v)), v, e)$ to $G$;
4. for each $(p, v) \in t^\bullet$, add a new condition $b$ to $B$, add $(e, v, b)$ to $G$ and
   update $h$ with $h(b) = p$;
5. extend $\mathit{pred}$ with $\mathit{pred}(e) = \mathit{pred\_e}$;
6. extend $\mathit{conflict}$ with the newly created minimal non structural conflicts,
   if any.
```

5 Symbolic diagnosis: formal problem setting

5.1 Observations

Observations and their impact on the original system model are represented by adding new places and transitions in the high-level Petri net.

A *sensor* is a place s of a high-level Petri net that has no output arc and at most one input arc from each transition $t \in T$. To simplify the notations, we assume that the variable associated with this arc is always λ_s. When a transition $t \in T$ fires, the value taken by λ_s is called the alarm.

A *local observation sequence* from the sensor s is a finite sequence of alarms $(\lambda_{s,1}, \ldots, \lambda_{s,n_s})$. A *global observation* from a set S of sensors is a mapping A from sensors $s \in S$ to observation sequences $(\lambda_{s,1}, \ldots, \lambda_{s,n_s})$. Consider two observations A and A', which associate with each sensor $s \in S$, the observation sequences $(\lambda_{s,1}, \ldots, \lambda_{s,n_s})$ and $(\lambda'_{s,1}, \ldots, \lambda'_{s,n'_s})$ respectively. We say that A is a *prefix* of A', written $A \leq A'$, if for all $s \in S$, $n_s \leq n'_s$ and $(\lambda_{s,1}, \ldots, \lambda_{s,n_s}) = (\lambda'_{s,1}, \ldots, \lambda'_{s,n_s})$.

5.2 Diagnosis net $\mathcal{D}(N, A)$

In this section we show how to build a net $\mathcal{D}(N, A)$ from a net N modeling a system and an observation A of this system. The idea is to constrain the model so that each transition of the model that sends an alarm to a sensor s is not allowed to fire until all the previous alarms sent to s have been treated. To achieve this we create a new place $\overline{s}$, add an arc from $\overline{s}$ to each transition that sends an alarm to s, and ensure that s contains a token if and only if all the alarms sent to s have been treated. The treatment of the alarms received by sensor s is modeled by a set of new transitions $t_{s,i},\ i = 1, \ldots, n_s$ (one for each observation). Transition $t_{s,i}$ guarantees that the i^{th} alarm received by s matches the observation $\lambda_{s,i}$. Once the alarm is treated, $t_{s,i}$ puts a token in the place $\overline{s}$, which allows the transitions of the model to emit new alarms. The formal definition of $\mathcal{D}(N, A)$ follows.

For a net $N \stackrel{\text{def}}{=} (P_N, T_N, W_N, \iota_N)$ and an observation A from a set S of sensors of N, we define the net $\mathcal{D}(N, A) \stackrel{\text{def}}{=} (P, T, W, \iota)$, called *net N observed as A*, as follows (we assume that m is a fresh variable name):

- $P \stackrel{\text{def}}{=} P_N \uplus \{\overline{s} \mid s \in S\} \uplus \{p_{s,i} \mid s \in S,\ i = 0, \ldots, n_s\}$
- $T \stackrel{\text{def}}{=} T_N \uplus \{t_{s,i} \mid s \in S,\ i = 1, \ldots, n_s\}$
- $W \stackrel{\text{def}}{=} W_N + \{\!|(\bot, m, \overline{s}), (\bot, m, p_{s,0}) \mid s \in S|\!\}$
  $+ \{\!|(\overline{s}, m, t) \mid s \in S \wedge (t, \lambda_s, s) \in W_N|\!\}$
  $+ \{\!|(s, \lambda_s, t_{s,i}), (t_{s,i}, m, \overline{s}) \mid s \in S,\ i = 1, \ldots, n_s|\!\}$
  $+ \{\!|(p_{s,i-1}, m, t_{s,i}), (t_{s,i}, m, p_{s,i}) \mid s \in S,\ i = 1, \ldots, n_s|\!\}$
- $\iota(t) \stackrel{\text{def}}{=} \iota_N(t) \wedge (m = \bullet)$ if $t \in T_N$
  $\iota(t_{s,i}) \stackrel{\text{def}}{=} (\lambda_s = \lambda_{s,i}) \wedge (m = \bullet)$ for all $s \in S,\ i = 1, \ldots, n_s$

Figure 5 shows the net of Figure 3 observed as α, ρ, α from sensor A and γ, β from sensor B.

Remark. For two observations A and A' such that $A \leq A'$, $\mathcal{D}(N, A)$ is a subnet of $\mathcal{D}(N, A')$. Indeed $\mathcal{D}(N, A')$ can be built from $\mathcal{D}(N, A)$ by adding the places and transitions required by the new alarms, and arcs connecting the new transitions. No new arc is added to the old transitions. That is why every execution of the net $\mathcal{D}(N, A)$ is also a valid execution of $\mathcal{D}(N, A')$.

5.3 Global diagnosis

We call *diagnosis of observation A on net N* the symbolic unfolding $\mathcal{U}_{\mathcal{D}(N,A)}$ of the net N observed as A. For each set $F \subseteq E$ of concurrent events such that the restriction of h to F is a bijection from F to $\{t_{s,n_s} \mid s \in S\}$, the configuration $\lceil F \rceil$ explains the observation A.

We may want to get rid of the causalities due to the observation. For this purpose, we remove all the events and conditions corresponding to the sensors or to the observation. This operation, called projection on N removes the causalities due to the observation. But we must keep the information of the (structural and non structural) conflicts due to the observation, that do not appear any more in the projected net.

Figure 4 shows all the possible explanations of the example of Figure 3.

On-the-fly computation. The unfolding of $\mathcal{D}(N, A)$ can be computed by the algorithm of Section 4.5. Moreover, we can adapt this algorithm in order to compute on-the-fly the partial order histories that explain the observed alarms. Indeed, if A and A' are two observations such that A is a prefix of A', then, consecutively to the final remark of Section 5.2, each branching process of $\mathcal{D}(N, A)$ is also a branching process of $\mathcal{D}(N, A')$. Then we can compute on-the-fly the explanations by updating $\mathcal{D}(N, A)$ each time a new alarm is observed. After this modification is done, the algorithm will continue and compute the explanations of the new observation.

6 Conclusion

We have presented a possible approach to the supervision/diagnosis of distributed systems, in which the explanations are given by a family of partial orders on the observable events, represented by an unfolding graph. The main contribution of the paper is to consider parameters in the model. These parameters are used to model incomplete information on the system under supervision (i.e. partially observed). We think it is an important aspect to deal with real contexts. We have different perspectives. From the practical point of view, we are starting the implementation of the algorithm. The main extension we plan is to deal with a distributed supervision architecture; that is extend the approach presented in [7] to the symbolic framework we consider. An other work in progress is to study time Petri nets as a particular case of our parameterized model. The variables of the model are used to model the different instant of transition firings. This will define a new notion of unfolding for time Petri nets, which keeps concurrency.

More generally, because of the "local" property of the unfolding algorithm, we think our approach could be extended to deal with dynamic systems, in which the model can evolve during observation.

References

1. Benveniste, A., Fabre, E., Jard, C., Haar, S.: Diagnosis of asynchronous discrete event systems, a net unfolding approach. IEEE Trans. on Automatic Control **48(5)** (2003) 714-727
2. Cassandras, C., Lafortune, S.: Introduction to discrete event systems. Kluwer Academic Publishers (1999)
3. Engelfriet, J.: Branching processes of Petri nets. Acta Informatica **28** (1991) 575–591
4. Desel, J., and Esparza, J.: Free choice Petri nets. Cambridge University Press (1995)
5. Esparza, J., Römer, S., Vogler, W.: An improvement of McMillan's unfolding algorithm. In: Proc. of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'1996) T. Margaria and B. Steffen (Eds.), Springer-Verlag, Lecture Notes in Computer Science **1664** (1999) 2–20.
6. Esparza, J., Römer, S.: An unfolding algorithm for synchronous products of transition systems. In: Proc. of CONCUR'99, Springer-Verlag, Lecture Notes in Computer Science **1664** (1999)
7. Fabre, E.: Monitoring distributed systems with distributed algorithms. In: Proc. of the 2002 IEEE Conf. on Decision and Control (2002) 411–416
8. Goodwin, G.C., Sin, K.S.: Adaptive filtering, prediction, and control. Prentice-Hall, Upper Sadle River, N.J. (1984)
9. Jensen, K.: Coloured Petri nets. Basic concepts, analysis methods and practical use. EATCS Monographs on Theoretical Computer Science, Springer-Verlag (1992)
10. Khomenko, V., Koutny, M.: Branching processes of high-level Petri nets. In: Proc. of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2003), H. Garavel and J. Hatcliff (Eds.), Springer-Verlag, Lecture Notes in Computer Science (2003)
11. Kozura, V.E.: Unfolding of coloured Petri nets. Technical Report 80, A.P. Ershov Institute of Informatics Systems (2000)
12. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. Comptes Rendus du Premier Congrès des Mathématiciens des Pays Slaves **395** (1927) 92–101
13. Rabiner, L.R., Juang, B.H.: An introduction to hidden Markov models. IEEE ASSP magazine **3** (1986) 4–16
14. Reisig, W.: Petri Net: an introduction. ETACS Monographs on Theoretical Computer Science, Springer-Verlag **4** (1985)
15. Sampath, M., Sengupta, R., Sinnamohideen, K., Lafortune, S., Teneketzis, D.: Failure diagnosis using discrete event models. IEEE Trans. on Systems Technology **4(2)** (1996) 105–124
16. Vernier, I. Symbolic executions of symmetrical parallel programs. In: Proc. of 4th Euromicro Workshop on Parallel and Distributed Processing, Braga, Portugal (1996) 327–334