DISTRIBUTED DIAGNOSIS FOR LARGE DISCRETE EVENT DYNAMIC SYSTEMS [1]

E. Fabre, [*] **A. Benveniste,** [*] **C. Jard** [**]

[*] *IRISA/INRIA, Rennes, France*
[**] *IRISA/CNRS, Rennes, France*

AbstractThis paper presents a framework to deal with large systems, which cannot be handled as a whole. We propose to model them as a graph of interacting subsystems, and to base all processings on this factorization of the large system.

Keywords: distributed system, large-scale system, discrete-event dynamic system, diagnosis, state estimation, dynamic programming

1. INTRODUCTION

In many applications, systems are too large to be handled as a whole: the number of possible states of the system explodes. The situation can be worse: some distributed systems, like a telecommunication network for example, are made of independent components, evolving with their own clock, and interacting on some particular events. As a consequence, no notion of global time is available. To make supervision problems tractable, and in particular failure diagnosis, we model a distributed system as a graph of interacting subsystems, with the appropriate semantics of trajectories and stochastic framework. A centralized supervisor, collecting all observations from the system and knowing a model of the whole system, may not be affordable. We advocate instead a processing "by parts," and push the idea up to a completely distributed supervising architecture, with one local supervisor sitting on top of each subsystem, and coordinating its activity with supervisors in its neighborhood. This framework has very tight connections with Markov fields theory and Bayesian networks, which allows to adapt all results and algorithms developed in these fields to the processing of large DEDS.

Our approach to distributed systems combines two key elements. The first one concerns the *structure* of a distributed system, which we define as a graph of interacting subsystems (section 2). They are first defined in a very simple setting, assuming no dynamics, which emphasizes the capabilities of this framework. We particularly stress the distributed processing aspects. The second key point concerns distributed *dynamic* systems (section 3). Dynamic systems lead to distributed processings, and in particular to distributed monitoring/diagnosis procedures, provided one adopts the right *semantics* for trajectories. These semantics make explicit use of the concurrency between subsystems, which greatly reduces the combinatorial explosion of the number of possible behaviors, compared to usual semantics. Given these two ingredients, section 4 proceeds to the explicit design of distributed diagnosis algorithms.

A long version of this paper is in preparation (Fabre *et al.*, 01), containing in particular the extension of this work to stochastic distributed systems and proofs of theorems. We only present some salient features.

[1] This work was partially supported by RNRT (French National Research Network in Telecommunications), through project Magda, dedicated to the distributed failure diagnosis of telecommunication networks (see `http://magda.elibel.tm.fr`)

2. DISTRIBUTED SYSTEMS AND ALGORITHMS

The systems we consider operate on sets of variables. Variables are denoted by capital letters: $A, B, V \ldots$ and take values $a, b, v \ldots$ in domains $\mathcal{D}_A, \mathcal{D}_B, \mathcal{D}_V \ldots$ Let $\mathcal{V} = \{V_1, \ldots, V_n\}$ be a set of variables; a *configuration* or *state* $\mathbf{v}$ is a function which assigns to each variable of $\mathcal{V}$ a value of its domain. By abuse of notations, configurations are represented as tuples $\mathbf{v} = (v_1, \ldots, v_n)$, assuming a natural ordering of variables, and their domain is denoted by $\mathcal{D}_\mathcal{V} = \mathcal{D}_{V_1} \times \cdots \times \mathcal{D}_{V_n}$. For $\mathcal{V}' \subset \mathcal{V}$, $\Pi_{\mathcal{V}'}$ is the canonical projection on configurations of $\mathcal{V}'$.

2.1 *Factorization into subsystems*

A system is defined as a pair $\mathcal{S} = (\mathcal{V}, \mathcal{O})$, where $\mathcal{V} = \{V_1, \ldots, V_n\}$ is a set of variables, and $\mathcal{O} \subseteq \mathcal{D}_\mathcal{V}$ is the set of possible *configurations* or *states* of $\mathcal{S}$. $\mathcal{O}$ is a set of legal tuples $(v_1, \ldots, v_n)$ for $\mathcal{S}$, or, equivalently, $\mathcal{O}$ gathers constraints on $\mathcal{S}$.

The *composition* (or *product*) of systems $\mathcal{S}_1 = (\mathcal{V}_1, \mathcal{O}_1)$ and $\mathcal{S}_2 = (\mathcal{V}_2, \mathcal{O}_2)$ is defined as

$$\mathcal{S} = (\mathcal{V}, \mathcal{O}) = \mathcal{S}_1 | \mathcal{S}_2 \Leftrightarrow \begin{cases} \mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \\ \mathcal{O} = \mathcal{O}_1 \wedge \mathcal{O}_2 \end{cases} \quad (1)$$

Hence subsystems $\mathcal{S}_1$ and $\mathcal{S}_2$ "communicate" through their common variables $\mathcal{V}_1 \cap \mathcal{V}_2$, and configurations of $\mathcal{S}$ are obtained by the conjunction [2] of the constraints defining $\mathcal{S}_1$ and $\mathcal{S}_2$.

System $\mathcal{S}$ is said to be a *distributed system* as soon as it can be expressed as the product $\mathcal{S} = \mathcal{S}_1 | \mathcal{S}_2 | \ldots | \mathcal{S}_N$, where $N \geq 2$, $\mathcal{S}_i = (\mathcal{V}_i, \mathcal{O}_i)$ and $\mathcal{V}_i \subseteq \mathcal{V}_j$ never holds for $1 \leq i \neq j \leq N$. In particular, none of the $\mathcal{V}_i$'s is as large as $\mathcal{V}$, hence $\mathcal{S}$ is defined by the conjunction of *local* constraints on *subsets* of variables. A factorization imposes some structure to $\mathcal{S}$ which is often displayed as a hypergraph $\mathcal{G}$: variables are nodes of the graph, and subsystems $\mathcal{S}_i$ appear as hyperedges, i.e. sets of variables (fig. 1).

Given $\mathcal{S} = (\mathcal{V}, \mathcal{O})$ and knowing that $\mathcal{S}$ factorizes on $\mathcal{G}$, given by $\mathcal{V}_1, \ldots, \mathcal{V}_N$, one can easily determine possible sets $\mathcal{O}_i$ by restricting $\mathcal{O}$ to variables in $\mathcal{V}_i$, i.e. $\mathcal{O}_i = \Pi_{\mathcal{V}_i}(\mathcal{O})$, and by extension $\mathcal{S}_i = \Pi_{\mathcal{V}_i}(\mathcal{S})$. Although factorizations of $\mathcal{S}$ on $\mathcal{G}$ may be numerous, the one obtained in that way is canonical in the sense that the $\mathcal{O}_i$'s are minimal: every *local state* in $\mathcal{O}_i$ is the projection of at least one *global state* of $\mathcal{O}$. A factorization property imposes some structure to $\mathcal{S}$: let $\mathcal{S}$ be any system, define $\mathcal{S}_i = \Pi_{\mathcal{V}_i}(\mathcal{S})$, then the product $\mathcal{S}_1 | \ldots | \mathcal{S}_N$ is generally *larger* than $\mathcal{S}$, i.e. allows more configurations.

2.2 *Working with factorized forms*

The major advantage of factorized representations resides in the compact description of a large set of long vectors ($\mathcal{O}$) by small sets of short vectors (the $\mathcal{O}_i$'s). It appears in an obvious manner when subsystems have no interaction: $\mathcal{S} = (\{A\}, \{a, a', a''\}) \,|\, (\{B\}, \{b, b', b''\})$ is a more compact representation than the extended list of possible pairs of values for variables A, B. Therefore, factorized representations are particularly suited to large systems where the size and number of state vectors explodes. Hence *all computations on such systems should be performed on their factorized form.* This is precisely the principle we adopt in the present paper.

The most efficient algorithms handling large systems under their factorized form make an explicit use of the graphical representation of interactions between subsystems (fig. 2), see (Benveniste *et al.*, 95; Pearl, 86) for an overview. We describe the simplest one here, as the prototype of several processings we perform in the sequel. The problem is the following: assume $\mathcal{S}$ factorizes into $\mathcal{S}_1 | \ldots | \mathcal{S}_N$, one wishes to compute the "canonical" sets $\mathcal{O}'_i = \Pi_{\mathcal{V}_i}(\mathcal{O})$ *without computing* $\mathcal{O}$ (which would be too large). We first illustrate the principles on an example.

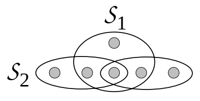

Figure 1. *System chain: $\mathcal{S}_1$ separates $\mathcal{S}_2$ from $\mathcal{S}_3$.*

Example. Assume $\mathcal{S} = \mathcal{S}_1 | \mathcal{S}_2 | \mathcal{S}_3$ and $\mathcal{S}_1$ separates $\mathcal{S}_2$ from $\mathcal{S}_3$, i.e. $\mathcal{V}_{2,3} \subseteq \mathcal{V}_1$ (see fig. 1). Then

$$\begin{aligned} \mathcal{O}'_1 &= \Pi_{\mathcal{V}_1}(\mathcal{O}_1 \wedge \mathcal{O}_2 \wedge \mathcal{O}_3) \\ &= \mathcal{O}_1 \wedge \Pi_{\mathcal{V}_1}(\mathcal{O}_2 \wedge \mathcal{O}_3) \\ &= \mathcal{O}_1 \wedge \Pi_{\mathcal{V}_1}(\mathcal{O}_2) \wedge \Pi_{\mathcal{V}_1}(\mathcal{O}_3) \\ &= \mathcal{O}_1 \wedge \Pi_{\mathcal{V}_{1,2}}(\mathcal{O}_2) \wedge \Pi_{\mathcal{V}_{1,3}}(\mathcal{O}_3) \qquad (2) \end{aligned}$$

where the third equality comes from the fact that $\mathcal{S}_2$ and $\mathcal{S}_3$ have all their common variables inside $\mathcal{V}_1$. Hence $\mathcal{O}'_1$ needs the *messages* $\Pi_{\mathcal{V}_{1,i}}(\mathcal{O}_i)$ from its two neighbors $\mathcal{S}_i$, $i = 2, 3$. In the same way, for $\mathcal{O}'_2$ (and symmetrically for $\mathcal{O}'_3$) one has

$$\begin{aligned} \mathcal{O}'_2 &= \Pi_{\mathcal{V}_2}(\mathcal{O}_1 \wedge \mathcal{O}_2 \wedge \mathcal{O}_3) \\ &= \mathcal{O}_2 \wedge \Pi_{\mathcal{V}_2}(\mathcal{O}_1 \wedge \mathcal{O}_3) \\ &= \mathcal{O}_2 \wedge \Pi_{\mathcal{V}_2}[\mathcal{O}_1 \wedge \Pi_{\mathcal{V}_1}(\mathcal{O}_3)] \\ &= \mathcal{O}_2 \wedge \Pi_{\mathcal{V}_{1,2}}[\mathcal{O}_1 \wedge \Pi_{\mathcal{V}_{1,3}}(\mathcal{O}_3)] \qquad (3) \end{aligned}$$

where the third equality derives from $\mathcal{V}_{2,3} \subseteq \mathcal{V}_1$. The merge equation (3) needs message $\Pi_{\mathcal{V}_{1,2}}[\mathcal{O}_1 \wedge$

[2] We use the "$\wedge$" as a shorthand; a better expression would be $\mathcal{O} = \Pi^{-1}_{\mathcal{V}_1}(\mathcal{O}_1) \cap \Pi^{-1}_{\mathcal{V}_2}(\mathcal{O}_2)$ if $\mathcal{O}$ is considered as a set, or $\mathcal{O} = (\mathcal{O}_1 \circ \Pi_{\mathcal{V}_1}) \cdot (\mathcal{O}_2 \circ \Pi_{\mathcal{V}_2})$ if $\mathcal{O}$ is considered as an indicator function.

$\Pi_{\mathcal{V}_{1,3}}(\mathcal{O}_3)]$ from $\mathcal{S}_1$, which derives from a merge in $\mathcal{S}_1$ of $\mathcal{O}_1$ and another message $\Pi_{\mathcal{V}_{1,3}}(\mathcal{O}_3)$ received from $\mathcal{S}_3$. This *propagation* phenomenon (from $\mathcal{S}_3$ to $\mathcal{S}_2$ through $\mathcal{S}_1$) reveals that sets $\mathcal{O}'_i$ can be obtained by exchanging messages between systems which are direct neighbors, and performing local merges inside each system.

We now formalize the message passing algorithmfor a factorization $\mathcal{S} = \mathcal{S}_1|\ldots|\mathcal{S}_N$ defining the hypergraph $\mathcal{G}$. *In the remaining of the paper, we assume that $\mathcal{G}$ is a* hypertree*, or* tree *for short.*

Definition 1. Let hypergraph $\mathcal{G}$ be defined by edges $\mathcal{V}_1,\ldots,\mathcal{V}_N$, and let $\mathcal{X},\mathcal{Y},\mathcal{Z} \subseteq \mathcal{V} = \mathcal{V}_1 \cup \ldots \cup \mathcal{V}_N$ be sets of nodes. $\mathcal{Z}$ is said to *separate* $\mathcal{X}$ from $\mathcal{Y}$ on $\mathcal{G}$ iff there exists a partition $I \cup J \cup K = \{1,\ldots,N\}$ such that $\mathcal{X} \subseteq \mathcal{V}_{I\cup K}$, $\mathcal{Y} \subseteq \mathcal{V}_{J\cup K}$, $\mathcal{V}_K \subseteq \mathcal{Z}$ and $\mathcal{V}_I \cap \mathcal{V}_J \subseteq \mathcal{V}_K$. $\mathcal{G}$ is said to be a *tree* iff one single $\mathcal{V}_k$ is enough to separate any two $\mathcal{V}_i$ and $\mathcal{V}_j$ (fig. 2).

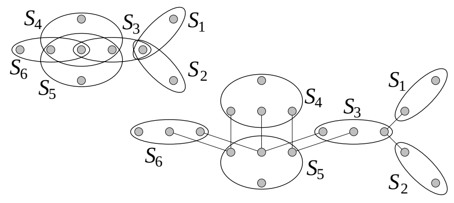

Figure 2. *A hypertree with 6 edges (top) and its tree structure evidenced (bottom) by associating neighbors to each edge. Some nodes are duplicated, and the correspondence between them indicate the neighboring structure.*

To display the tree structure of $\mathcal{G}$, one must determine which systems are direct neighbors. This can be done recursively by cutting off leaves of the tree. An edge $\mathcal{V}_i$ is said to be a *leaf* of $\mathcal{G}$ iff there exists another edge $\mathcal{V}_k$ which separates it from all remaining edges. The edge $\mathcal{V}_i$ is made a direct neighbor of $\mathcal{V}_k$ and is removed from $\mathcal{G}$. The remaining hypergraph keeps the tree property, hence one can repeat the procedure (fig. 2). Notice that the neighboring relation constructed in that way is not unique. In the sequel, the term "tree" will denote such a neighboring structure on $\mathcal{G}$.

Let us denote by $\mathbf{N}(\mathcal{S}_i)$ the set of subsystems $\mathcal{S}_j$ which are direct neighbors of $\mathcal{S}_i$, and let us say there is an oriented link from $\mathcal{S}_i$ to $\mathcal{S}_j$ (and symmetrically) as soon as they are direct neighbors. The algorithm computing sets $\mathcal{O}'_i$ on the tree $\mathcal{G}$ is based on messages "circulating" on these links.

Algorithm $\mathbf{A}_1$

(1) initialization : for each link $\mathcal{S}_i \to \mathcal{S}_j$, define the message

$$\mathcal{M}_{\mathcal{S}_i\to\mathcal{S}_j} = \mathcal{D}_{\mathcal{V}_{i,j}}$$

(2) repeat until no message can be changed

(a) choose one link $\mathcal{S}_i \to \mathcal{S}_j$

(b) update the message on that link by

$$\bar{\mathcal{O}}_i = \mathcal{O}_i \wedge (\bigwedge_{\mathcal{S}_k \in \mathbf{N}(\mathcal{S}_i)\setminus\mathcal{S}_j} \mathcal{M}_{\mathcal{S}_k\to\mathcal{S}_i})$$

$$\mathcal{M}_{\mathcal{S}_i\to\mathcal{S}_j} := \Pi_{\mathcal{V}_{i,j}}(\bar{\mathcal{O}}_i)$$

(3) for each $\mathcal{S}_i$, define $\mathcal{O}'_i$ by

$$\mathcal{O}'_i = \mathcal{O}_i \wedge (\bigwedge_{\mathcal{S}_k \in \mathbf{N}(\mathcal{S}_i)} \mathcal{M}_{\mathcal{S}_k\to\mathcal{S}_i})$$

The proof of convergence extends ideas appearing in the previous example, and of course heavily relies on the separation property. We omit here for lack of space. Nevertheless, let us mention that $\mathbf{A}_1$ remains valid if sets $\mathcal{O}_i$ evolve during the processing and eventually stabilize. We use this property in the sequel.

3. DYNAMIC SYSTEMS AND CONCURRENCY

We now move to dynamic systems with concurrency, viewed as a generelization of Petri nets (Esparza and Romer, 99).

3.1 *Systems and tiles*

Definition 2. A dynamic system $\mathcal{S}$ is a triple $(\mathcal{V},\mathcal{T},\mathcal{I},\Sigma)$ where $\mathcal{V} = \{V_1,\ldots,V_n\}$ is a set of variables, $\mathcal{I} \subseteq \mathcal{D}_{\mathcal{V}}$ is a set of initial values $\mathbf{v} = (v_1,\ldots,v_n)$ of the system, Σ is a set of labels and $\mathcal{T}$ is a set of transitions or *tiles* defined on variables of $\mathcal{V}$ and on the label set Σ.

Definition 3. A tile $\mathbf{t} \in \mathcal{T}$ is a 4-tuple $(\mathcal{V}_{\mathbf{t}}, \mathbf{v}^-_{\mathbf{t}}, \sigma_{\mathbf{t}}, \mathbf{v}^+_{\mathbf{t}})$ where $\mathcal{V}_{\mathbf{t}} \subseteq \mathcal{V}$ is a set of variables, $\mathbf{v}^-_{\mathbf{t}}, \mathbf{v}^+_{\mathbf{t}} \in \mathcal{D}_{\mathcal{V}_{\mathbf{t}}}$ are respectively the pre-state and the post-state of $\mathbf{t}$, and $\sigma_{\mathbf{t}} \in \Sigma$ is a label.

The composition of systems extends the definition of section 2. Let $\mathcal{S}_i = (\mathcal{V}_i,\mathcal{T}_i,\mathcal{I}_i,\Sigma_i)$, $i = 1,2$, then

$$\mathcal{S} = (\mathcal{V},\mathcal{T},\mathcal{I},\Sigma) = \mathcal{S}_1|\mathcal{S}_2 \Leftrightarrow \begin{cases} \mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \\ \mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \\ \mathcal{I} = \mathcal{I}_1 \wedge \mathcal{I}_2 \\ \Sigma = \Sigma_1 \cup \Sigma_2 \end{cases} \quad (4)$$

In the same way, a system $\mathcal{S}$ is said to be a *distributed dynamic system* as soon as it can be expressed as $\mathcal{S} = \mathcal{S}_1|\mathcal{S}_2|\ldots|\mathcal{S}_N$ where $N \geq 2$, $\mathcal{S}_i = (\mathcal{V}_i,\mathcal{T}_i,\mathcal{I}_i,\Sigma_i)$ and $\mathcal{V}_i \subseteq \mathcal{V}_j$ never holds for $1 \leq i \neq j \leq N$. Notice that the factorization of $\mathcal{S}$ limits the size of its tiles (in terms of variables involved).

3.2 *Partial order semantics on trajectories*

Let $\mathbf{t} \in \mathcal{T}$ be a tile and $\mathbf{v}$ a configuration of $\mathcal{S}$, $\mathbf{t}$ is *connectible* to $\mathbf{v}$, denoted by $\mathbf{v}[\mathbf{t}\rangle$, iff $\mathbf{v}_{\mathcal{V}_{\mathbf{t}}} = \mathbf{v}^-_{\mathbf{t}}$. By connecting $\mathbf{t}$ to $\mathbf{v}$, one gets the new configuration $\mathbf{v}'$ defined by $\mathbf{v}'_{\mathcal{V}_\mathbf{t}} = \mathbf{v}^+_\mathbf{t}$ and $\mathbf{v}'_{\mathcal{V}\setminus\mathcal{V}_\mathbf{t}} = \mathbf{v}_{\mathcal{V}\setminus\mathcal{V}_\mathbf{t}}$. By analogy with Petri nets, the connection is denoted by $\mathbf{v}[\mathbf{t}\rangle\mathbf{v}'$. The most straightforward definition of a *run* (or *trajectory*) ω of $\mathcal{S}$ would be as usually a sequence $(\mathbf{v}, \mathbf{t}_1, \ldots, \mathbf{t}_n)$ such that $\mathbf{v} \in \mathcal{I}$, $\mathbf{t}_1, \ldots, \mathbf{t}_n \in \mathcal{T}$, and $\mathbf{v}[t_1\rangle\mathbf{v}_1[t_2\rangle\mathbf{v}_2 \;\cdots\; \mathbf{v}_{n-1}[\mathbf{t}_n\rangle\mathbf{v}_n$. However, the number of possible sequences explodes, because of concurrency, as shown by the next example.

Consider the run $\omega_1 = ((v_1, v_2), \mathbf{t}_1, \ldots, \mathbf{t}_8)$ of fig. 3, where transitions appear as rectangles, the grey part of which identifies impacted variables.

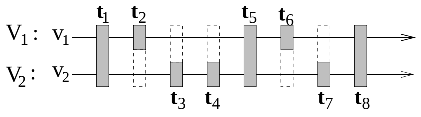

Figure 3. *Graphical representation of a run, as a sequence of transitions.*

By definition of connectibility, a run ω_2 obtained by inverting say $\mathbf{t}_2$ and $\mathbf{t}_3$, or $\mathbf{t}_6$ and $\mathbf{t}_7$, is also valid and finishes with the same configuration. More generally, one can (recursively) exchange in ω_1 any two successive transitions $\mathbf{t}$ and $\mathbf{t}'$ such that $\mathcal{V}_\mathbf{t} \cap \mathcal{V}_{\mathbf{t}'} = \emptyset$. In this case, firing $\mathbf{t}$ and $\mathbf{t}'$ in any order, or even simultaneously, yields the same result : they are said to be *concurrent* in the run. Concurrency is a very important notion in (distributed) systems based on tiles : it identifies areas of independent behaviors in a run. For example, the order of firings between $\mathbf{t}_1$ and $\mathbf{t}_5$ is irrelevant in ω_1. Therefore, one should define define a trajectory not any more as a *sequence* of firings, but as an *equivalence class* of sequences for concurrency. This amounts to considering a run as a *partial order*, or a *puzzle*, of tiles (fig. 4).

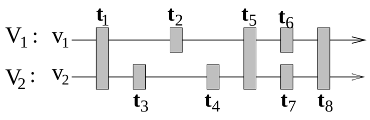

Figure 4. *A run as a partial order of tiles.*

Given a run ω as a sequence of transitions, the partial order $\prec$ representing its puzzle is defined as follows : for any pair $\mathbf{t}, \mathbf{t}'$ of transitions, appearing in this order in ω, $\mathbf{t} \prec \mathbf{t}'$ iff $\mathcal{V}_\mathbf{t} \cap \mathcal{V}_{\mathbf{t}'} \neq \emptyset$. The partial order $\prec$ is then completed by transitivity. In other words, and referring to fig 4, $\prec$ appears to be defined by the total ordering of transitions impacting each variable. One easily checks that the set of linear extensions of $\prec$ define the equivalence class of ω. From now on, we adopt the so-called *true concurrency semantics* (TCS) for trajectories, which doesn't distinguish runs of the same class, or equivalently considers runs as partial orders of firings, i.e. puzzles.

Practical implementation. The definition of $\prec$ suggests a very convenient way to implement the true concurrency semantics in practice. A puzzle ω on variables $\mathcal{V} = \{V_1, \ldots, V_n\}$ is equivalently described as a tuple $(\omega_{V_1}, \ldots, \omega_{V_n})$ where ω_{V_i} is the restriction of ω to variable V_i, i.e. the initial value v_i followed by the *sequence* of transitions concerning V_i. On the example of fig 4, this yields $\omega_{V_1} = (v_1, \mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_5, \mathbf{t}_6, \mathbf{t}_8)$ and $\omega_{V_2} = (v_2, \mathbf{t}_1, \mathbf{t}_3, \mathbf{t}_4, \mathbf{t}_5, \mathbf{t}_7, \mathbf{t}_8)$. Observe that ω can be recovered without ambiguity from the tuple of sequences ω_{V_i} since there exists a unique way to "synchronize" these sequences : if $\mathbf{t}$ concerns variables V_i and V_j, its k^{th} occurence in ω_{V_i} necessarily corresponds to its k^{th} occurence in ω_{V_j}.

Notation. We denote by $\omega \odot \mathbf{t}$ the connectibility of $\mathbf{t}$ to puzzle ω, and by $\omega \cdot \mathbf{t}$ the resulting puzzle after connection.

3.3 *Diagnosis*

The diagnosis problem for DEDS usually amounts to testing a property on the hidden trajectory of the system, given some observable events. For a matter of space, we simplify it here to infering all possible runs matching the observations.

Let us define the visible part $\phi(\omega)$ of run ω by replacing each tile $\mathbf{t}$ in the partial order $\prec$ by its label $\sigma_\mathbf{t}$ (see fig. 4, left). Hence $\phi(\omega) = (\mathcal{L}, \prec)$, where $\mathcal{L}$ is a sequence of labels. The diagnosis problem then amounts to recovering all ω's such that $\phi(\omega)$ coincides [3] with a given $(\mathcal{L}, \prec)$.

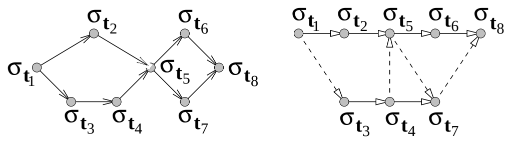

Figure 5. *Left : partial order of labels $\phi(\omega)$ associated to the run ω of fig. 4. Right : an observation of this partial order as a pair of sequences (dotted lines are lost relations).*

In practice, one hardly observes an entire partial order of events, for two reasons. First, in a distributed system $\mathcal{S} = |_i\mathcal{S}_i$, observations may be collected at different locations in the system instead of being centralized. Typically, one can assume that observations (i.e. tile labels) of each component $\mathcal{S}_i$ are collected by a local supervisor. Partial order relations between observations collected on different supervisors cannot be obtained without extra communications. So we assume that such relations are lost. Secondly, in each supervisor, one generally observes a sequence of events, which is a linear extension of this partial order, relying on the so-called *causal observation assumption.* This

[3] Silent tiles are excluded, for simplicity.

assumption states that if $\mathbf{t} \prec \mathbf{t}'$ in ω, one cannot observe $\sigma_{\mathbf{t}'}$ before $\sigma_{\mathbf{t}}$.

We thus reduce observations to a tuple $(\mathcal{L}_1, \ldots, \mathcal{L}_N)$ of sequences of events, one per component $\mathcal{S}_i$: relations $\prec$ between sequences are unknown, and each $\mathcal{L}_i$ is a linear extension of the "true" $(\mathcal{L}, \prec)$, restricted to $\mathcal{S}_i$ (fig. 5, right). Therefore, a run ω is *compatible* with $(\mathcal{L}_1, \ldots, \mathcal{L}_N)$ iff there exists a linear extension of $\phi(\omega)$ that matches some interleaving of sequences $\mathcal{L}_1, \ldots, \mathcal{L}_N$.

4. DISTRIBUTED DIAGNOSIS ALGORITHMS

We now assume that the dynamic system $\mathcal{S}$ factorizes into $\mathcal{S}_1 | \ldots | \mathcal{S}_N$, and has produced some unknown puzzle. Observations $\mathcal{L} = (\mathcal{L}_1, \ldots, \mathcal{L}_N)$, with $\mathcal{L}_i = (\sigma_{i,1}, \ldots, \sigma_{i,T_i})$, have been collected in a distributed manner, with one local supervisor associated to each component $\mathcal{S}_i$. The objective of this section is to solve the diagnosis problem with a distributed algorithm. As already suggested in section 2, the architecture of this distributed algorithm parallels the structure of the system itself: each local supervisor is performing some computations and exchanges messages with its neighbors. Let us stress the fact that the supervisor of component $\mathcal{S}_i$ only knows the local observation sequence $\mathcal{L}_i$, the model $\mathcal{S}_i$, who are its neighbors and what are the common variables with this neighbors. Most building blocks of this distributed procedure were presented in section 2. We focus now on the missing part, which concerns specific aspects of the diagnosis problem see also (Fabre *et al.*, 00).

4.1 *Centralized diagnosis, single player*

Let us first ignore the factorization of $\mathcal{S}$ and assume all transition labels are collected by a global supervisor into the sequence $\mathcal{L} = (\sigma_1, \ldots, \sigma_T)$, still assuming causal observation. Let $\mathcal{O}$ denote the (possibly infinite) set of all trajectories of $\mathcal{S}$, and $\mathcal{O}(t)$ the subset of trajectories which have a prefix matching the first t observations in $\mathcal{L}$. For convenience, we handle $\mathcal{O}(t)$ as this set of prefixes, suggesting that the rest of the trajectory is free. Obviously, $\mathcal{O}(t+1) \subset \mathcal{O}(t)$: collecting more and more observations amounts to recursively filtering (or constraining) $\mathcal{O}$. And prefixes in $\mathcal{O}(t+1)$ are clearly obtained by connecting a new tile $\mathbf{t}$ to elements of $\mathcal{O}(t)$, provided $\sigma_{\mathbf{t}} = \sigma_{t+1}$. We rather adopt in algorithm $\mathcal{A}_2$ an "asynchronous" construction of $\mathcal{O}(T)$, which updates a set $\mathcal{A}$ of prefixes with different lengths. Let $|\omega|$ represent the number of tiles in ω, we define the extension of a single trajectory by

$$\text{Ext}(\omega) = \{\omega \cdot \mathbf{t} \;:\; \mathbf{t} \in \mathcal{T}, \omega \odot \mathbf{t}, \sigma_{\mathbf{t}} = \sigma_{|\omega|+1}\} \quad (5)$$

Algorithm $\mathbf{A}_2$

```
(1) initialization: $\mathcal{A} = \{(\mathbf{v}) : \mathbf{v} \in \mathcal{I}\}$
(2) repeat until $\forall \omega \in \mathcal{A},\ |\omega| = T$
    (a) choose $\omega \in \mathcal{A}$ such that $|\omega| < T$
    (b) replace $\omega$ by its extension
            $\mathcal{A} := \mathcal{A} \setminus \{\omega\} \cup \text{Ext}(\omega)$
(3) define $\mathcal{O}(T)$ as $\mathcal{A}$
```

4.2 *Centralized diagnosis, two players*

In the distributed case, (5) cannot be performed at once since a local supervisor only knows part of the tile set $\mathcal{T}$. So it must collaborate with other supervisors and perform (5) by parts. Let us partition $\mathcal{T}$ into two sets $\mathcal{T}_A, \mathcal{T}_B$ associated to *players* A and B respectively. We consider the standpoint of player A: only tiles of $\mathcal{T}_A$ are observed in $\mathcal{L}$, hence tiles of $\mathcal{T}_B$ are silent for A. We denote by $\text{Ext}_A(\omega)$ operation (5) where tiles are taken in $\mathcal{T}_A$ instead of $\mathcal{T}$, and by $|\omega|_A$ the number of tiles of $\mathcal{T}_A$ in ω. Trajectories already extended by player A are not discarded but stored in a "waiting set" $\mathcal{W}$, in order to allow possible extensions by player B.

Algorithm $\mathbf{A}_3$ (Player A)

```
(1) initialization
        $\mathcal{A} = \{(\mathbf{v}) : \mathbf{v} \in \mathcal{I}\}, \quad \mathcal{W} = \emptyset$
(2) repeat until $\forall \omega \in \mathcal{A},\ |\omega|_A = T$ and player $B$
    has sent "finished"
    (a) on reception of message $\omega \rightsquigarrow \omega'$ from $B$,
            $\mathcal{A} := \mathcal{A} \cup \{\omega'\}$
    (b) otherwise,
        (i) choose $\omega \in \mathcal{A} : |\omega|_A < T$, if any
        (ii) replace $\omega$ by its extension
            $\mathcal{A} := \mathcal{A} \setminus \{\omega\} \cup \text{Ext}_A(\omega)$
            $\mathcal{W} := \mathcal{W} \cup \{\omega\}$
(3) set $\mathcal{O}(T) = \mathcal{A}$
```

Message $\omega \rightsquigarrow \omega'$ means that ω has been extended into ω' by player B. The extension policy of B is unspecified, but B must read prefixes in $\mathcal{W} \cup \mathcal{A}$, extend them by one tile only, and eventually stop. Observe that B does refine the current solution set $\mathcal{A} \cup \mathcal{W}$ by connecting a tile to ω, but this refinement remains "hidden" until the prefix ω is removed form $\mathcal{W}$, i.e. recognized as completely extended. In $\mathbf{A}_3$, removals occur all at once at the end, when B declares its termination: trajectories in $\mathcal{W}$ are finished for both A and B.

4.3 *Distributed diagnoser*

We now gather all elements in view of a distributed diagnosis algorithm. The "players" are naturally the supervisors of components $\mathcal{S}_i$, which refine trajectories in order to satisfy their local observations $\mathcal{L}_i$. But for a matter of efficiency, runs of $\mathcal{S}$ are handled under their factorized form, thanks to the following result.

Theorem 1. For $\mathcal{S}$ factorizing into $\mathcal{S}_1|\ldots|\mathcal{S}_N$, let $\mathcal{O}$ be the set of trajectories of $\mathcal{S}$, and $\mathcal{O}_i$ be the restriction of these trajectories to variables of component $\mathcal{S}_i$: $\mathcal{O}_i = \Pi_{\mathcal{V}_i}(\mathcal{O})$. Let us associate a variable V^H to every variable $V \in \mathcal{V}$, with $\mathcal{D}_{V^H} = \mathcal{D}_V \times \mathcal{T}^*$; a value of V^H denotes a history on V. Let $\bar{\mathcal{S}}$ (resp. $\bar{\mathcal{S}}_i$) be the (non dynamic) system $(\mathcal{V}^H, \mathcal{O})$ (resp. $(\mathcal{V}_i^H, \mathcal{O}_i)$), then one has the factorization $\bar{\mathcal{S}} = \bar{\mathcal{S}}_1|\ldots|\bar{\mathcal{S}}_N$. The same result holds for $\mathcal{O}$ the set of trajectories matching some observation $\mathcal{L} = (\mathcal{L}_1, \ldots, \mathcal{L}_N)$.

The reader is referred to (Fabre *et al.*, 01) for a proof. Thanks to theorem 1, the solution set O can be obtained under its factorized form, the supervisor of $\mathcal{S}_i$ being in charge of constructing $\mathcal{O}_i$. Each $\mathcal{O}_i$ is obtained by recursively refining a current set of projections $\omega_{\mathcal{V}_i}$, either by local extensions in $\mathcal{S}_i$, in order to satisfy observation $\mathcal{L}_i$, or by incorporating constraints due to the neighbors. Let us denote by $|\omega_{\mathcal{V}_i}|_i$ the number of tiles of $\mathcal{T}_i$ in $\omega_{\mathcal{V}_i}$, and by $\mathrm{Ext}_i(\omega_{\mathcal{V}_i})$ the extension with tiles of $\mathcal{T}_i$ matching the next observation in $\mathcal{L}_i$. Just like messages from player B in $\mathbf{A}_3$ section 4.2, every extension of $\omega_{\mathcal{V}_i}$ into $\omega'_{\mathcal{V}_i}$ must be communicated to neighbors. But only changes on shared variables are necessary to a neighbor S_j. Therefore extentions $\omega_{\mathcal{V}_i} \rightsquigarrow \omega'_{\mathcal{V}_i}$ in $\mathcal{S}_i$ generate the update messages $\omega_{\mathcal{V}_{i,j}} \rightsquigarrow \omega'_{\mathcal{V}_{i,j}}$ towards $\mathcal{S}_j$. By symmetry, on reception in $\mathcal{S}_i$ of such a message from $\mathcal{S}_j$, every current solution $\bar{\omega}_{\mathcal{V}_i}$ with the right projection on common variables must accept the update :

$$\mathrm{Upd}(\bar{\omega}_{\mathcal{V}_i}\,;\,\omega_{\mathcal{V}_{i,j}} \to \omega'_{\mathcal{V}_{i,j}}) = \begin{cases} \emptyset \text{ if } \bar{\omega}_{\mathcal{V}_{i,j}} \neq \omega_{\mathcal{V}_{i,j}} \\ \bar{\omega}'_{\mathcal{V}_i} \text{ otherwise} \end{cases}$$
$$\text{where } \bar{\omega}'_{\mathcal{V}_{i,j}} = \omega'_{\mathcal{V}_{i,j}},\ \ \bar{\omega}'_{\mathcal{V}_i \setminus \mathcal{V}_j} = \bar{\omega}_{\mathcal{V}_i \setminus \mathcal{V}_j}$$

By organizing these operations, one gets

Algorithm $\mathbf{A}_4$ (Supervisor of $\mathcal{S}_i$)

```
(1) initialization
        $\mathcal{A}_i = \{(\mathbf{v}_i) \ : \ \mathbf{v}_i \in \mathcal{I}_i\}, \quad \mathcal{W}_i = \emptyset$
(2) repeat until global-end is detected
    (a) on reception of message $\mu$ from $\mathcal{S}_j$
        (i) update the current solution set
                $\mathcal{A}_i := \mathcal{A}_i \cup \mathrm{Upd}(\mathcal{A}_i \cup \mathcal{W}_i\,;\,\mu)$
        (ii) to every other neighbor $\mathcal{S}_k$, propagate message $\mu$ reduced to $\mathcal{V}_{i,k}$
    (b) on decision to extend a local trajectory,
        (i) select $\omega_{\mathcal{V}_i} \in \mathcal{A}_i \ : \ |\omega_{\mathcal{V}_i}|_i < T_i$, if any
        (ii) replace $\omega_{\mathcal{V}_i}$ by its extension
                $\mathcal{A}_i := \mathcal{A}_i \setminus \{\omega_{\mathcal{V}_i}\} \cup \mathrm{Ext}_i(\omega_{\mathcal{V}_i})$
                $\mathcal{W}_i := \mathcal{W}_i \cup \{\omega_{\mathcal{V}_i}\}$
        (iii) send the corresponding updates to neighbors $\mathcal{S}_j,\ j \in \mathbf{N}(i)$
    (c) set the boolean value of local-end
            $\mathtt{local-end} = [\,\forall \omega_{\mathcal{V}_i} \in \mathcal{A}_i,\ |\omega_{\mathcal{V}_i}|_i = T_i\,]$
    (d) if local-end, start detecting global-end
(3) set $\mathcal{O}_i(T_i) = \mathcal{A}_i$
```

`global-end` is characterized by the fact that all supervisors are in the `local-end` state, and no more message is in circulation, which could reactivate a supervisor. This is a classical distributed termination detection.

Observe that $\mathbf{A}_4$ is identical in nature to $\mathbf{A}_1$ provided sets $\mathcal{O}_i$ are allowed to evolve in time. Only the construction of messages differ : in $\mathbf{A}_1$ messages propagate all possible configurations (of shared variables), while in $\mathbf{A}_4$ they only propagate changes in these sets of configurations.

5. CONCLUSION

We have proposed a general framework making the link between dynamic systems and graphical models, which greatly opens the range of affordable processings for large systems. Many variations are possible, including distributed state estimation, maximum likelihood estimation (for stochastic systems, non presented here), etc. The promising turbo algorithms have also natural counterparts in this setting. Finally, let us mention that a prototype based on this framwork is being experimented for failure diagnosis in telecommunication networks.

6. REFERENCES

Benveniste, A., B.C. Levy, E. Fabre and P. Le Guernic (95). A calculus of stochastic systems: specification, simulation, and hidden state estimation. *Theo. Comp. Sci.* (152), 171–217.

Esparza, J. and S. Romer (99). An unfolding algorithm for synchronous products of transition systems. In: *proc. CONCUR'99.* Vol. 1664 of *LNCS.*

Fabre, E., A. Benveniste and C. Jard (01). Distributed algorithms for bayesian networks of dynamic systems. *in preparation.*

Fabre, E., A. Benveniste, C. Jard, L. Ricker and M. Smith (00). Distributed state reconstruction for discrete event systems. In: *proc. of 39th Conf. on Detection and Control, Sydney.*

Pearl, J. (86). Fusion, propagation, and structuring in belief networks. *Art. Intel.* **29**, 241–288.