Monitoring Distributed Systems with Distributed Algorithms [1]

Eric Fabre, Vincent Pigourier[2]

Abstract

This paper proposes a framework to process large distributed systems by parts, through distributed algorithms. We consider distributed (discrete event) systems as the combination of elementary components. Each component defines dynamics on several state variables, and the composition is simply defined by sharing variables. The compound system is asynchronous: each component evolves with its own clock, and exchanges information with its neighbors by means of the shared variables. An interaction graph can be associated to such a compound system: two components are neighbors of each other as soon as they share one (or more) variables. This structure is reminiscent of Bayesian networks, or Markov random fields, which use a graph to display dependencies between random variables. The parallel can actually be pushed quite far. In this paper we show that a large family of modular algorithms developped for Markov fields, in order to solve problems like maximum likelihood state estimation, can be translated into distributed algorithms to monitor large distributed dynamic systems.

1 Introduction

This paper relies on other results published in the proceedings of CDC'02 as [1]. Notations are not redefined.

The composition principle is a natural tool to build large complex systems out of elementary functions or local specifications. However, although its use is widely spread as a design procedure for dynamic systems, there has been little work trying to make use of the modular nature of a system to design dedicated monitoring algorithms [11, 9, 13]. This is the scope of the present paper, motivated by the distributed monitoring of telecommunication networks. A compound system can be represented as an interaction graph of components (fig. 1 bottom); our aim is to design a parallel monitoring architecture (fig. 1 top), composed of local supervisors, one per component, which coordinate their actions. This strategy offers many advantages, among which a convenient way to update the monitoring algorithms as the supervised system evoles, but, more importantly, a solution to the state explosion phenomenon occuring in large systems.

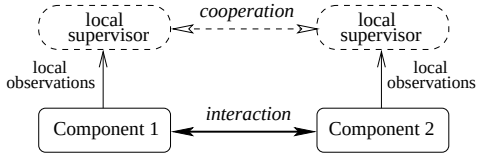

Figure 1: *A distributed system (bottom) and a modular monitoring architecture (for example for state estimation).*

The paper is organized as follows. Section 2 briefly presents existing algorithms on *static* systems, with an appropriate reformulation. It explains how the factorization of a large system into a product of components opens the way to modular algorithms. A simple reduction problem is stated and solved in that way. Its solution is a paradigm of most modular processings. All results of this section extend to a stochastic framework, not presented for lack of space (see [2]). Section 3 extends this framework to distributed *dynamic* systems, and recalls a factorization property on trajectories of theses systems [1]. This property forms the basis of distributed diagnosis algorithms presented in section 4.

2 Modular processings for static systems

2.1 Definition of compound systems

A (static) system is defined as a pair $\mathcal{S} = (\mathcal{V}, \mathcal{O})$, where $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$ is a set of variables, and $\mathcal{O} \subseteq \mathcal{D}_{V_1} \times \ldots \mathcal{D}_{V_n}$ is the set of possible *states* of $\mathcal{S}$. Hence $\mathcal{O}$ is a set of legal tuples $(v_1, \ldots, v_n)$ for $\mathcal{S}$; in other words, $\mathcal{S}$ is defined by *constraints* on the value of its variables.

Let $\mathcal{S}_1 = (\mathcal{V}_1, \mathcal{O}_1)$ and $\mathcal{S}_2 = (\mathcal{V}_2, \mathcal{O}_2)$ be two such systems, we define their *composition* (or *product*) by

$$\mathcal{S} = (\mathcal{V}, \mathcal{O}) = \mathcal{S}_1 | \mathcal{S}_2 \quad \Leftrightarrow \quad \left\{ \begin{array}{rcl} \mathcal{V} & = & \mathcal{V}_1 \cup \mathcal{V}_2 \\ \mathcal{O} & = & \mathcal{O}_1 \wedge \mathcal{O}_2 \end{array} \right. \quad (1)$$

Hence the two subsystems $\mathcal{S}_1$ and $\mathcal{S}_2$ "communicate" through the variables $\mathcal{V}_1 \cap \mathcal{V}_2$ they have in common,

[1] This work is partially supported by RNRT (National Research Network in Telecommunication) through the Magda project (Modelling and Learning for a Distributed Management of Alarms), see `http://magda.elibel.tm.fr/`

[2] IRISA/INRIA, Campus de Beaulieu, 35042 Rennes cedex, France; `Name.Surname@irisa.fr`

and states of $\mathcal{S}$ are obtained by the conjunction of the constraints defining $\mathcal{S}_1$ and $\mathcal{S}_2$. Specifically, $\mathcal{O} = \mathcal{O}_1 \wedge \mathcal{O}_2$ is a shorthand for $\mathcal{O} = (\Pi_{\mathcal{V}_1})^{-1}(\mathcal{O}_1) \cap (\Pi_{\mathcal{V}_2})^{-1}(\mathcal{O}_2)$, where $\Pi_{\mathcal{V}_i}$ is the canonical projection erasing variables outside $\mathcal{V}_i$.

A system $\mathcal{S}$ is said to be a *distributed* (or *compound*) *system* as soon as it can be expressed as

$$\mathcal{S} \quad = \quad \mathcal{S}_1 | \mathcal{S}_2 | \dots | \mathcal{S}_N \tag{2}$$

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. Factorization 2 imposes some structure to $\mathcal{S}$ which is often displayed by means of a hypergraph $\mathcal{G}$: variables are represented as nodes of the graph, and subsystems $\mathcal{S}_i$ appear as hyperedges, i.e. sets of variables (see fig. 2, or fig. 3 left).

2.2 Relations between $\wedge$ and Π

To define compound systems, and handle them with modular algorithms as we do in the sequel, one doesn't need to specify the exact nature of composition $\wedge$ and of projection operators $\Pi_{\mathcal{V}}$. The important properties are summarized in the following axioms, which form the basis of all subsequent developments.

Let us consider an abstract notion of system, generically denoted by $\mathcal{O}$. Systems are provided with two basic operations: a *composition* law, and a set of *reduction* operators. The composition of systems, denoted by $\mathcal{O} = \mathcal{O}_1 \wedge \mathcal{O}_2$, is commutative and associative. The family of reduction operators $\{\Pi_{\mathcal{V}}, \mathcal{V} \subseteq \mathcal{V}_{\max}\}$ is indexed by sets of variables; reductions operate on a single system: $\mathcal{O}' = \Pi_{\mathcal{V}}(\mathcal{O})$. We assume composition and reduction satisfy the following axioms.

$$\forall\, \mathcal{V}_1, \mathcal{V}_2 \subseteq \mathcal{V}_{\max}, \quad \Pi_{\mathcal{V}_1} \circ \Pi_{\mathcal{V}_2} \;=\; \Pi_{\mathcal{V}_1 \cap \mathcal{V}_2} \tag{3}$$

which expresses that reduction operators are actually projections.

$$\forall \mathcal{O},\; \exists \mathcal{V} \subseteq \mathcal{V}_{\max},\;\; \Pi_{\mathcal{V}}(\mathcal{O}) \;=\; \mathcal{O} \tag{4}$$

$\mathcal{O}$ is said to *operate on variables of* $\mathcal{V}$, whence notation $\mathcal{S} = (\mathcal{V}, \mathcal{O})$ for a system. Axiom (3) induces the existence of a smaller set of variables on which $\mathcal{O}$ operates. Finally, the central axiom concerns the relation between composition and reduction. Let $\mathcal{S}_1 = (\mathcal{V}_1, \mathcal{O}_1)$, $\mathcal{S}_2 = (\mathcal{V}_2, \mathcal{O}_2)$ be two systems, then

$$\forall \mathcal{V}_3 \supseteq \mathcal{V}_1 \cap \mathcal{V}_2,\; \Pi_{\mathcal{V}_3}(\mathcal{O}_1 \wedge \mathcal{O}_2) = \Pi_{\mathcal{V}_3}(\mathcal{O}_1) \wedge \Pi_{\mathcal{V}_3}(\mathcal{O}_2) \tag{5}$$

This is a kind of conditional distributivity property of Π with respect to $\wedge$. It expresses that the interaction between systems $\mathcal{O}_1$ and $\mathcal{O}_2$ is completely captured by their shared variables $\mathcal{V}_1 \cap \mathcal{V}_2$. With (3,4,5), one can readily show that $\mathcal{O}_1 \wedge \mathcal{O}_2$ operates on variables $\mathcal{V}_1 \cup \mathcal{V}_2$, whence the definition of $|$ in (1). But the interesting points are in the following equations, which derive from the above axioms and are instrumental in building modular algorithms.

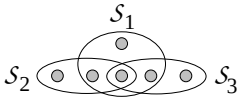

Figure 2: *A chain of systems: $\mathcal{S}_1$ separates $\mathcal{S}_2$ from $\mathcal{S}_3$.*

Consider $\mathcal{O} = \mathcal{O}_1 \wedge \mathcal{O}_2$, where $\mathcal{O}_i$ operates on $\mathcal{V}_i$, then

$$\Pi_{\mathcal{V}_1}(\mathcal{O}) \quad = \quad \mathcal{O}_1 \wedge \Pi_{\mathcal{V}_1 \cap \mathcal{V}_2}(\mathcal{O}_2) \tag{6}$$

This equation expresses how $\mathcal{O}_1$ is modified when composed with $\mathcal{O}_2$. Specifically, the influence of $\mathcal{O}_2$ on $\mathcal{O}_1$ reduces to $\Pi_{\mathcal{V}_1 \cap \mathcal{V}_2}(\mathcal{O}_2)$. Next, assume $\mathcal{O} = \mathcal{O}_1 \wedge \mathcal{O}_2 \wedge \mathcal{O}_3$, where $\mathcal{O}_1$ separates $\mathcal{O}_2$ from $\mathcal{O}_3$, i.e. variables of $\mathcal{V}_2 \cap \mathcal{V}_3$ are all contained in$\mathcal{V}_1$. In other words, systems organize in a *chain*, as on fig. 2. Then

$$\Pi_{\mathcal{V}_1}(\mathcal{O}) \quad = \quad \mathcal{O}_1 \wedge \Pi_{\mathcal{V}_1 \cap \mathcal{V}_2}(\mathcal{O}_2) \wedge \Pi_{\mathcal{V}_1 \cap \mathcal{V}_3}(\mathcal{O}_3) \tag{7}$$

i.e. $\mathcal{O}_2$ and $\mathcal{O}_3$ have independent influences on $\mathcal{O}_1$. Moreover, one has

$$\Pi_{\mathcal{V}_2}(\mathcal{O}) \quad = \quad \mathcal{O}_2 \wedge \Pi_{\mathcal{V}_2 \cap \mathcal{V}_1}[\mathcal{O}_1 \wedge \Pi_{\mathcal{V}_1 \cap \mathcal{V}_3}(\mathcal{O}_3)] \tag{8}$$

which displays a *propagation phenomenon* of $\mathcal{O}_3$ to $\mathcal{O}_2$ through $\mathcal{O}_1$, reminiscent, for example, of recursive equations in Kalman filtering[1]. These equations suggest that $\Pi_{\mathcal{V}_i}(\mathcal{O})$ can be computed by exchanging messages between neighbor systems (i.e. systems sharing variables). We formalize this point in the next section.

Before, let us introduce a last property, which will not be necessary to design modular algorithms. Composition is said to be *involutive* iff

$$\forall \mathcal{O},\, \forall \mathcal{V},\;\; \mathcal{O} \wedge \Pi_{\mathcal{V}}(\mathcal{O}) \;=\; \mathcal{O} \tag{9}$$

i.e. composing a system with "part of itself" doesn't change that system. Involutivity induces the interesting following consequence:

$$\mathcal{O} = \mathcal{O}_1 \wedge \mathcal{O}_2 \quad \Rightarrow \quad \mathcal{O} = \Pi_{\mathcal{V}_1}(\mathcal{O}) \wedge \Pi_{\mathcal{V}_2}(\mathcal{O}) \tag{10}$$

which generalizes to an arbitrary number of factors. In other words, if $\mathcal{S}$ *factorizes* as $\mathcal{S} = \mathcal{S}_1 | \dots | \mathcal{S}_N$, then reduced systems $\mathcal{S}'_i \triangleq \Pi_{\mathcal{V}_i}(\mathcal{S})$ give another factorization of $\mathcal{S}$, which is canonical.

2.3 Computation of reduced components

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 (sets $\mathcal{O}_i$).

[1] By the way, Kalman filtering can be expressed in the present framework, with $\wedge$ and Π operations adapted to stochastic systems.

Hence *all computations concerning such systems should be performed on their factorized form.* This is precisely the principle we adopt in the present paper, under the name "modular algorithms." We illustrate the principle on a typical example, representative of many other processings: given $\mathcal{S} = \mathcal{S}_1 | \dots | \mathcal{S}_N$, we want to compute the canonical factorization of $\mathcal{S}$, i.e. elements $\mathcal{S}'_i \triangleq \Pi_{\mathcal{V}_i}(\mathcal{S})$, *without computing* $\mathcal{S}$, which would be a too large object.

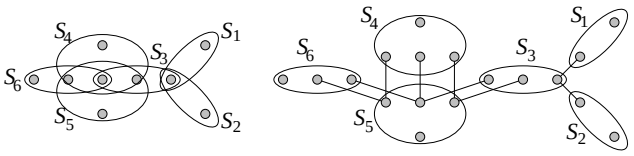

Figure 3: *A hypertree with 6 edges (left). Its tree structure is evidenced on the righthand side by associating neighbors to each edge/system. Some nodes are duplicated, and the correspondence between them indicate the neighboring structure.*

The most efficicent algorithms to do so rely on the graphical representation of interactions between components $\mathcal{S}_i$, i.e. on the hypergraph $\mathcal{G}$ defined by sets $\mathcal{V}_i$. In terms of Markov random fields, the interaction graph evidences conditional independence relations between systems, which can be exploited in estimation procedures. We exactly rely on this principle. In this paper, we focus on systems having *a tree structured interaction graph*: an example is given on fig. 3 (left); the tree structure is evidenced by associating "direct neighbors" to each system $\mathcal{S}_i$ (fig. 3 right). See [1] for details.

Let us denote by $\mathbf{N}(i)$ the indexes of subsystems which are direct neighbors of $\mathcal{S}_i$ on the tree, 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. A message $\mathcal{M}_{i,j}$ brings to $\mathcal{S}_j$ a summary of constraints about $\mathcal{V}_j$ found in the subtree "behind $\mathcal{S}_i$" from the standpoint of $\mathcal{S}_j$.

Algorithm $\mathbf{A}_1$

1. initialization: for each link $i \to j$, define message $\mathcal{M}_{i,j} := \mathcal{D}_{\mathcal{V}_i \cap \mathcal{V}_j}$
2. until no more message can be changed, choose one link $i \to j$, and update its message by

$$\tilde{\mathcal{O}}_i := \mathcal{O}_i \wedge \Big(\bigwedge_{k \in \mathbf{N}(i) \setminus j} \mathcal{M}_{k,i} \Big) \quad (11)$$

$$\mathcal{M}_{i,j} := \Pi_{\mathcal{V}_i \cap \mathcal{V}_j}(\tilde{\mathcal{O}}_i) \quad (12)$$

3. termination: for each $\mathcal{S}_i$, define $\mathcal{O}'_i$ by

$$\mathcal{O}'_i := \mathcal{O}_i \wedge \Big(\bigwedge_{k \in \mathbf{N}(i)} \mathcal{M}_{k,i} \Big) \quad (13)$$

Theorem 1 $\mathbf{A}_1$ *converges in bounded time towards a unique stable point for messages, which yield the desired reduced systems* $\mathcal{S}'_i$. *Convergence is guaranteed whatever the ordering of updates.* $\mathbf{A}_1$ *is insensitive to the initialization values of messages.*

The proof is given in [2]. Basically, (11) merges information coming from various branches at $\mathcal{S}_i$, as in (7), and (12) propagates it towards $\mathcal{S}_j$ as in (6). Since messages accumulate constraints coming from neighbor systems, $\tilde{\mathcal{O}}_i$ finally converges to $\Pi_{\mathcal{V}_i}(\bigwedge_{l \in L_{i \to j}} \mathcal{O}_l)$ where $\mathcal{L}_{i \to j}$ are indexes of systems located in the branch behind $\mathcal{S}_i$ from the standpoint of $\mathcal{S}_j$.

The simple structure of $\mathbf{A}_1$ is at the core of many processings for distributed systems. In a stochastic framework, for example, with appropriate definitions of $\wedge$ and Π, it allows the computations of the most likely state $\mathbf{v}^*$ of $\mathcal{S}$ given observed values on some variables of $\mathcal{V}$. This most likely state is obtained through its projections $\mathbf{v}^*_i$ in each component $\mathcal{S}_i$, since $\mathbf{A}_1$ only provides *local views* of $\mathcal{S}$.

2.4 Extension to changing systems

One can slightly enlarge the validity domain of $\mathbf{A}_1$ to capture some dynamic features. Assume that constraints carried by components $\mathcal{S}_i$ evolve in time: $\mathcal{S}_i(t_i) = (\mathcal{V}_i, \mathcal{O}_i(t_i))$, where the clock $t_i \in \mathbb{N}$ is *local* to $\mathcal{S}_i$, and is bounded: $0 \leq t_i \leq T_i$. We thus represent "time" in the global system $\mathcal{S}$ by a clock vector $(t_1, \dots, t_N)$. Let us build algorithm $\mathbf{A}_2$ from $\mathbf{A}_1$ by the following changes

i. initialize $(t_1, \dots, t_N)$ to $(0, 0, \dots, 0)$

ii. make equations (11) and (13) depend on local time, by changing $\mathcal{O}_i$ into $\mathcal{O}_i(t_i)$,

iii. add a choice in the while loop at step 2 between updating a message or making time t_i evolve $(t_i := t_i + 1)$ in some system $\mathcal{S}_i$, provided $t_i < T_i$,

iv. replace the stop condition at step 2 by "all clocks t_i have reached their final value T_i, and no message can be changed."

Change *iii* expresses that a system $\mathcal{S}_i$ may change while $\mathbf{A}_2$ is running. Like $\mathbf{A}_1$, $\mathbf{A}_2$ converges and yields the $\mathcal{S}'_i \triangleq \Pi_{\mathcal{V}_i}[\bigwedge_j \mathcal{S}_j(T_j)]$, i.e. the same result as $\mathbf{A}_1$ applied to final systems $\mathcal{S}_i(T_i)$. This framework is particularly interesting when systems $\mathcal{S}_i(t_i)$ refine their local state sets as they collect observations.

3 Distributed dynamic systems

3.1 Distributed dynamic systems

This section extends the previous framework by replacing static components $\mathcal{S}_i$ by dynamic systems: local variables become state variables, and a component $\mathcal{S}_i$ is a finite state machine on these variables [8, 7]. More precisely, recalling definitions of [1]

Definition 1 *A dynamic system $\mathcal{S}$ is a triple $(\mathcal{V}, \mathcal{I}, \mathcal{T})$ where $\mathcal{V}$ is a variable set, $\mathcal{I} \subseteq \mathcal{D}_{\mathcal{V}}$ a set of initial states, and $\mathcal{T}$ a finite set of transitions or* tiles *defined on variables of $\mathcal{V}$ and on a label set Σ. 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 variable set, $\mathbf{v}_{\mathbf{t}}^-, \mathbf{v}_{\mathbf{t}}^+ \in \mathcal{D}_{\mathcal{V}_{\mathbf{t}}}$ are respectively the pre-state and the post-state of $\mathbf{t}$ (i.e. tuples of values, one for each element of $\mathcal{V}_{\mathbf{t}}$), and $\sigma_{\mathbf{t}} \in \Sigma$ is a label.*

The composition of dynamic systems extends the definition of section 2. Let $\mathcal{S}_i = (\mathcal{V}_i, \mathcal{I}_i, \mathcal{T}_i)$, $i = 1, 2$, then $\mathcal{S} = \mathcal{S}_1 | \mathcal{S}_2$ is defined by

$$\mathcal{S} = (\mathcal{V}, \mathcal{I}, \mathcal{T}) = \mathcal{S}_1 | \mathcal{S}_2 \quad \Leftrightarrow \quad \left\{ \begin{array}{lcl} \mathcal{V} & = & \mathcal{V}_1 \cup \mathcal{V}_2 \\ \mathcal{I} & = & \mathcal{I}_1 \wedge \mathcal{I}_2 \\ \mathcal{T} & = & \mathcal{T}_1 \cup \mathcal{T}_2 \end{array} \right. \quad (14)$$

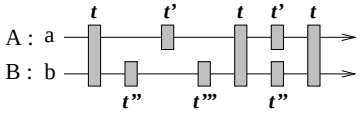

Figure 4: *Partial order of events with several occurences of the same tile.*

Runs of such systems are obtained by successively connecting tiles to a state $\mathbf{v}$ of $\mathcal{S}$. Tile $\mathbf{t}$ is firable at state $\mathbf{v}$ iff $\Pi_{\mathcal{V}_{\mathbf{t}}}(\mathbf{v}) = \mathbf{v}_{\mathbf{t}}^-$. By firing $\mathbf{t}$, only variables $\mathcal{V}_{\mathbf{t}}$ are changed in $\mathbf{v}$: they take the new value $\mathbf{v}_{\mathbf{t}}^+$, which defines the next state $\mathbf{v}'$ of $\mathcal{S}$. It is not convenient to represent runs as sequences of events, as is the case usually: the consecutive firings of tiles $\mathbf{t}$ and $\mathbf{t}'$ may impact different variable sets ($\mathcal{V}_{\mathbf{t}} \cap \mathcal{V}_{\mathbf{t}'} = \emptyset$). These two firings (or events) are said to be *concurrent*: the order in which they occur is meaningless to $\mathcal{S}$. To capture this phenomenon, we adopt the so-called true concurrency semantics (TCS) to describe trajectories of $\mathcal{S}$, which represent runs as partial orders of events. Fig. 4 depicts an example of a run in these semantics: the first firings of $\mathbf{t}'$ and $\mathbf{t}''$ are concurrent (these two events can be inverted). By contrast, the first firing of $\mathbf{t}''$ is not concurrent with the second firing of $\mathbf{t}'$ since the former is followed by a firing of $\mathbf{t}$ which preceeds the latter.

As detailed in [1], the representation of a run ω in the TCS is given by a tuple $\omega = (\omega_V, V \in \mathcal{V})$ of traces ω_V, where ω_V is the initial value of V followed by the sequence of transitions that have operated on V. For the example of fig. 4, this yields

$$(\omega_A, \omega_B) \quad \text{where} \quad \left\{ \begin{array}{l} \omega_A = (a, \mathbf{t}, \mathbf{t}', \mathbf{t}, \mathbf{t}', \mathbf{t}) \\ \omega_B = (b, \mathbf{t}, \mathbf{t}'', \mathbf{t}''', \mathbf{t}, \mathbf{t}'', \mathbf{t}) \end{array} \right. \quad (15)$$

Observe that no information is lost; the kth firing of $\mathbf{t}$ in ω_A necessarily coincides with the kth firing of $\mathbf{t}$ in ω_B because $\mathcal{V}_{\mathbf{t}} = \{A, B\}$, which identifies shared events. The modular processings we present later actually operate on *extended runs* $(\omega, \prec)$, where $\prec$ is a partial order relation on events of ω extending the partial order of ω alone.

3.2 Factorization property on runs

To a dynamic system $\mathcal{S} = (\mathcal{V}, \mathcal{I}, \mathcal{T})$ we associate a static system $\bar{\mathcal{S}} = (\bar{\mathcal{V}}, \bar{\mathcal{O}})$ describing its possible trajectories in the true concurrency semantics. Variables of $\bar{\mathcal{S}}$ are defined by $\bar{\mathcal{V}} = \{\Omega_V : V \in \mathcal{V}\}$, where Ω_V takes values in $\mathcal{D}_{\Omega_V} = \mathcal{D}_V \times (\mathcal{T}_V)^*$, with $\mathcal{T}_V = \{\mathbf{t} \in \mathcal{T}, V \in \mathcal{V}_{\mathbf{t}}\}$. So a value ω_V of Ω_V is a possible history of variable V alone (as ω_A and ω_B above). The state set $\bar{\mathcal{O}}$ of $\bar{\mathcal{S}}$ is composed of all (finite size) runs of $\mathcal{S}$, handled as pairs $(\omega, \prec)$ where ω is a tuple $(\omega_V, V \in \mathcal{V})$ and $\prec$ is exactly the partial order described by ω alone.

Assuming $\mathcal{S} = (\mathcal{V}, \mathcal{I}, \mathcal{T}) = \mathcal{S}_1 | \ldots | \mathcal{S}_N$, we define components $\bar{\mathcal{S}}_i = (\bar{\mathcal{V}}_i, \bar{\mathcal{O}}_i)$ in a similar manner, except that we first extend the transition sets of each $\mathcal{S}_i$. Specifically, $\bar{\mathcal{V}}_i = \{\Omega_V : V \in \mathcal{V}_i\}$, but $\bar{\mathcal{O}}_i$ is composed of trajectories $(\Omega_{\mathcal{V}_i}, \prec_i)$ of the extended system $\mathcal{S}_i^e = (\mathcal{V}_i, \mathcal{I}_i, \mathcal{T}_i^e)$. The extended transition set is given by $\mathcal{T}_i^e = \{\mathbf{t} \in \mathcal{T} : \mathcal{V}_{\mathbf{t}} \cap \mathcal{V}_i \neq \emptyset\}$, i.e. it incorporates all transitions of other components $\mathcal{S}_j$ which have an action on variables of $\mathcal{S}_i$. Trajectories of $\mathcal{S}_i^e$ are built by connecting tiles as if they were "truncated" to variables of $\mathcal{V}_i$ (see fig. 7 in [1]).

The composition of systems $\bar{\mathcal{S}}_i$ is best defined through the composition of local runs. Let $(\omega_1, \prec_1), (\omega_2, \prec_2)$ be two local runs, their composition[2] is given by

$$(\omega_1, \prec_1) \bar{\wedge} (\omega_2, \prec_2) = \mathcal{P}[(\omega_1 \wedge \omega_2, \prec_1 \wedge \prec_2)] \quad (16)$$

where $\omega_1 \wedge \omega_2$ glues the tuples provided they coincide on shared variables (as in section 2), partial order $\prec_1 \wedge \prec_2$ is the smaller extension of both $\prec_1$ and $\prec_2$ on events of $\omega_1 \wedge \omega_2$, and operator $\mathcal{P}$ discards pairs $(\omega, \prec)$ not corresponding to a valid partial order of events.

In [1], the following important result is stated:

$$\mathcal{S} = \mathcal{S}_1 | \ldots | \mathcal{S}_N \quad \Rightarrow \quad \bar{\mathcal{S}} = \bar{\mathcal{S}}_1 | \ldots | \bar{\mathcal{S}}_N \quad (17)$$

It expresses that the trajectory set of $\mathcal{S}$ factorizes as a product of local trajectory sets. As a consequence, one can hope handling $\bar{\mathcal{S}}$ or its subproducts by means of local computations, as in section 2.

[2] This equation is slightly abusive since the right hand side term may be empty. One must read it in the sense of singleton composition.

To do so, one needs a notion of projection on runs $(\omega, \prec)$. $\bar{\Pi}_{\mathcal{V}'}(\omega, \prec)$ is defined as the pair $(\omega', \prec')$, where $\omega' = (\omega_V : V \in \mathcal{V}')$ is the restriction of ω to variables of $\mathcal{V}'$, and $\prec'$ the restriction of $\prec$ to events appearing in ω'. It can be shown that $\bar{\wedge}$ and operators $\bar{\Pi}_{\mathcal{V}}$ satisfy axioms (3,4,5,9).

4 Distributed diagnosis algorithms

We formulate the diagnosis problem in a very simple way: system $\mathcal{S}$ runs, and an observer collects transition labels of events occuring in $\mathcal{S}$. Labels are collected as a sequence $\mathcal{L} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_n)$ which is a linear extension of the partial order of labels produced by $\mathcal{S}$. The goal is to recover all hidden trajectories of $\mathcal{S}$ that could have produced $\mathcal{L}$. We focus on the special case where observations are collected at different locations on $\mathcal{S}$. Specifically, for $\mathcal{S} = \mathcal{S}_1 | \ldots | \mathcal{S}_N$, we assume a local observer collects labels produced by component $\mathcal{S}_i$. Observations thus take the form of a tuple $(\mathcal{L}_1, \ldots, \mathcal{L}_N)$ of label sequences, $\mathcal{L}_i = (\mathbf{o}_{i,1}, \ldots, \mathbf{o}_{i,T_i})$, $1 \leq i \leq N$. Due to the distributed collection of observations, the interleaving of these sequences is lost.

4.1 Asynchronous centralized diagnosis

Let us first ignore the factorization of $\mathcal{S}$ and assume tile labels are collected as a single sequence $\mathcal{L}$. Trajectories explaining $\mathcal{L}$ can be obtained by a guided simulation principle. Let $\mathcal{A}$ denote an "active set" of runs of $\mathcal{S}$ matching part of observations. $\mathcal{A}$ is initialized to inital states of $\mathcal{S}$: $\mathcal{A} = \{(\mathbf{v}) : \mathbf{v} \in \mathcal{I}\}$. Let us then recursively update $\mathcal{A}$ by

$$\mathcal{A} \quad := \quad (\mathcal{A} \setminus \{\omega\}) \cup \mathrm{Ext}(\omega) \tag{18}$$

where ω is any run of $\mathcal{A}$. The extension $\mathrm{Ext}(\omega)$ of a run amounts to connecting all possible tiles matching the next observation in $\mathcal{L}$:

$$\mathrm{Ext}(\omega) \quad \triangleq \quad \{\omega \cdot \mathbf{t} \ : \ \mathbf{t} \in \mathcal{T}, \omega \odot \mathbf{t}, \sigma_{\mathbf{t}} = \mathbf{o}_{|\omega|+1}\} \tag{19}$$

$\omega \odot \mathbf{t}$ and $\omega \cdot \mathbf{t}$ respectively denote the connectibility and the connection of tile $\mathbf{t}$ to run ω, and $|\omega|$ is the number of tiles in ω. Obviously, the recursion (18) has a unique stable point, which is the solution to the diagnosis problem.

4.2 Distributed diagnosis

For a distributed system and distributed observations, the centralized approach above is inappropriate: it handles global runs of $\mathcal{S}$, and thus is faced to state explosion. Fortunately, if runs of $\mathcal{S}$ are restricted to those matching observations $(\mathcal{L}_1, \ldots, \mathcal{L}_N)$, the factorization result (17) still holds. Therefore one can aim at building the solution set to the diagnosis problem in a factorized form.

To do so, we propose a distributed supervision architecture, with a local supervisor on top of each component $\mathcal{S}_i$, knowing obervations $\mathcal{L}_i$, and in charge of building the projection on variables $\mathcal{V}_i$ of the solution set, i.e. runs of $\mathcal{S}$ explaining all observations. Local supervisors must of course coordinate their work to do so.

The structure of this procedure can be outlined simply. Assume the local supervisor of $\mathcal{S}_i$ can directly select in $\bar{\mathcal{O}}_i$ local runs that match its local observations $\mathcal{L}_i$. Let us denote this set by $\mathcal{A}_i$. Then algorithm $\mathbf{A}_1$ provides the appropriate strategy to gather all these local constraints and yield local projections of the solution set: $\mathcal{A}'_i = \bar{\Pi}_{\mathcal{V}_i}(\mathcal{A}_1 \bar{\wedge} \ldots \bar{\wedge} \mathcal{A}_N)$. Unfortunately, sets $\mathcal{A}_i$ are not directly accessible: these sets of (local) runs must be *built* recursively, by successive extensions as in (18). Each time a run ω_i of $\mathcal{A}_i$ is extended, to match one more local observation, a new constraint is incorporated to $\mathcal{A}_i$. Hence we are exactly in the setting of algorithm $\mathbf{A}_2$ where local state sets change in time.

Let us denote by $|\bar{\omega}_i|_i$ the number of tiles of $\mathcal{T}_i$ in run $\bar{\omega}_i$, and define local extensions as

$$\begin{aligned} \mathrm{Ext}_i(\bar{\omega}_i) \quad &= \quad \{\, \bar{\omega}_i \cdot \mathbf{t}_i \ : \ \mathbf{t}_i \in \mathcal{T}_i, \bar{\omega}_i \odot \mathbf{t}_i, \\ &\qquad \sigma_{\mathbf{t}_i} = \mathbf{o}_{i,k+1}, \, k = |\bar{\omega}_i|_i\} \qquad (20) \\ \mathrm{Ext}^e_i(\bar{\omega}_i) \quad &= \quad \{\, \bar{\omega}_i \cdot \mathbf{t} \ : \ \mathbf{t} \in \mathcal{T}^e_i \setminus \mathcal{T}_i, \bar{\omega}_i \odot \mathbf{t}_i\} \qquad (21) \end{aligned}$$

where $\odot$ and $\cdot$ in (21) check connectibility over variables of $\mathcal{V}_i$ only.

Algorithm $\mathbf{A}_3$ (Supervisor of $\mathcal{S}_i$)

1. initialization:

$$\begin{aligned} \mathcal{A}_i \quad &:= \quad \{((\mathbf{v}_i), \emptyset) : \mathbf{v}_i \in \mathcal{I}_i\} \qquad (22) \\ \mathcal{A}_i \quad &:= \quad (\mathrm{Ext}^e_i)^*(\mathcal{A}_i) \qquad (23) \\ \mathcal{M}_{i,j} \quad &:= \quad \mathcal{D}_{\bar{\mathcal{V}}_i \cap \bar{\mathcal{V}}_j} \qquad (24) \end{aligned}$$

2. repeat until no message can be changed and no more run in $\mathcal{A}_i$ can be extended

   (a) on decision to extend a local run, choose $\bar{\omega}_i \in \mathcal{A}_i \bar{\wedge} (\bar{\wedge}_{k \in \mathbf{N}(i)} \mathcal{M}_{k,i})$ such that $|\bar{\omega}_i|_i < T_i$, if any, then

$$\mathcal{A}_i := (\mathcal{A}_i \setminus \{\bar{\omega}_i\}) \cup (\mathrm{Ext}^e_i)^*[\mathrm{Ext}_i(\bar{\omega}_i)] \tag{25}$$

   (b) on decision to update message $\mathcal{M}_{i,j}$,

$$\begin{aligned} \tilde{\mathcal{A}}_i \quad &:= \quad \mathcal{A}_i \bar{\wedge} (\bar{\wedge}_{k \in \mathbf{N}(i) \setminus j} \mathcal{M}_{k,i}) \qquad (26) \\ \mathcal{M}_{i,j} \quad &:= \quad \bar{\Pi}_{\mathcal{V}_i \cap \mathcal{V}_j}(\tilde{\mathcal{A}}_i) \qquad (27) \end{aligned}$$

3. at termination:

$$\mathcal{A}'_i \quad = \quad \mathcal{A}_i \bar{\wedge} (\bar{\wedge}_{k \in \mathbf{N}(i)} \mathcal{M}_{k,i}) \tag{28}$$

Although the termination criterion is not local in $\mathbf{A}_3$, termination can be detected with a distributed protocol [14], which makes $\mathbf{A}_3$ a completely distributed and

asynchronous procedure. A more bothering point concerns (25) since the supervisor of $\mathcal{S}_i$ must compute the influence of all possible extensions in other systems. At the expense of 1/ larger sets $\mathcal{A}_i$, with many useless elements, and 2/ knowledge of extended systems $\mathcal{S}_i^e$. It seems more realistic to assume that the local supervisor of $\mathcal{S}_i$ only knows tiles $\mathcal{T}_i$, and thus cooperates with neighbor supervisors to recursively build the set $\mathcal{A}_i$. Specifically, extensions performed in $\mathcal{A}_j$ and impacting variables $\mathcal{V}_i \cap \mathcal{V}_j$ must be proposed as possible continuations for runs in $\mathcal{A}_i$. An improvement of $\mathbf{A}_3$ implementing this mechanism, and thus handling minimal sets $\mathcal{A}_i$ of local runs, is described in [2].

5 Conclusion

We have proposed a framework to handle large distributed dynamic systems as a graph of interacting components. This framework makes a tight connection between dynamic systems on the one hand, and classical models like Markov fields or Bayesian networks on the other hand. The common feature is the modular definition of systems, by composition of local specifications on subsets of variables, from which a separation property is derived. The separation property on Markov fields induces a notion of sufficient statistics, which forms the basis of modular algorithms. By extension, a similar notion can be defined for distributed dynamic systems, which opens the way to distributed monitoring algorithms. The crucial advantage of distributed algorithms is their ability to work on large systems by local computations : the global state of the system is never handled. This is certainly a key to the state explosion phenomenon.

The parallel between the static and the dynamic frameworks allows to translate a large family of modular algorithms developed for Markov fields into distributed algorithms for distributed dynamic systems. We have presented one of them here, to solve a distributed diagnosis problem, assuming the supervised system has a tree structure. But potential extensions are numerous, for example to deal with systems which do not have a tree structure (this is the default situation with Markov fields). Let us mention for example the celebrated turbo-decoding algorithms (dedicated to turbo error correcting codes, for digital communications), which are excellent approximate estimation algorithms for complex fields. They have a natural counterpart on the side of distributed dynamic systems.

Finally, let us mention that the present framework has been successfully applied to perform distributed failure diagnosis for SDH telecommunication networks, see the MAGDA project at `http://magda.elibel.tm.fr` for details.

References

[1] E. Fabre, "Compositional Models of Distributed and Asynchronous Dynamical Systems," CDC'02.

[2] E. Fabre, "Distributed Diagnosis for Large Discrete Event Dynamic Systems," in preparation.

[3] J. Pearl, "Fusion, Propagation, and Structuring in Belief Networks," Artificial Intelligence, vol. 29, pp. 241-288, 1986.

[4] S.L. Lauritzen, D.J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," J. Royal Statistical Society, Series B, vol. 50, n. 2, pp. 157-224, 1988.

[5] E. Fabre, A. Benveniste, C. Jard, L. Ricker, M. Smith, "Distributed state reconstruction for discrete event systems," proc. 39th Conf. on Detection and Control, Sydney, dec. 2000.

[6] A. Benveniste, E. Fabre, S. Haar "Markov Nets : Probabilistic Models for Distributed and Concurrent Systems," Inria research report n. 4253, sept. 2001.

[7] J. Esparza, S. Römer, "An unfolding algorithm for synchronous products of transition systems," in proceedings of CONCUR'99, LNCS 1664.

[8] A. Arnold, "Finite Transition Systems," Prentice Hall, 1992.

[9] A. Aghasaryan, E. Fabre, A. Benveniste, R. Boubour, C. Jard, "Fault Detection and Diagnosis in Distributed Systems : an Approach by Partially Stochastic Petri nets," Journal of Discrete Event Dynamic Systems, special issue on Hybrid Systems, vol. 8, pp. 203-231, June 98.

[10] A. Benveniste, B.C. Levy, E. Fabre, P. Le Guernic, "A Calculus of Stochastic Systems : Specification, Simulation, and Hidden State Estimation," Theoretical Computer Science, no. 152, pp. 171-217, 1995.

[11] R. Debouk, S. Lafortune, D. Teneketzis, "Coordinated decentralized protocols for failure diagnosis of discrete event systems," Journal of Discrete Event Dynamic Systems : Theory and Applications, vol. 10, no. 1-2, pp. 33-86, Jan. 2000.

[12] P. Baroni, G. Lamperti, P. Pogliano, M. Zanella, "Diagnosis of large active systems," Artificial Intelligence 110, pp. 135-183, 1999.

[13] L. Rozé, M-O. Cordier, "Diagnosis Discrete-Event Systems: Extending the Diagnoser Approach to Deal with Telecommunication Networks," J. of Discrete Event Dynamic Sytems: Theory and Applications, vol. 12, pp. 43-81, 2002.

[14] M. Raynal, "Distributed Algorithms and Protocols," Wiley & Sons, 1988.