Compositional Models of Distributed and Asynchronous Dynamical Systems [1]

Eric Fabre[2]

Abstract

This paper proposes a framework to describe and handle distributed systems as an interaction graph of elementary components. Components are discrete event systems operating on several state variables, and defining local dynamics on these variables. Components are interconnected by sharing variables, which defines the interaction graph of the compound system. They evolve asynchronously, with their own clock, so there is no notion of global time. This behavior is captured by the so-called true concurrency semantics on trajectories of the system. Just like the global system factorizes as a product of components, we prove that its trajectories also "factorize." As a consequence, the global system can be handled by parts, for example for state estimation ; the global state of the system is never computed. This is a key to deal with large systems. This framework has been applied to design distributed diagnosis algorithms for telecommunication networks.

1 Introduction

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 [3, 9, 13, 5]. Telecommunication networks are a good example of this situation. A network can be viewed as a large system composed of interconnected elements (add/drop multiplexers, routers, etc.). A model of this large dynamic system can be obtained by composing models of network elements, the latter being themselves compound systems combining elementary functions. The global system is generally intractable because of the state explosion phenomenon. Hence problems like (maximum likelihood) state estimation, failure diagnosis, etc. which would be easy for small (stochastic) systems, are handled in practice with heuristics, human experience or expert systems.

[1]Research supported in part by French RNRT under project MAGDA (Modelling and Learning for a Distributed Management of Alarms, `http://magda.elibel.tm.fr/`), and by the European Commission under project HYBRIDGE, IST-2001-32460.

[2]Irisa/Inria, Campus de Beaulieu, 35042 Rennes cedex, France ; `Eric.Fabre@irisa.fr`

We propose to circumvent the difficulty of size by abandoning any idea of global processing. Specifically, we propose to handle large systems by parts, or component by component, through modular algorithms (fig. 1). Turning back to the state estimation problem for example, and assuming observations are available on various components, this would mean running a state estimation algorithm for each component (fed with local observations), and coodinating them in order to provide coherent solutions. Doing so, the state explosion phenomenon can remain under control, since only local states of a component are handled. But the difficulty concentrates in the coordination procedure. Observe also that this paradigm slightly changes the nature of the problem : one is not any more interested in the state of the global system, but rather in its "projection" on a particular component. We thus obtain viewpoints on the solution.

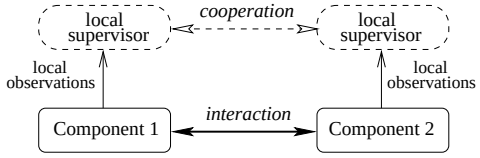

Figure 1: *A distributed system (bottom) and a modular monitoring architecture (for example for state estimation).*

We believe modular processings, and further distributed processings, are a key to handle large compound systems. In this paper, we focus on a modeling framework in view of this objective. We base our construction on ideas developed for Markov random fields, or Bayesian networks, which can be viewed as *static* compound systems. A Markov field is obtained by 1/ taking a large number of variables, and 2/ specifying local interactions on subsets of these variables. Interactions may be defined by constraints on possible values, and/or by *potential* functions, which allows to build a probability distribution on all variables. This construction through local specifications yields a factorization property on constraints, or on the probability distribution. In many applications, one has to recover the (most likely) state/configuration of the field matching some observations. Due to the size of the field, a global handling is unaffordable, and a large variety of modular algorithms have been designed, precisely based on the local specification of the field. Modular algorithms

rely on the so-called separation property, which roughly says that two areas of the field are independent given values at their interface. As a consequence, they can be processed separately except at this interface, where information must be exchanged. These results are recalled in section 2.

We propose to extend this approach to distributed *dynamic* systems. These systems are obtained by connecting local asynchronous components. Components have their own dynamics, a private notion of time (so there is no global time in the system), but synchronize with neighboring components on some particular events, like the exchange of messages for example. Our main contribution concerns the statement of a separation property on trajectories of these distributed dynamic systems (section 3). This property allows to extend most algorithms designed for static systems (or Markov fields) to distributed dynamic systems. This paper focusses on the modeling framework; algorithms are presented in a companion paper [1].

2 Compound static systems

2.1 Systems and composition

Notations. The systems we consider operate on sets of variables. Variables are denoted by capital letters: $A, B, V \ldots$ They take values in finite domains $\mathcal{D}_A, \mathcal{D}_B, \mathcal{D}_V \ldots$ and values are expressed by lower-case letters $a, b, v \ldots$ Let $\mathcal{V} = \{V_1, \ldots, V_n\}$ be a set of variables; a *state* $\mathbf{v}$ is a function that assigns to each variable of $\mathcal{V}$ a value of its domain. By abuse of notations, we represent states as tuples $\mathbf{v} = (v_1, \ldots, v_n)$, assuming there exists some natural ordering on variables of $\mathcal{V}$, and we denote by $\mathcal{D}_\mathcal{V} = \mathcal{D}_{V_1} \times \cdots \times \mathcal{D}_{V_n}$ the domain of states. For $\mathcal{V}' \subset \mathcal{V}$, we denote by $\Pi_{\mathcal{V}'}$ the canonical projection on states of $\mathcal{V}'$, which erases variables of $\mathcal{V}$ not present in $\mathcal{V}'$.

A *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 $\mathcal{O} \subseteq \mathcal{D}_\mathcal{V}$ is the set of legal *states* $(v_1, \ldots, v_n)$ of $\mathcal{S}$. Hence a system is defined by *constraints* on the value of its variables. For a matter of space, the general case of stochastic systems is not described here (see [2] for details).

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 the *composition* of systems 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 $\mathcal{S}_2$ interact through the variables $\mathcal{V}_1 \cap \mathcal{V}_2$ they have in common, 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 $\mathcal{O} = (\Pi_{\mathcal{V}_1})^{-1}(\mathcal{O}_1) \cap (\Pi_{\mathcal{V}_2})^{-1}(\mathcal{O}_2)$. Observe that composition is commutative and associative.

$\mathcal{S}$ is said to be a *distributed system* as soon as it can be expressed as

$$\mathcal{S} \quad = \quad \mathcal{S}_1 | \mathcal{S}_2 | \ldots | \mathcal{S}_N \quad (2)$$

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 (fig. 2, left).

Given a hypergraph $\mathcal{G}$ defined by edges $\mathcal{V}_1, \ldots, \mathcal{V}_N$, one can check that a system $\mathcal{S}$ "factorizes on $\mathcal{G}$" by first building subsystems $\mathcal{S}_i = \Pi_{\mathcal{V}_i}(\mathcal{S})$ and then comparing $\mathcal{S}$ to $\mathcal{S}_1 | \ldots | \mathcal{S}_N$. Factorization limits the size of constraints in a system, in terms of number of variables involved in these constraints. Hence the product $\mathcal{S}_1 | \ldots | \mathcal{S}_N$ is generally *larger* than $\mathcal{S}$, i.e. allows more states or is more permissive. For example, let $\mathcal{S} = (\{A, B, C\}, \{(a, b, c'), (a, b', c), (a', b, c)\})$. Its projections on $\mathcal{V}_1 = \{A, B\}$, $\mathcal{V}_2 = \{B, C\}$ and $\mathcal{V}_3 = \{C, A\}$ are defined by

$$\begin{array}{rcl} \mathcal{S}_1 = \Pi_{\{A,B\}}(\mathcal{S}) & = & (\{A, B\}, \{(a, b), (a, b'), (a', b)\}) \\ \mathcal{S}_2 = \Pi_{\{B,C\}}(\mathcal{S}) & = & (\{B, C\}, \{(b, c), (b, c'), (b', c)\}) \\ \mathcal{S}_3 = \Pi_{\{A,C\}}(\mathcal{S}) & = & (\{A, C\}, \{(a, c), (a, c'), (a', c)\}) \end{array}$$

The product yields $\mathcal{S}' = \mathcal{S}_1 | \mathcal{S}_2 | \mathcal{S}_3 = (\{A, B, C\}, \{(a, b, c), (a, b, c'), (a, b', c), (a', b, c)\})$ which contains the extra state (a, b, c). To remove this state, one needs a constraint acting simultaneously on A, B and C.

This property generalizes in the following way: let $\mathcal{V} = \cup_{i=1}^{N} \mathcal{V}_i$ and $\mathcal{V} = \cup_{j=1}^{M} \mathcal{V}'_j$ define two hypergraphs $\mathcal{G}$ and $\mathcal{G}'$ on $\mathcal{V}$, such that $\forall i, \exists j : \mathcal{V}_i \subset \mathcal{V}'_j$. We say that $\mathcal{G} \subset \mathcal{G}'$, or $\mathcal{G}$ is finer that $\mathcal{G}'$. Then any system factorizing on $\mathcal{G}$ also factorizes on $\mathcal{G}'$.

2.2 Separation property

The interaction graph not only displays the structure of interactions beween components, but also indicates which information about the whole system is useful to a given component. This is based on the separation criterion. Let $\mathcal{S}$ factorize into $\mathcal{S}_1 | \ldots | \mathcal{S}_N$, which defines the hypergraph $\mathcal{G}$. For $L \subset \{1, \ldots, N\}$ an index set, we denote by $\mathcal{S}_L = (\mathcal{V}_L, \mathcal{O}_L)$ the product $|_{l \in L} \mathcal{S}_l$, and by $\mathbf{v}_L$ a state in $\mathcal{O}_L$.

Definition 1 *Let the hypergraph $\mathcal{G}$ be defined by edges $\mathcal{V}_1, \ldots, \mathcal{V}_N$. Let $\mathcal{X}, \mathcal{Y}, \mathcal{Z} \subseteq \mathcal{V}$ 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).*

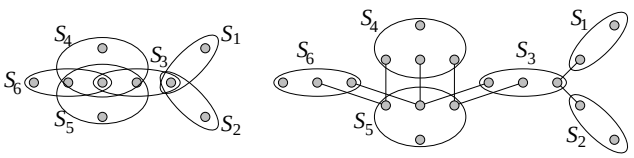

Figure 2: *A hypertree with 6 edges (left). Its tree structure is evidenced on the righthand side by associating neighbors to each edge. Some nodes are duplicated, and the correspondence between them indicate the neighboring structure.*

The separation criterion defines some notion of independence in the global system. Assume for example $\mathcal{S}$ factorizes into $\mathcal{S}_1|\mathcal{S}_2|\mathcal{S}_3$, where $\mathcal{S}_1$ separates $\mathcal{S}_2$ from $\mathcal{S}_3$ (fig. 3) (this situation may be appear after the grouping of terms in the general form $\mathcal{S} = \mathcal{S}_1|\dots|\mathcal{S}_N$). Let $\mathbf{v}_1$ be a state of $\mathcal{S}_1$, and chose any pair of states $(\mathbf{v}_2, \mathbf{v}_3)$ in $\mathcal{O}_2 \times \mathcal{O}_3$ such that $\Pi_{\mathcal{V}_1\cap\mathcal{V}_2}(\mathbf{v}_2) = \Pi_{\mathcal{V}_1\cap\mathcal{V}_2}(\mathbf{v}_1)$ and $\Pi_{\mathcal{V}_1\cap\mathcal{V}_3}(\mathbf{v}_3) = \Pi_{\mathcal{V}_1\cap\mathcal{V}_3}(\mathbf{v}_1)$. Then $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ can be glued to form a state of the compound system $\mathcal{S}$. In other words, given $\mathbf{v}_1$, the choices of $\mathbf{v}_2$ and $\mathbf{v}_3$ are independent (provided they match $\mathbf{v}_1$). This is reminiscent of Markov chains, where trajectories in the past and in the future are independent given the present.

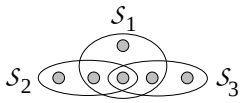

Figure 3: *A chain of systems : $\mathcal{S}_1$ separates $\mathcal{S}_2$ from $\mathcal{S}_3$.*

2.3 Towards modular processings

Some components $\mathcal{S}_i$ may determine exactly the value of part of their variables. Hence the framework above captures the case where observations are available in the system. So one may be interested in computing the canonical factorization of $\mathcal{S}$, i.e. $\mathcal{S} = \mathcal{S}'_1|\dots|\mathcal{S}'_N$ where $\mathcal{S}'_i = (\mathcal{V}_i, \mathcal{O}'_i) \triangleq \Pi_{\mathcal{V}_i}(\mathcal{S})$. This means reducing local state sets $\mathcal{O}_i$ to $\mathcal{O}'_i$, which contain states $\mathbf{v}_i$ participating to at least one global state $\mathbf{v}$ of $\mathcal{S}$, i.e. matching observations[1].

Conditional independence is central to design fast estimation algorithms for Markov processes. By analogy, the separation criterion is central in defining what a component should know about the rest of the system, in view of the reduction task mentioned above. In the simple case $\mathcal{S} = \mathcal{S}_1|\mathcal{S}_2$, one has

$$\begin{aligned} \mathcal{O}'_i &= \Pi_{\mathcal{V}_i}(\mathcal{O}_i \wedge \mathcal{O}_j) \\ &= \mathcal{O}_i \wedge \Pi_{\mathcal{V}_i}(\mathcal{O}_j) \\ &= \mathcal{O}_i \wedge \Pi_{\mathcal{V}_{1\cdot 2}}(\mathcal{O}_j) \end{aligned} \tag{3}$$

for $1 \le i, j \le 2$, $i \ne j$, and with $\mathcal{V}_{1\cdot 2} = \mathcal{V}_1 \cap \mathcal{V}_2$. In the second equality, we slightly abuse notations with $\Pi_{\mathcal{V}_i}(\mathcal{O}_j)$ since $\mathcal{V}_i \not\subseteq \mathcal{V}_j$. By convention, this means that $\mathcal{O}_j$ is first extended to $\mathcal{V}_i \cup \mathcal{V}_j$ by $\Pi^{-1}_{\mathcal{V}_j}$ and then reduced to $\mathcal{V}_i$. The term $\Pi_{\mathcal{V}_{1\cdot 2}}(\mathcal{O}_j)$ represents the necessary and sufficient information $\mathcal{S}_i$ needs to know about $\mathcal{S}_j$ to determine $\mathcal{O}'_i$.

When $\mathcal{S}$ factorizes as on fig. 3, for $\mathcal{S}_2$ (3) becomes

$$\begin{aligned} \mathcal{O}'_2 &= \Pi_{\mathcal{V}_2}(\mathcal{O}_1 \wedge \mathcal{O}_2 \wedge \mathcal{O}_3) \\ &= \mathcal{O}_2 \wedge \Pi_{\mathcal{V}_2}(\mathcal{O}_1 \wedge \mathcal{O}_3) \\ &= \mathcal{O}_2 \wedge \Pi_{\mathcal{V}_2}[\mathcal{O}_1 \wedge \Pi_{\mathcal{V}_1}(\mathcal{O}_3)] \\ &= \mathcal{O}_2 \wedge \Pi_{\mathcal{V}_{1\cdot 2}}[\mathcal{O}_1 \wedge \Pi_{\mathcal{V}_{1\cdot 3}}(\mathcal{O}_3)] \end{aligned} \tag{4}$$

where the second equality uses (3). The third one derives from the fact that variables in $\mathcal{V}_3 \setminus \mathcal{V}_1$ will be erased by $\Pi_{\mathcal{V}_2}$ since $\mathcal{V}_{2\cdot 3} \subseteq \mathcal{V}_1$. (4) expresses first that the information about $\mathcal{S}$ which is necessary to $\mathcal{S}_2$ can be summarized at the level of the intermediary component $\mathcal{S}_1$: $\mathcal{O}_1 \wedge \Pi_{\mathcal{V}_{1\cdot 3}}(\mathcal{O}_3)$ only involves variables of $\mathcal{S}_1$. Secondly, the nested shape of (4) is reminiscent of a recursive estimation procedure for Markov chains, where information of the past ($\mathcal{S}_3$) is combined with information of the present ($\mathcal{S}_1$) and forwarded toward the future ($\mathcal{S}_2$). In the same way, $\mathcal{O}'_1$ is given by

$$\begin{aligned} \mathcal{O}'_1 &= \Pi_{\mathcal{V}_1}(\mathcal{O}_1 \wedge \mathcal{O}_2 \wedge \mathcal{O}_3) \\ &= \mathcal{O}_1 \wedge \Pi_{\mathcal{V}_1}(\mathcal{O}_2 \wedge \mathcal{O}_3) \\ &= \mathcal{O}_1 \wedge \Pi_{\mathcal{V}_1}(\mathcal{O}_2) \wedge \Pi_{\mathcal{V}_1}(\mathcal{O}_3) \\ &= \mathcal{O}_1 \wedge \Pi_{\mathcal{V}_{1\cdot 2}}(\mathcal{O}_2) \wedge \Pi_{\mathcal{V}_{1\cdot 3}}(\mathcal{O}_3) \end{aligned} \tag{5}$$

where the second equality uses (3), and the third one comes from the fact that $\mathcal{S}_2$ and $\mathcal{S}_3$ have all their common variables inside $\mathcal{V}_1$. Hence $\mathcal{S}_1$ needs messages $\Pi_{\mathcal{V}_{1\cdot i}}(\mathcal{O}_i)$ from its two neighbors $\mathcal{S}_i$, $i = 2, 3$. Again, by analogy with state estimation for Markov chains, (5) can be read as a merge equation of information coming from past and future.

If $\mathcal{S} = \mathcal{S}_1|\dots|\mathcal{S}_N$ is a tree, it can be shown that adequate combinations of the local operations (4) and (5) are enough to compute all sets $\mathcal{O}'_i$ in finite time [1, 2].

3 Distributed dynamic systems

We now extend the previous framework to dynamic systems. These systems are finite state machines (FSM) with two special features : first they involve several (state) variables instead of a single one, and secondly transitions operate on part of these variables. This framework (close to models in [7, 6]) can be seen as an extension of Petri nets. Special trajectory semantics are defined to capture the asynchronous behavior of components. A separation property is then stated on trajectories of compound systems, which opens the way to modular processings.

[1] In a stochastic setting, this reduction yields, for example, local values in $\mathcal{V}_i$ of the most likely state of $\mathcal{S}$, given observations.

3.1 Systems and composition

Definition 2 *A* dynamic system $\mathcal{S}$ *is a triple* $(\mathcal{V},\mathcal{I},\mathcal{T})$ *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 states* $\mathbf{v}=(v_1,\ldots,v_n)$*, and* $\mathcal{T}$ *is 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 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* $\mathbf{t}$ *(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 systems extends (1). 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

$$\mathcal{S}=(\mathcal{V},\mathcal{I},\mathcal{T})=\mathcal{S}_1|\mathcal{S}_2 \quad\Leftrightarrow\quad \left\{\begin{array}{rcl}\mathcal{V}&=&\mathcal{V}_1\cup\mathcal{V}_2\\ \mathcal{I}&=&\mathcal{I}_1\wedge\mathcal{I}_2\\ \mathcal{T}&=&\mathcal{T}_1\cup\mathcal{T}_2\end{array}\right. \quad (6)$$

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

3.2 Trajectories

We now proceed to defining runs of a dynamic system. Let $\mathbf{t}\in\mathcal{T}$ be a tile and $\mathbf{v}$ a state of $\mathcal{S}$. $\mathbf{t}$ is said to be *enabled* by $\mathbf{v}$ iff $\mathbf{v}_{\mathcal{V}_\mathbf{t}}\triangleq\Pi_{\mathcal{V}_\mathbf{t}}(\mathbf{v})=\mathbf{v}_\mathbf{t}^-$. By connecting $\mathbf{t}$ to $\mathbf{v}$ (or by *firing* $\mathbf{t}$), one gets state $\mathbf{v}'$ defined by

$$\mathbf{v}'_{\mathcal{V}_\mathbf{t}}=\mathbf{v}_\mathbf{t}^+ \qquad\qquad \mathbf{v}'_{\mathcal{V}\setminus\mathcal{V}_\mathbf{t}}=\mathbf{v}_{\mathcal{V}\setminus\mathcal{V}_\mathbf{t}}$$

By analogy with Petri nets, connectibility is denoted by $\mathbf{v}[\mathbf{t}\rangle$, and the result of the connection by $\mathbf{v}[\mathbf{t}\rangle\mathbf{v}'$.

At runtime, a system may fire several times the same transition, so we define trajectories in terms of *events*. An event $\mathbf{e}$ represents the firing of transition $\mathbf{t}=\phi(\mathbf{e})\in\mathcal{T}$. The most straightforward definition of a *run* (or *trajectory*) ρ for system $\mathcal{S}$ would be as usually a sequence $(\mathbf{v},\mathbf{e}_1,\ldots,\mathbf{e}_n)$ of events anchored at some initial state $\mathbf{v}$, and satisfying

$$\mathbf{v}[\phi(\mathbf{e}_1)\rangle\mathbf{v}_1[\phi(\mathbf{e}_2)\rangle\mathbf{v}_2\ \cdots\ \mathbf{v}_{n-1}[\phi(\mathbf{e}_n)\rangle\mathbf{v}_n \quad (7)$$

However, this definition implements the total order semantics on runs, which is not adequate here since it assumes a global clock governing the behavior of the system. We rather wish to capture the asynchry which is inherently present in a distributed system : two components naturally evolve independently in time between instants at which they synchronize (exchange of a message for example).

3.3 Partial order semantics on trajectories

Let us consider a system $\mathcal{S}$ with two variables A and B. Fig. 4 (top) represents a run $\rho_1=((a,b),\mathbf{e}_1,\ldots,\mathbf{e}_8)$ where events appear as rectangles, the grey part of which identifies impacted variables. Intermediary values of variables are not represented. Due to the definition of connectibility, the run ρ_2 depicted on fig. 4 (bottom) is also valid and finishes with the same values of A and B. It is obtained by applying a series of permutations in ρ_1, where two successive events $\mathbf{e}$ and $\mathbf{e}'$ can be permuted iff $\mathcal{V}_\mathbf{e}\cap\mathcal{V}_{\mathbf{e}'}=\emptyset$. In this case, firing $\phi(\mathbf{e})$ and $\phi(\mathbf{e}')$ in any order, or even simultaneously, yields the same result : $\mathbf{e}$ and $\mathbf{e}'$ are said to be *concurrent* in the run.

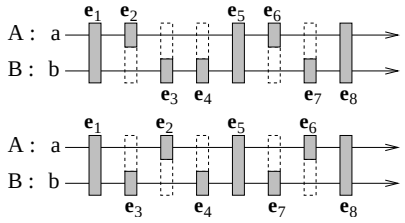

Figure 4: *These two sequences of events are considered as equivalent in the true concurrency semantics.*

Concurrency is a central notion in (distributed) systems based on tiles : it identifies areas of independent behaviors in a run. For example, variables A and B evolve independently between $\mathbf{e}_1$ and $\mathbf{e}_5$; the ordering of firings in this section is irrelevant. Therefore, it becomes natural to define a trajectory not any more as a *sequence* of events, but as an *equivalence class* of sequences, where equivalence derives from concurrency. This amounts to considering a run as a *partial order* of events (or *puzzle*), as depicted on fig. 5.

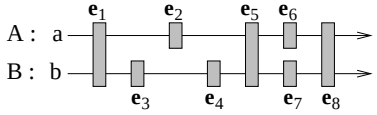

Figure 5: *A run as a partial order of events.*

Formally, for two sequences $\rho_1=(\mathbf{v},\ldots,\mathbf{e},\mathbf{e}',\ldots)$ and $\rho_2=(\mathbf{v},\ldots,\mathbf{e}',\mathbf{e},\ldots)$ differing only by the order of $\mathbf{e}$ and $\mathbf{e}'$, let us say $\rho_1\sim\rho_2$ iff $\mathcal{V}_\mathbf{e}\cap\mathcal{V}_{\mathbf{e}'}=\emptyset$: $\mathbf{e}$ and $\mathbf{e}'$ are concurrent. The equivalence relation $\approx$ we consider is defined as $\sim^*$, i.e. $\rho\approx\rho'$ iff there exists a sequence $\rho=\rho_0,\rho_1,\ldots,\rho_n=\rho'$ such that $\rho_i\sim\rho_{i+1}$. In words, one can go from ρ to ρ' by permuting successive concurrent events. In the true concurrency semantics, runs ω are equivalence classes $[\rho]_\approx$, that we denote as $\omega=(\rho,\prec)$, where $\prec$ is the partial order relation on events of ρ.

3.4 Practical handling of partial orders

Working with partial orders of events may require heavy notations. In a sequence, a simple index is enough to distinguish two events. In a partial order, one would like to avoid a total indexing and rather identify an event by its "position" in the partial order. To do so, we propose here a notation based on the notion of *trace*. We illustrate it on an example (a

more formal derivation is given in [2]). Let us come back to fig. 5, and replace events by the tile they represent, which gives fig. 6. It can be shown that this

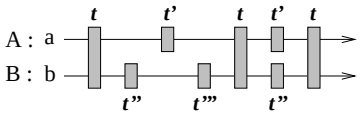

Figure 6: *Partial order of events with several occurences of the same tile.*

run ω is equivalently described by the pair (ω_A, ω_B), where ω_V is the initial value v of variable V followed by the *sequence of tiles* that have impacted V. On the example:

$$\begin{aligned} \omega_A &= (a, \mathbf{t}, \mathbf{t}', \mathbf{t}, \mathbf{t}', \mathbf{t}) \\ \omega_B &= (b, \mathbf{t}, \mathbf{t}'', \mathbf{t}''', \mathbf{t}, \mathbf{t}'', \mathbf{t}) \end{aligned}$$

This notation naturally captures the total ordering of events impacting a given variable, but also keeps track of the synchronization information between sequences ω_V : if a tile $\mathbf{t}$ is such that $\{V_1, V_2\} \subseteq \mathcal{V}_\mathbf{t}$, then the k-th occurence of $\mathbf{t}$ in ω_{V_1} and its k-th occurence in ω_{V_2} correspond to the same event. This allows to completely recover the partial order $\omega = (\rho, \prec)$.

3.5 Separation property

We now proceed to establishing a factorization property on *trajectories* of a system $\mathcal{S}$. 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 ω_V in domain $\mathcal{D}_{\Omega_V}$

$$\mathcal{D}_{\Omega_V} = \mathcal{D}_V \times (\mathcal{T}_V)^* \quad \text{with} \quad \mathcal{T}_V = \{\mathbf{t} \in \mathcal{T}, V \in \mathcal{V}_\mathbf{t}\}$$

Hence variable Ω_V in $\bar{\mathcal{S}}$ describes all possible histories (in $\mathcal{S}$) of variable V alone[2]. The state set $\bar{\mathcal{O}}$ of $\bar{\mathcal{S}}$ is composed of all (finite size) trajectories of $\mathcal{S}$, handled as tuples $\omega_\mathcal{V} = (\omega_V, V \in \mathcal{V})$, following the notation of the previous section.

Assuming $\mathcal{S} = (\mathcal{V}, \mathcal{I}, \mathcal{T}) = \mathcal{S}_1 | \dots | \mathcal{S}_N$, 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 $\Omega_{\mathcal{V}_i}$ of the extended system $\mathcal{S}_i^e = (\mathcal{V}_i, \mathcal{I}_i, \mathcal{T}_i^e)$. The new transition set is given by

$$\mathcal{T}_i^e = \mathcal{T}_i \cup (\bigcup_{V \in \mathcal{V}_i} \mathcal{T}_V) \tag{8}$$

[2]Notice that we could have limited $\mathcal{D}_{\Omega_V}$ to values $(v, \mathbf{t}_1, \dots, \mathbf{t}_n)$ which guarantee coherent modifications of the value of V, i.e. such that $\mathbf{v}^-_{\mathbf{t}_1}(V) = v$ and $\mathbf{v}^+_{\mathbf{t}_i}(V) = \mathbf{v}^-_{\mathbf{t}_{i+1}}(V)$, $i = 1 \dots n-1$. This property is ensured by the definition of $\bar{\mathcal{O}}$ and further of sets $\bar{\mathcal{O}}_i$.

i.e. it incorporates all transitions of other components $\mathcal{S}_j$ which have an action on variables of $\mathcal{V}_i$. Trajectories of $\mathcal{S}_i^e$ are built by connecting tiles as in (7), but checking only the connectibility on variables of $\mathcal{V}_i$. Hence $\mathcal{S}_i^e$ behaves as if it had all transitions of $\mathcal{S}$ "truncated" to variables of $\mathcal{V}_i$ (see fig. 7).

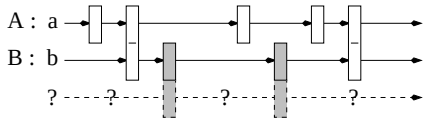

Figure 7: *Trajectory of an extended system $\mathcal{S}_i^e$, with $\mathcal{V}_i = \{A, B\}$. Gray tiles belong to other components. Their action on variables outside $\mathcal{V}_i$ is ignored.*

Finally, let us refine the definition (1) of composition | for static systems $\bar{\mathcal{S}}_i$. The conjunction operation $\wedge$ on local state sets becomes

$$\bar{\mathcal{O}}_i \wedge \bar{\mathcal{O}}_j = \mathcal{P}\left[(\Pi_{\bar{\mathcal{V}}_i})^{-1}(\bar{\mathcal{O}}_i) \cap (\Pi_{\bar{\mathcal{V}}_j})^{-1}(\bar{\mathcal{O}}_j)\right] \tag{9}$$

where operator $\mathcal{P}$ keeps only tuples $(\omega_V, V \in \mathcal{V}_i \cup \mathcal{V}_j)$ corresponding to valid partial orders. In other words, states of $\bar{\mathcal{S}}_i | \bar{\mathcal{S}}_j$ are obtained by gluing local trajectories of $\mathcal{S}_i^e$ and $\mathcal{S}_j^e$ which coincide on shared variables $\mathcal{V}_i \cap \mathcal{V}_j$, provided the agregated tuple do represent a partial order. Despite the presence of $\mathcal{P}$, the refined composition operator | remains commutative and associative.

Theorem 1 *Let $\mathcal{S}$ factorize into $\mathcal{S}_1 | \dots | \mathcal{S}_N$, and let static systems $\bar{\mathcal{S}}$ and $\bar{\mathcal{S}}_i, 1 \le i \le N$, be defined as above. The following factorization holds: $\bar{\mathcal{S}} = \bar{\mathcal{S}}_1 | \dots | \bar{\mathcal{S}}_N$.*

The proof is given in the extended paper [2]. Theorem 1 almost brings us back to the framework of section 2. This factorized representation of trajectories not only reduces the combinatorial explosion of possible trajectories of a compound system, but also opens the way to modular processings. For example, one can imagine solving problems like determining all trajectories of $\mathcal{S}$ which match some observed events on components (this is the purpose of the companion paper [1]), by means of local processings like (3), (4) and (5). As in section 2, this will provide "viewpoints" on these trajectories, i.e one will get restrictions of these possible trajectories to components $\mathcal{S}_i$. The missing ingredient toward this objective is precisely the notion of restriction of a trajectory, that we introduce now.

3.6 Modular processings on traces

Local processings (3) to (5) were designed for the simple composition rule (1) on static systems. The presence of the $\mathcal{P}$ operator in the new definition of composition (9) requires to modify the notion of projection to make them hold. We briefly give hints on how this can be done.

Runs ω we have defined are actually *traces* of a distributed system. In trace theory, the projection of a trace to a subset $\mathcal{E}$ of events is defined as restriction to $\mathcal{E}$ of the partial order represented by that trace. With notation $\omega_{\mathcal{V}} = (\omega_V, V \in \mathcal{V})$ for traces, $\omega_{\mathcal{V}'} = (\omega_V, V \in \mathcal{V}')$ does capture all events concerning variables of $\mathcal{V}' \subseteq \mathcal{V}$, but $\omega_{\mathcal{V}'}$ may not encode all partial order relations on these events that were present in $\omega_{\mathcal{V}}$. Consider the example of fig. 8 : $\mathbf{e}_1$ and $\mathbf{e}_4$ appear as concurrent in (ω_A, ω_B), whereas they are causaly related in $(\omega_A, \omega_B, \omega_C)$.

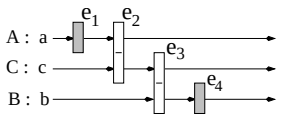

Figure 8: *Without ω_C, a causal link from $\mathbf{e}_1$ to $\mathbf{e}_4$ is lost.*

In order to keep track of this information, let us define $\bar{\Pi}_{\mathcal{V}'}(\omega_{\mathcal{V}})$ as the restriction of the partial order $\omega_{\mathcal{V}}$ to events impacting variables in $\mathcal{V}' \subseteq \mathcal{V}$. This results in $\omega_{\mathcal{V}'}$ augmented with extra causality relations, whence notation $\bar{\Pi}_{\mathcal{V}'}(\omega_{\mathcal{V}}) = (\omega_{\mathcal{V}'}, \prec)$. On the example above, $\bar{\Pi}_{\Omega_A,\Omega_B}(\omega)$ results in (ω_A, ω_B) augmented with the causality relation $\mathbf{e}_2 \prec \mathbf{e}_3$. To comply with this new structure, we refine $\wedge$ on traces as

$$(\omega_{\mathcal{V}_i}, \prec_i) \wedge (\omega_{\mathcal{V}_j}, \prec_j) \quad = \quad \mathcal{P}[(\omega_{\mathcal{V}_i} \wedge \omega_{\mathcal{V}_j}, \prec_i \wedge \prec_j)]$$

where $\omega_{\mathcal{V}_i} \wedge \omega_{\mathcal{V}_j}$ is the usual composition for static systems (may yield $\emptyset$), $\prec_i \wedge \prec_j$ is the union of extra partial order relations, and $\mathcal{P}$ checks that the right hand side term is a valid partial order of events. With this definition of composition, equations (3) to (5) become valid for systems $\bar{\mathcal{S}}_i$ provided Π is replaced by $\bar{\Pi}$, the restriction of a trace $(\omega_V, \prec)$. This opens the way to the reconstruction of trajectories of a distributed dynamic system by means of modular algorithms.

4 Conclusion

We have proposed a framework to handle large distributed 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 the system, by composition of local specifications on subsets of variables, from which a separation property is derived. The separation property on Markov fields induces a conditional independence relation and allows to define a notion of sufficient statistics, which forms the basis of modular algorithms. By extension, we have derived a similar notion of "sufficient statistics" for distributed dynamic systems, which captures information both in time and in space (presented in a non stochastic case here). This notion forms the basis of modular estimation algorithms in Markov fields, which can thus be directly transposed for distributed dynamic systems, to solve problems like state/trajectory estimation given (distributed) observations. This is described in the companion paper [1]. This framework has been successfully applied to design distributed diagnosis algorithms for SDH telecommunication networks. We believe modular processings could also make large systems amenable to problems like verification or control.

References

[1] E. Fabre, V. Pigourier, "Monitoring distributed systems with distributed algorithms," *CDC'02.*

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

[3] R. Debouk, S. Lafortune, D. Teneketzis, "Coordinated decentralized protocols for failure diagnosis of discrete event systems," *J. Disc. Event Dyn. Sys. : Th. and Appli.*, vol. 10, no. 1-2, pp. 33-86, Jan. 2000.

[4] P. Baroni, G. Lamperti, P. Pogliano, M. Zanella, "Diagnosis of large active systems," *Artif. Intel. 110*, pp. 135-183, 1999.

[5] Y. Pencolé, M-O. Cordier, L. Rozé, "A decentralized model-based diagnostic tool for complex systems," *13th IEEE Int. Conf. on Tools with Artif. Intel. (ICTAI'01)*, pp. 95-102, Dallas, 2001.

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

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

[8] W. Vogler, "Modular Construction and Partial Order Semantics of Petri Nets," LNCS no. 625, 1992.

[9] C. G. Cassandras, S. Lafortune, "Introduction to discrete event systems," Kluwer Acad. Pub., 1999.

[10] J. Pearl, "Fusion, Propagation, and Structuring in Belief Networks," *Artif. Intel.*, vol. 29, pp. 241-288, 1986.

[11] 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.

[12] 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.

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