Symbolic Supervisory Control of Distributed Systems with Communications

Gabriel Kalyon, Tristan Le Gall, Hervé Marchand, Thierry Massart

Symbolic Supervisory Control of Distributed Systems with Communications

Gabriel Kalyon, Tristan Le Gall, Hervé Marchand, Thierry Massart

***Abstract*—We consider the control of distributed systems composed of subsystems communicating asynchronously; the aim is to build local controllers that restrict the behavior of a distributed system in order to satisfy a global state avoidance property. We model distributed systems as *communicating finite state machines* with reliable unbounded FIFO queues between subsystems. Local controllers can only observe the behavior of their proper subsystem and do not see the queue contents. To refine their control policy, controllers can use the FIFO queues to communicate by piggy-backing extra information (some timestamps and their state estimates) to the messages sent by the subsystems. We provide an algorithm that computes, for each local subsystem (and thus for each controller), during the execution of the system, an estimate of the current global state of the distributed system. We then define a synthesis algorithm to compute local controllers. Our method relies on the computation of (co-)reachable states. Since the reachability problem is undecidable in our model, we use abstract interpretation techniques to obtain overapproximations of (co-)reachable states. An implementation of our algorithms provides an empirical evaluation of our method.**

I. Introduction

In the framework of control of distributed systems, two classes of systems are generally considered, depending on whether the communication between subsystems is *synchronous*[1] or not. When the network communication can be done through multiplexing or when the synchrony hypothesis [3] can be made, the *decentralized control problem* and the *modular control problem* address the design of coordinated controllers that jointly ensure the desired properties for this kind of systems [40], [35], [34], [10], [20]. When considering *asynchronous* distributed systems, the communication delays between the components of the system must also be taken into account. Note that in both cases the *distributed control synthesis* is undecidable [32], [38].

Our aim is to solve the latter problem, when the system to be controlled is composed of n (finite) subsystems that communicate through reliable unbounded FIFO channels. These subsystems are modeled by *communicating finite state machines* [5] (CFSM for short), a classical model for distributed systems like communication protocols [31], [22] and web services [30]. Following the architecture described in Figure 1, we assume that each subsystem is controlled by a *local* controller which only observes the actions fired by its subsystem and communicates with it with zero delays. The control decision is based on the knowledge each local controller has about the current state of the whole system. Controllers communicate with each other by adding some extra information (some timestamps and their state estimates) to the messages normally exchanged by the subsystems. These communications allow them to refine their knowledge, so that control decisions may be more permissive.

In this paper, we focus on the *state avoidance control problem* that consists in preventing the system from reaching some bad states. To solve this control problem, we first compute offline (i.e. before the system execution), the set of states that leads to bad states by only taking uncontrollable transitions. We then compute online (i.e. during the execution of the controlled system) state estimates for each controller so that they can take a better control decision. Since the (co-)reachability problem is undecidable in our settings, we rely on the abstract interpretation techniques of [22] to ensure the termination of the computations of our algorithms by overapproximating the possible FIFO channel contents (and hence the state estimates) by regular languages.

Related Works. Over the past years a considerable research effort has been done in decentralized supervisory control [35], [40], [34], [15] that allows to synthesize individual controllers that have a partial observation of the system's moves and can communicate with each other [34], [1], [24]. The pioneer work of Pnueli and Rosner [32] shows that the synthesis of distributed systems is in general undecidable. In [9], Gastin *et al.* study the decidability of LTL synthesis depending on the architecture of the distributed system. However, in these works the authors consider a synchronous architecture between the controllers. In [38], Tripakis studies the decidability of the existence of controllers such that a set of responsiveness properties is satisfied in a decentralized framework with communication delays between the controllers. He shows that

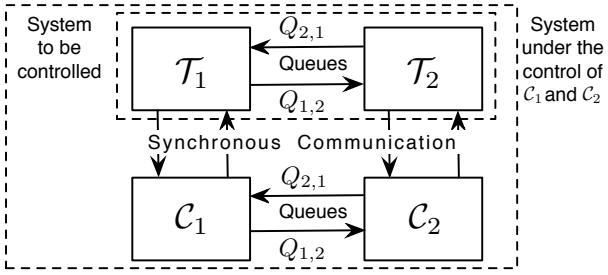

Fig. 1. Control architecture of a distributed system.

G. Kalyon and T. Massart are with Université Libre de Bruxelles (U.L.B.), Bruxelles, T. Le Gall is with CEA LIST, Saclay and H. marchand is with INRIA, centre Rennes - Bretagne Atlantique, Rennes

[1]By synchronous communication, we mean that the communication between controllers is instantaneous.

the problem is undecidable when there is no communication or when the communication delays are unbounded. In [13], Irasihi proves the decidability a decentralized control problem of discrete event systems with k-bounded-delay communication. In [2], Bensalem *et al.* propose a knowledge-based algorithm for distributed control: each subsystem is controlled according to a (local) knowledge of the property to ensure. When local knowledge is not sufficient, synchronizations are added until a decision can be taken (the reachability problem is decidable in their model). Unlike them, the reachability problem is undecidable in our model, the state estimates are a form of knowledge that does not depend on the property to ensure, and we never add synchronizations.

The control of concurrent systems is closely related to our framework [15], [10], [20], [23]. However, in this setting, the system is composed of several subsystems that communicate with zero delay (and similarly for the controllers) whereas in our approach, the subsystems and the controllers communicate asynchronously and we thus have to take into account the *a priori unbounded* communication delay to perform the computation of the controllers.

Our problem differs from the synthesis problem (see e.g. [26], [11]) which asks to synthesize a communication protocol and to distribute the actions of a specification depending on the subsystem where they must be executed, and to synchronize them in such a way that the resulting distributed system is equivalent to the given global specification.

In [7], Darondeau synthesizes distributed controllers for distributed system communicating by bounded channels. He states a sufficient condition allowing to decide if a controller can be implemented by a particular class of Petri nets that can be further translated into communicating automata. Some other works deal with the computation of a state estimate of a centralized system with distributed controllers. For example, in [39], Xu and Kumar propose a distributed algorithm which computes an estimate of the current state of a system. Local estimators maintain and update local state estimates from their own observation of the system and information received from the other estimators. In their framework, the local estimators communicate between them through reliable FIFO channels with delays, whereas the system is monolithic, and therefore these FIFO channels are not included into the global states of the system. Moreover, as we consider concurrent systems, we also have to take account the communication delay between sub-systems to compute the state-estimates as well as the control policies. Finally, compared with [39], we have chosen to exchange information between controllers using existing communication channel between subsystems. This renders the computation of the state-estimates completely different. Note also that the global state estimate problem of a distributed system is related to the problems of (Mazurkiewicz) *trace model checking* and *global predicate detection*; this later aims to see if there exists a possible global configuration of the system that satisfies a given global predicate ϕ. A lot of related works, consider an offline approach where the execution, given as a Mazurkiewicz trace [28] is provided from the beginning (see e.g. [12], [19] for a review and efficient methods). Online global predicate detection has been studied, e.g. in [14], [36]. The proposed solution implies a central monitor which receives on the fly the execution trace. Note that one of the main issues in these problems is to have a precise estimation on the sequences of events in the distributed execution. Therefore, standard techniques based e.g. on vector clocks [8], [27] are used to generate a partial ordering of events; and so does also our method. However, compared to the above mention works, our problem is particular for one or several reasons. First, the information must be received by all local controllers since no central monitor is present; then FIFO queues are part of the global states; finally these controllers must take proactive measures to prevent the system from taking an unsafe action.

Outline. The remainder of this paper is structured as follows. In section II, we present an overview of our control method. In section III, we define the formalism of *communicating finite state machines*, that we use to model distributed systems. We formally define, in section IV, the control mechanisms and the state avoidance control problem. In section V, we present an algorithm that computes estimates of the current state of a distributed system. In section VI, we define a control algorithm, using this state estimate algorithm, for our state avoidance control problem, and we explain how we can ensure the termination of this control algorithm by using abstract interpretation techniques. Section VII gives some experimental results.

Note. This paper is an extended version of two conference papers [18] and [17]. It provides the full process allowing to derive controllers from a state-based specification and a plant by means of state-based estimates and abstract interpretation techniques, whereas [17] was only presenting the state-based algorithms and [18] the control point of view with an overview of the state-based estimates computation point of view. The proofs absent from this paper are available in [16].

II. Overview of the Method

This section provides an informal presentation, through a running example, of the model, problem and main idea of our method.

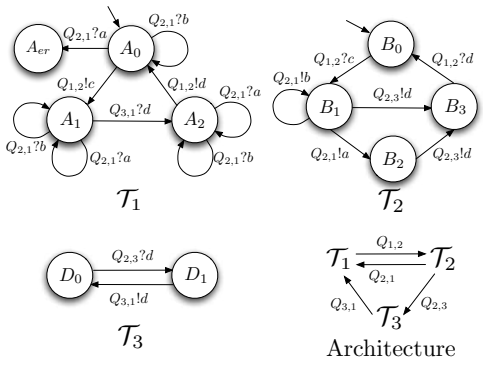

Fig. 2. Running example

Running Example. Figure 2 models a factory where three components $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ work together and communicate through four FIFO channels $Q_{1,2}$, $Q_{2,1}$, $Q_{2,3}$ and $Q_{3,1}$.

Subsystem $\mathcal{T}_2$ produces two kinds of items, a and b, and sends these items to $\mathcal{T}_1$ (action $\xrightarrow{Q_{2,1}!a}$) which must finish the job. At reception (action $\xrightarrow{Q_{2,1}?a}$), $\mathcal{T}_1$ must immediately take care of each received item. $\mathcal{T}_1$ can take care of b items at any time, but must be in *turbo mode* (locations A_1 and A_2) to take care of a items. When $\mathcal{T}_1$ receives an item a, in normal mode (location A_0), an error occurs (location A_{er}). Messages c and d help the communication between the different subsystems, by telling when $\mathcal{T}_1$ is in turbo mode and when $\mathcal{T}_2$ starts and stops to send items.

A state of the global system is naturally given by a tuple $\langle \ell_1, \ell_2, \ell_3, w_{1,2}, w_{2,1}, w_{2,3}, w_{3,1} \rangle$ where ℓ_i ($\forall i \in [1,3]$) gives the current location of the subsystem $\mathcal{T}_i$ and $w_{1,2}, w_{2,1}, w_{2,3}, w_{3,1}$ gives the content of the queues $Q_{1,2}, Q_{2,1}, Q_{2,3}, Q_{3,1}$. Let $Bad = \{\langle \ell_1, \ell_2, \ell_3, M^*, M^*, M^*, M^* \rangle \,|\, \ell_1 = A_{er}\}$ be the set of states we want to avoid, where $M = \{a, b, c, d\}$ is the set of messages (items in transit).

Computation of the Set of Forbidden Global States. The first step of our algorithm is to compute $I(Bad)$, the set of states that can lead to Bad by a sequence of uncontrollable transitions (input transitions). The only uncontrollable transition that leads to Bad is: $A_0 \xrightarrow{Q_{2,1}?a} A_{err}$, so the set of *forbidden global states* is: $I(Bad) = Bad \cup \{\langle \ell_1, \ell_2, \ell_3, M^*, b^*.a.M^*, M^*, M^* \rangle \,|\, \ell_1 = A_0\}$. The most permissive control policy is thus to disable the action $A_2 \xrightarrow{Q_{1,2}!d} A_0$ only when there is a message a in the channel $Q_{2,1}$. However, local controllers do not observe the content of FIFO channels. Therefore, the communication between local controllers must provide enough information to have a good knowledge of the content of FIFO channels.

State Estimates and Communication Between Controllers. This knowledge is given by some estimates of the current global state of the system. Each local controller has one state estimate to represent its knowledge and use it to define its control policy. The estimate of a controller $\mathcal{C}_i$ is mainly updated online by observing its local subsystem $\mathcal{T}_i$. Moreover, controllers can communicate with each other by adding their state estimate to the messages normally exchanged by the subsystems. In our example, when subsystem $\mathcal{T}_2$ sends message d to subsystem $\mathcal{T}_3$, its controller $\mathcal{C}_2$ knows whether a message a has been sent. $\mathcal{C}_3$ can then forward this information to $\mathcal{C}_1$. So, when $\mathcal{T}_1$ is in location A_2, its controller $\mathcal{C}_1$ knows whether there is a message a in $Q_{2,1}$ and it can then define the right control policy, i.e. it disables the transition $A_2 \xrightarrow{Q_{1,2}!d} A_0$ if and only if there is a message a in $Q_{2,1}$.

Effective Algorithm. The general control problem that we want to solve is *undecidable*. We then use abstract interpretation techniques to ensure, at the price of some overapproximations, that the computations of our control algorithm always terminate. In our case, we abstract queue contents by *regular languages*.

Discussion on the Model and the Method. The CFSM model we consider in this work is a theoretical framework that allow us to reason about control problems without considering the technical limitations of actual implementations of e.g. communication protocols[2]. Indeed, we consider unbounded FIFO channels since it is a useful abstraction to reason about communication protocols of asynchronous distributed systems without having to specify the size of the buffers. Therefore, our method gives valid results even when the FIFO are bounded. Our method also aims at computing an optimal knowledge (for each local controller) of the global state of the system. This allows local controllers to have the most permissive control strategy w.r.t. past communications (see section V). This knowledge (*state estimates*) includes a finite, symbolic representation of possible FIFO channels content. States estimates are piggy-backed to normal messages. This is both the main advantage and the main drawback of our method, since it leads to optimal state estimates but it also adds complex information to the original messages. While in our examples, messages are represented by single letters and state estimates seem to be more complex, in practice, actual messages can be bigger without increasing the size of state estimates. Therefore, the additional information may be proportionally quite small for protocols that transmit data packages like TCP/IP. Moreover, we suggest some ways to decrease the size of additional information at the end of this paper.

III. Model of the System

We model distributed systems by *communicating finite state machines* (CFSMs) [5] with reliable unbounded FIFO channels (also called *queues* below). CFSMs with unbounded channels are very useful to model and verify communication protocols, since we can reason on them without having to consider the actual size of the queues, which depend on the implementation of the protocol.

Model.

Definition 1 (Communicating Finite State Machines): A CFSM $\mathcal{T}$ *is defined by a 6-tuple* $\langle L, \ell_0, Q, M, \Sigma, \Delta \rangle$*, where* (i) L *is a finite set of locations,* (ii) $\ell_0 \in L$ *is the initial location,* (iii) Q *is a finite set of queues,* (iv) M *is a finite set of messages,* (v) $\Sigma \subseteq Q \times \{!, ?\} \times M$ *is a finite set of actions, which are either an* output $i!m$ *to specify that the message* $m \in M$ *is written on the queue* $i \in Q$ *or an* input $i?m$ *to specify that the message* $m \in M$ *is read on the queue* $i \in Q$*, and* (vi) $\Delta \subseteq L \times \Sigma \times L$ *is a finite set of transitions.*

An *output transition* $\langle \ell, i!m, \ell' \rangle$ indicates that, when the system moves from the location ℓ to ℓ', a message m must be added at the end of the queue i. An *input transition* $\langle \ell, i?m, \ell' \rangle$ indicates that, when the system moves from ℓ to ℓ', a message m must be present at the beginning of the queue i and must be removed from this queue. To simplify the presentation of our method, this model has no internal actions (i.e. events that are local to a subsystem and that are neither inputs nor outputs) and we assume that $\mathcal{T}$ is deterministic i.e., $\forall \ell \in L, \forall \sigma \in \Sigma : |\{\ell' \in L | \langle \ell, \sigma, \ell' \rangle \in \Delta\}| \leq 1$. Those restrictions are not mandatory and our implementation [29]

[2]As illustrated in this section, buffers can also be used to model place where items are stored in a manufacturing system waiting to be transformed by another machine (modeled by a sub-system of the CFSM)

accepts CFSMs with internal actions and non-deterministic ones. For $\sigma \in \Sigma$, the set of transitions of $\mathcal{T}$ labeled by σ is denoted by $\mathsf{Trans}(\sigma)$. An *event* e is the occurrence of a transition δ_e.

Semantics. A *global state* of a CFSM $\mathcal{T}$ is a tuple $\langle \ell, w_1, ..., w_{|Q|} \rangle \in X = L \times (M^*)^{|Q|}$ where ℓ is the current location of $\mathcal{T}$ and $w_1, ..., w_{|Q|}$ are finite words on M^* which give the content of the queues in Q.

Definition 2 (Semantics of a CFSM): The semantics of a CFSM $\mathcal{T} = \langle L, \ell_0, Q, M, \Sigma, \Delta \rangle$ is given by an LTS $[\![\mathcal{T}]\!] = \langle X, \vec{x}_0, \Sigma, \rightarrow \rangle$, where (i) $X \stackrel{\mathbf{def}}{=} L \times (M^*)^{|Q|}$ is the set of states, (ii) $\vec{x}_0 \stackrel{\mathbf{def}}{=} \langle \ell_0, \epsilon, \ldots, \epsilon \rangle$ is the initial state, (iii) Σ is the set of actions, and (iv) $\rightarrow \stackrel{\mathbf{def}}{=} \bigcup_{\delta \in \Delta} \stackrel{\delta}{\rightarrow} \subseteq X \times \Sigma \times X$ is the transition relation where $\stackrel{\delta}{\rightarrow}$ is defined as follows:

$$\frac{\delta = \langle \ell, i!m, \ell' \rangle \in \Delta \quad w_i' = w_i \cdot m}{\langle \ell, w_1, \ldots, w_i, \ldots, w_{|Q|} \rangle \xrightarrow{\delta} \langle \ell', w_1, \ldots, w_i', \ldots, w_{|Q|} \rangle}$$

$$\frac{\delta = \langle \ell, i?m, \ell' \rangle \in \Delta \quad w_i = m \cdot w_i'}{\langle \ell, w_1, \ldots, w_i, \ldots, w_{|Q|} \rangle \xrightarrow{\delta} \langle \ell', w_1, \ldots, w_i', \ldots, w_{|Q|} \rangle}$$

To simplify the notations, we often denote transition $\vec{x} \xrightarrow{\delta_e} \vec{x}'$ by $\vec{x} \xrightarrow{e} \vec{x}'$. An *execution* of $\mathcal{T}$ is a sequence $\vec{x}_0 \xrightarrow{e_1} \vec{x}_1 \xrightarrow{e_2} \ldots \xrightarrow{e_m} \vec{x}_m$ where $\vec{x}_0 = \langle \ell_0, \varepsilon, \cdots, \varepsilon \rangle$ is the only initial state and $\vec{x}_i \xrightarrow{e_{i+1}} \vec{x}_{i+1} \in \rightarrow \forall i \in [0, m-1]$. Given a set of states $Y \subseteq X$, $\mathsf{Reach}^{\mathcal{T}}_{\Delta'}(Y)$ corresponds to the set of states that are reachable in $[\![\mathcal{T}]\!]$ from Y only triggering transitions of $\Delta' \subseteq \Delta$ in $\mathcal{T}$, whereas $\mathsf{Coreach}^{\mathcal{T}}_{\Delta'}(Y)$ denotes the set of states from which Y is reachable only triggering transitions of Δ':

$$\mathsf{Reach}^{\mathcal{T}}_{\Delta'}(E) \stackrel{\mathbf{def}}{=} \bigcup_{n \geq 0} (\mathsf{Post}^{\mathcal{T}}_{\Delta'}(E))^n \tag{1}$$

$$\mathsf{Coreach}^{\mathcal{T}}_{\Delta'}(E) \stackrel{\mathbf{def}}{=} \bigcup_{n \geq 0} (\mathsf{Pre}^{\mathcal{T}}_{\Delta'}(E))^n \tag{2}$$

where $(\mathsf{Post}^{\mathcal{T}}_{\Delta'}(E))^n$ and $(\mathsf{Pre}^{\mathcal{T}}_{\Delta'}(E))^n$ are the n^{th} functional power of $\mathsf{Post}^{\mathcal{T}}_{\Delta'}(E) \stackrel{\mathbf{def}}{=} \{\vec{x}' \in X | \exists \vec{x} \in E, \exists \delta \in \Delta' : \vec{x} \xrightarrow{\delta} \vec{x}'\}$ and $\mathsf{Pre}^{\mathcal{T}}_{\Delta'}(E) \stackrel{\mathbf{def}}{=} \{\vec{x}' \in X | \exists \vec{x} \in E, \exists \delta \in \Delta' : \vec{x}' \xrightarrow{\delta} \vec{x}\}$. Although there is no general algorithm that can exactly compute the (co)reachability set [5], there exists a technique that allows us to compute an overapproximation of this set (see section VI-B). Given a sequence of actions $\overline{\sigma} = \sigma_1 \cdots \sigma_m \in \Sigma^*$ and two states $x, x' \in X$, $x \xrightarrow{\overline{\sigma}} x'$ denotes that the state x' is reachable from x by executing $\overline{\sigma}$.

Product of CFSM. A distributed system $\mathcal{T}$ is generally composed of several subsystems $\mathcal{T}_i$ ($\forall i \in [1..n]$) acting in parallel. In our case, this global system $\mathcal{T}$ is defined by a CFSM resulting from the product of the n subsystems $\mathcal{T}_i$, also modeled by CFSMs. This can be defined through the product of two subsystems.

Definition 3 (Product): Given two CFSMs $\mathcal{T}_i = \langle L_i, \ell_{0,i}, Q_i, M_i, \Sigma_i, \Delta_i \rangle$, their product, denoted by $\mathcal{T}_1 || \mathcal{T}_2$, is defined by a CFSM $\mathcal{T} = \langle L, \ell_0, Q, M, \Sigma, \Delta \rangle$, where (i) $L \stackrel{\mathbf{def}}{=} L_1 \times L_2$, (ii) $\ell_0 \stackrel{\mathbf{def}}{=} (\ell_{0,1}, \ell_{0,2})$, (iii) $Q \stackrel{\mathbf{def}}{=} Q_1 \cup Q_2$, (iv) $M \stackrel{\mathbf{def}}{=} M_1 \cup M_2$, (v) $\Sigma \stackrel{\mathbf{def}}{=} \Sigma_1 \cup \Sigma_2$, and (vi) $\Delta \stackrel{\mathbf{def}}{=} \{\langle \langle \ell_1, \ell_2 \rangle, \sigma_1, \langle \ell_1', \ell_2 \rangle \rangle | (\langle \ell_1, \sigma_1, \ell_1' \rangle \in \Delta_1) \wedge (\ell_2 \in L_2)\} \cup \{\langle \langle \ell_1, \ell_2 \rangle, \sigma_2, \langle \ell_1, \ell_2' \rangle \rangle | (\langle \ell_2, \sigma_2, \ell_2' \rangle \in \Delta_2) \wedge (\ell_1 \in L_1)\}$.
This operation is associative and commutative up to state renaming.

Definition 4 (Distributed system): A distributed system $\mathcal{T} = \langle L, \ell_0, Q, M, \Sigma, \Delta \rangle$ is defined by the product of n CFSMs $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1..n]$) acting in parallel and exchanging information through FIFO channels.
Note that a distributed system is also modeled by a CFSM, since the product of several CFSMs is a CFSM. To avoid the confusion between the model of one subsystem and the model of the whole system, in the sequel, a CFSM $\mathcal{T}_i$ always denotes the model of a single process, and a CFSM $\mathcal{T} = \langle L, \ell_0, Q, M, \Sigma, \Delta \rangle$ always denotes the distributed system $\mathcal{T} = \mathcal{T}_1 || \ldots || \mathcal{T}_n$.

Communication Architecture. We consider an architecture for the system $\mathcal{T} = \mathcal{T}_1 || \ldots || \mathcal{T}_n$ defined in Definition 4 with *point-to-point* communication i.e., any subsystem $\mathcal{T}_i$ can send messages to any other subsystem $\mathcal{T}_j$ through a queue[3] $Q_{i,j}$. Thus, only $\mathcal{T}_i$ can write a message m on $Q_{i,j}$ (denoted by $Q_{i,j}!m$) and only $\mathcal{T}_j$ can read m on this queue (denoted by $Q_{i,j}?m$). Moreover, we suppose that the queues are unbounded, that the message transfers between the subsystems are reliable and may suffer from arbitrary non-zero delays, and that no *global clock* or *perfectly synchronized local clocks* are available. With this architecture[4], the set Q_i of $\mathcal{T}_i$ ($\forall i \in [1..n]$) can be rewritten as $Q_i = \{Q_{i,j}, Q_{j,i} \,|\, (1 \leq j \leq n) \wedge (j \neq i)\}$ and $\forall j \neq i \in [1..n]$, $\Sigma_i \cap \Sigma_j = \emptyset$. Let $\delta_i = \langle \ell_i, \sigma_i, \ell_i' \rangle \in \Delta_i$ be a transition of $\mathcal{T}_i$, $\mathsf{global}(\delta_i) \stackrel{\mathbf{def}}{=} \{\langle \langle \ell_1, \ldots, \ell_{i-1}, \ell_i, \ell_{i+1}, \ldots, \ell_n \rangle, \sigma_i, \langle \ell_1, \ldots, \ell_{i-1}, \ell_i', \ell_{i+1}, \ldots, \ell_n \rangle \rangle \in \Delta \,|\forall j \neq i \in [1..n] : \ell_j \in L_j\}$ is the set of transitions of Δ that can be built from δ_i in $\mathcal{T}$. We extend this definition to sets of transitions $D \subseteq \Delta_i$ of the subsystem $\mathcal{T}_i$: $\mathsf{global}(D) \stackrel{\mathbf{def}}{=} \bigcup_{\delta_i \in D} \mathsf{global}(\delta_i)$. We abuse notation and write $\Delta \setminus \Delta_i$ instead of $\Delta \setminus \mathsf{global}(\Delta_i)$ to denote the set of transitions of Δ that are not built from Δ_i. Given the set Σ_i of $\mathcal{T}_i$ ($\forall i \in [1..n]$) and the set Σ of $\mathcal{T}$, the projection P_i of Σ onto Σ_i is standard: $P_i(\varepsilon) = \varepsilon$ and $\forall w \in \Sigma^*, \forall a \in \Sigma, P_i(wa) = P_i(w)a$ if $a \in \Sigma_i$, and $P_i(w)$ otherwise. The inverse projection P_i^{-1} is defined, for each $L \subseteq \Sigma_i^*$, by $P_i^{-1}(L) = \{w \in \Sigma^* \mid P_i(w) \in L\}$.

IV. Framework and State Avoidance Control Problem

In the sequel, we are interested in the state avoidance control problem which consists in preventing the system from reaching some undesirable states.

A. Control Architecture

The distributed system $\mathcal{T}$ is composed of n subsystems $\mathcal{T}_i$ ($\forall i \in [1..n]$) and we want to associate a local controller $\mathcal{C}_i$ with each subsystem $\mathcal{T}_i$ in order to satisfy the control

[3]To simplify the presentation of our method, we assume that there is one queue from $\mathcal{T}_i$ to $\mathcal{T}_j$. But, our implementation is more permissive and zero, one or more queues can exist from $\mathcal{T}_i$ to $\mathcal{T}_j$.

[4]In our examples, we do not mention queue $Q_{i,j}$ when there is no message sent from $\mathcal{T}_i$ to $\mathcal{T}_j$.

requirements. Each controller $\mathcal{C}_i$ interacts with $\mathcal{T}_i$ in a feedback manner: $\mathcal{C}_i$ observes the last action fired by $\mathcal{T}_i$ and computes, from this observation and some information received from the other controllers (corresponding to some state estimates), a set of actions that $\mathcal{T}_i$ cannot fire in order to ensure the desired properties on the global system. Following the Ramadge & Wonham's theory [33], the set of actions Σ_i of $\mathcal{T}_i$ is partitioned into the set of controllable actions $\Sigma_{i,c}$, that can be forbidden by $\mathcal{C}_i$, and the set of uncontrollable actions $\Sigma_{i,uc}$, that cannot be forbidden by $\mathcal{C}_i$. The subsets $\Sigma_{1,c}, \ldots, \Sigma_{n,c}$ are disjoint, because $\Sigma_i \cap \Sigma_j = \emptyset$ $(\forall i \neq j \in [1..n])$. In this paper and in our implementation [29], inputs are uncontrollable and outputs are controllable, a classical assumption for reactive systems. Our algorithm however does not depend on this particular partition of the actions, since one of its parameters is the set of uncontrollable actions. The set of actions, that can be controlled by at least one controller, is denoted by Σ_c and is defined by $\Sigma_c \stackrel{\text{def}}{=} \bigcup_{i=1}^{n} \Sigma_{i,c}$; We also define $\Sigma_{uc} \stackrel{\text{def}}{=} \Sigma \setminus \Sigma_c = \bigcup_{i=1}^{n} \Sigma_{i,uc}$. This cut also induces a partition on the set of transitions Δ_i into the sets $\Delta_{i,c}$ and $\Delta_{i,uc}$. The set of transitions Δ is similarly partitioned into the sets Δ_c and Δ_{uc}.

B. Distributed Controller and Controlled Execution

The control decision depends on the current state of the global system $\mathcal{T}$ (i.e. state-feedback control). Unfortunately, a local controller does not generally know the current global state, due to its partial observation of the system. So, it must define its control policy from a *state estimate* corresponding to its evaluation of the states the system $\mathcal{T}$ can possibly be. It is formally defined as follows:

Definition 5 (Local Controller): A local controller $\mathcal{C}_i$ *is a function* $\mathcal{C}_i : 2^X \rightarrow 2^{\Sigma_{i,c}}$ *which defines, for each estimate* $E \in 2^X$ *of the current state of* $\mathcal{T}$ *according to* $\mathcal{C}_i$*, the set of controllable actions that* $\mathcal{T}_i$ *may not execute.*

This definition of a controller does not explain how each local controller can compute a state estimate. In section V, we define an algorithm that allows $\mathcal{C}_i$ to compute this state estimate during the execution of this system. Note that besides the preciseness of the state estimate, one important property that should be satisfied by the state estimate E is that the actual current state of the system is in E.

Based on Definition 5, a *distributed controller* is defined by:

Definition 6 (Distributed Controller): A distributed controller $\mathcal{C}_{\mathsf{di}}$ *is defined by a tuple* $\mathcal{C}_{\mathsf{di}} \stackrel{\text{def}}{=} \langle \mathcal{C}_i \rangle_{i=1}^{n}$ *where* $\mathcal{C}_i$ $(\forall i \in [1..n])$ *is a local controller.*

A *controlled execution* is an execution that can occur in $\mathcal{T}$ under the control of $\mathcal{C}_{\mathsf{di}}$.

Definition 7 (Controlled Execution): Given a distributed controller $\mathcal{C}_{\mathsf{di}} = \langle \mathcal{C}_i \rangle_{i=1}^{n}$*,* $s = \vec{x}_0 \xrightarrow{e_1} \vec{x}_1 \xrightarrow{e_2} \ldots \xrightarrow{e_m} \vec{x}_m$ *is a* controlled execution *of* $\mathcal{T}$ *under the control of* $\mathcal{C}_{\mathsf{di}}$ *if* $\forall k \in [1, m]$*, whenever* $\delta_{e_k} \in \Delta_i$ *and the estimate of* $\mathcal{C}_i$ *of the current state* $\vec{x}_{k-1}$ *of* $\mathcal{T}$ *is* E*,* $\sigma_{e_k} \notin \mathcal{C}_i(E)$.

Note that with this definition, the language of the controlled system is controllable with respect to the language of the original system. It is basically due to the fact that each local controller is only able to disable the controllable actions that can occur in its corresponding subsystem.

C. Definition of the Control Problem

Control synthesis aims at restricting the behavior of a system to satisfy a goal property. The goal properties we consider are invariance properties, defined by a subset $Good \subseteq X$ of states, in which any execution of the transition system should be confined. Alternatively, it can be viewed as a state avoidance property $Bad = X \setminus Good$, which defines a set of states that no execution should reach. Notice that the specification Bad can involve the contents of the FIFO channels (recall that $X = L \times (M^*)^{|Q|}$). We define the problem as follows:

Problem 1 (Distributed State Avoidance Control Problem): Given a set $Bad \subseteq X$ *of forbidden states, the* distributed state avoidance control problem *(the* distributed problem *for short) consists in synthesizing a distributed controller* $\mathcal{C}_{\mathsf{di}} = \langle \mathcal{C}_i \rangle_{i=1}^{n}$ *such that each controlled execution of the system* $\mathcal{T}$ *under the control of* $\mathcal{C}_{\mathsf{di}}$ *avoids* Bad*.*

Proposition 1: Given a distributed systems $\mathcal{T}$*, a distributed controller* $\mathcal{C}_{\mathsf{di}}$ *and a set of forbidden states* $Bad \subseteq X$*, it is undecidable to know whether* $\mathcal{C}_{\mathsf{di}}$ *solves Problem 1. Moreover, deciding the existence of a non-trivial controller* $\mathcal{C}_{\mathsf{di}}$ *solving Problem 1 is undecidable.*

Intuitively, this result is a consequence of the undecidability of the (co-)reachability problem in the CFSM model [5].

Remark 1 (Trivial solution and the non-blocking problem): Definition of Problem 1 does not tackle the non-blocking problem (i.e. by imposing that at every time at least one transition of one of the subsystem is allowed). Therefore, there exists a trivial solution of this problem, which consists in disabling all output transitions so that nothing happens in the controlled system. However, our aim is to find, as often as possible, solutions that are correct and enough permissive to be of practical value. Since the principle of safe control is to allow a transition only when the local controller is sure this transition cannot lead to a bad state, permissiveness directly depends on the knowledge local controllers have about the global system.

V. State Estimates of Distributed Systems

In this section, we present an algorithm that computes estimates of the current state of a distributed system. The result of this algorithm is used, in section VI, by our control algorithm which synthesizes distributed controllers for the distributed problem. We first recall the notion of *vector clocks* [21], a standard concept that we use to compute state estimates.

A. Vector Clocks

To allow the local controllers to have a better understanding of the execution of the distributed system, it is important to determine the causal and temporal relationship between the events that occur during the execution : events emitted by a same subsystem are ordered, while events emitted by different subsystems are generally not. When the concurrent subsystems communicate, additional ordering information can be obtained,

and the communication scheme can be used to obtain a partial order on the events of the system. In practice, vectors of logical clocks, called *Vector clocks* [21], can be used to time-stamp the events of a distributed system. The order of the vector clocks induces the order of the corresponding events. Vector clocks are formally defined as follows:

Definition 8 (Vector Clocks): Let $\langle D, \sqsubseteq \rangle$ be a partially ordered set, a vector clock mapping *of width n is a function $V : D \mapsto \mathbb{N}^n$ such that $\forall d_1, d_2 \in D : (d_1 \sqsubseteq d_2) \Leftrightarrow (V(d_1) \leq V(d_2))$.*

In general, for a distributed system composed of n subsystems, the partial order on events is represented by a vector clock mapping of width n. The method for computing this vector clock mapping depends on the communication scheme of the distributed system. For CFSMs, it can be computed by the Mattern's algorithm [27], which is based on the causal and thus temporal relationship between the sending and reception of any message transferred through any FIFO channel. This information is then used to determine a partial order, called *causality (or happened-before) relation* $\prec_c$, on the events of the distributed system. This relation is the smallest transitive relation satisfying the following conditions: (i) if the events $e_i \neq e_j$ occur in the same subsystem $\mathcal{T}_i$ and if e_i comes before e_j in the execution, then $e_i \prec_c e_j$, and (ii) if e_i is an output event occurring in $\mathcal{T}_i$ and if e_j is the corresponding input event occurring in $\mathcal{T}_j$, then $e_i \prec_c e_j$. In the sequel, when $e_i \prec_c e_j$, we say that e_j *causally depends* on e_i (or e_i *happened-before* e_j).

In Mattern's algorithm [27], each subsystem $\mathcal{T}_i$ $(\forall i \in [1..n])$ has a vector clock $V_i \in \mathbb{N}^n$. Each element $V_i[j]$ $(\forall j \in [1..n])$ is a counter which represents the knowledge of $\mathcal{T}_i$ regarding T_j and which can roughly be interpreted as follows: $\mathcal{T}_i$ knows that $\mathcal{T}_j$ has executed at least $V_i[j]$ events. Initially, each component of the vector V_i $(\forall i \in [1..n])$ is set to 0. Next, when an event e occurs in $\mathcal{T}_i$, the vector clock V_i is updated as follows: first, $V_i[i]$ is incremented (i.e., $V_i[i] \leftarrow V_i[i] + 1$) to indicate that a new event occurred in $\mathcal{T}_i$ and next two cases are considered:

- if e consists in sending message m to $\mathcal{T}_j$, vector clock V_i is attached to m and both information are sent to $\mathcal{T}_j$.
- if e corresponds to the reception of message m tagged with vector clock V_j, then V_i is set to the component-wise maximum of V_i and V_j. This allows us to take into account the fact that any event, that precedes the sending of m, should also precede the event e.

We now define a lemma related to vector clocks that will be used in the sequel:

Lemma 1: Given a sequence $se_1 = \vec{x}_0 \xrightarrow{e_1} \vec{x}_1 \xrightarrow{e_2} \ldots \xrightarrow{e_{i-1}} \vec{x}_{i-1} \xrightarrow{e_i} \vec{x}_i \xrightarrow{e_{i+1}} \vec{x}_{i+1} \xrightarrow{e_{i+2}} \ldots \xrightarrow{e_m} \vec{x}_m$ executed by $\mathcal{T}$, if $e_i \not\prec_c e_{i+1}$, then the sequence $se_2 = \vec{x}_0 \xrightarrow{e_1} \vec{x}_1 \xrightarrow{e_2} \ldots \xrightarrow{e_{i-1}} \vec{x}_{i-1} \xrightarrow{e_{i+1}} \vec{x}'_i \xrightarrow{e_i} \vec{x}_{i+1} \xrightarrow{e_{i+2}} \ldots \xrightarrow{e_m} \vec{x}_m$ can also occur in $\mathcal{T}$.

This property means that if two consecutive events e_i and e_{i+1} are such that $e_i \not\prec_c e_{i+1}$, then these events can be swapped without modifying the reachability of $\vec{x}_m$.

B. Computation of State Estimates

Each time an event occurs in subsystem $\mathcal{T}_i$, controller $\mathcal{C}_i$ updates its vector clock V_i and its state estimate E_i that should contain the current state of $\mathcal{T}$. Note that E_i must also contain any future state that can be reached from this current state by firing actions that do not belong to $\mathcal{T}_i$. Our state estimate algorithm proceeds as follows :

- When $\mathcal{T}_i$ sends a message m to $\mathcal{T}_j$, $\mathcal{T}_i$ attaches the vector clock V_i and the state estimate E_i of $\mathcal{C}_i$ to this message. Next, $\mathcal{C}_i$ observes the action fired by $\mathcal{T}_i$, and infers the fired transition. It then uses this information to update its state estimate E_i.
- When $\mathcal{T}_i$ receives a message m from $\mathcal{T}_j$, $\mathcal{C}_i$ observes the action fired by $\mathcal{T}_i$ and the information sent by $\mathcal{T}_j$ i.e., the state estimate E_j and the vector clock V_j of $\mathcal{C}_j$. It computes its new state estimate from these elements.

In both cases, the computation of the new state estimate E_i depends on the computation of reachable states. In this section, we assume that we have an operator that can compute an *approximation* of the reachable states. We explain in section VI how to compute this operator.

State Estimate Algorithm. Our algorithm, called *SE-algorithm*, computes state estimates of a distributed system. It is composed of three sub-algorithms: (i) the initialEstimate algorithm, which is only used when the system starts its execution, computes, for each controller, its initial state estimate (ii) the outputTransition algorithm computes online the new state estimate of $\mathcal{C}_i$ after an output of $\mathcal{T}_i$, and (iii) the inputTransition algorithm computes online the new state estimate of $\mathcal{C}_i$ after an input of $\mathcal{T}_i$.

initialEstimate *Algorithm:* Each component of the vector V_i is set to 0. To take into account that, before the execution of the first action of $\mathcal{T}_i$, the other subsystems $\mathcal{T}_j$ $(\forall j \neq i \in [1..n])$ could perform inputs and outputs, the initial state estimate of $\mathcal{C}_i$ is given by $E_i = \mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_i}(\langle \ell_{0,1}, \ldots, \ell_{0,n}, \epsilon, \ldots, \epsilon \rangle)$.

outputTransition *Algorithm:* Let E_i be the current state estimate of $\mathcal{C}_i$. When $\mathcal{T}_i$ fires an output transition $\delta = \langle \ell_1, Q_{i,j}!m, \ell_2 \rangle \in \Delta_i$, the following instructions are computed

Algorithm 1: initialEstimate($\mathcal{T}$)

input : $\mathcal{T} = \mathcal{T}_1 || \ldots || \mathcal{T}_n$.
output: The initial state estimate E_i of the controller $\mathcal{C}_i$ $(\forall i \in [1..n])$.

begin
  for $i \leftarrow 1$ ***to*** n **do** **for** $j \leftarrow 1$ ***to*** n **do** $V_i[j] \leftarrow 0$
  for $i \leftarrow 1$ ***to*** n **do**
    $E_i \leftarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_i}(\langle \ell_{0,1}, \ldots, \ell_{0,n}, \epsilon, \ldots, \epsilon \rangle)$
end

Algorithm 2: outputTransition($\mathcal{T}, V_i, E_i, \delta$)

input : $\mathcal{T} = \mathcal{T}_1 || \ldots || \mathcal{T}_n$, the vector clock V_i of $\mathcal{C}_i$, the current state estimate E_i of $\mathcal{C}_i$, and a transition $\delta = \langle \ell_1, Q_{i,j}!m, \ell_2 \rangle \in \Delta_i$.
output: The state estimate E_i after the output transition δ.

begin
  $V_i[i] \leftarrow V_i[i] + 1$
  $\mathcal{T}_i$ tags message m with $\langle E_i, V_i, \delta \rangle$ and writes this tagged message on $Q_{i,j}$
  $E_i \leftarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta \setminus \Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta}(E_i))$
end

to update the state estimate E_i:

- $V_i[i]$ is incremented (i.e., $V_i[i] \leftarrow V_i[i]+1$) to indicate that a new event has occurred in $\mathcal{T}_i$.
- $\mathcal{T}_i$ tags message m with $\langle E_i, V_i, \delta\rangle$ and writes this information on $Q_{i,j}$. The estimate E_i, tagging m, contains the set of states in which $\mathcal{T}$ can be *before* the execution of δ. The additional information $\langle E_i, V_i, \delta\rangle$ will be used by $\mathcal{T}_j$ to refine its state estimate.
- E_i is updated as follows to contain the current state of $\mathcal{T}$ and any future state that can be reached from this current state by firing actions that do not belong to $\mathcal{T}_i$: $E_i \leftarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta}(E_i))$. More precisely, $\mathsf{Post}^{\mathcal{T}}_{\delta}(E_i)$ gives the set of states in which $\mathcal{T}$ can be after the execution of δ. But, after the execution of this transition, $\mathcal{T}_j$ ($\forall j \neq i \in [1..n]$) could read and write on their queues. Therefore, we define the estimate E_i by $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta}(E_i))$.

Algorithm 3: inputTransition$(\mathcal{T}, V_i, E_i, \delta)$

input : $\mathcal{T} = \mathcal{T}_1||\ldots||\mathcal{T}_n$, the vector clock V_i of $\mathcal{C}_i$, the current state estimate E_i of $\mathcal{C}_i$ and a transition $\delta = \langle \ell_1, Q_{j,i}?m, \ell_2\rangle \in \Delta_i$. Message m is tagged with the triple $\langle E_j, V_j, \delta'\rangle$ where (i) E_j is the state estimate of $\mathcal{C}_j$ before the execution of δ' by $\mathcal{T}_j$, (ii) V_j is the vector clock of $\mathcal{C}_j$ after the execution of δ' by $\mathcal{T}_j$, and (iii) $\delta' = \langle \ell'_1, Q_{j,i}!m, \ell'_2\rangle \in \Delta_j$ is the output corresponding to δ.

output: The state estimate E_i after the input transition δ.

```
1  begin
2    \\ We consider three cases to update E_j
3    if V_j[i] = V_i[i] then
       Temp ← Post^T_δ(Reach^T_{Δ\Δ_i}(Post^T_{δ'}(E_j)))
4    else if V_j[j] > V_i[j] then
       Temp ← Post^T_δ(Reach^T_{Δ\Δ_i}(Reach^T_{Δ\Δ_j}(Post^T_{δ'}(E_j))))
5    else Temp ← Post^T_δ(Reach^T_Δ(Post^T_{δ'}(E_j)))
6    E_i ← Post^T_δ(E_i) \\ We update E_i
7    E_i ← E_i ∩ Temp \\ E_i = update of E_i ∩ update of E_j (i.e., Temp)
8    V_i[i] ← V_i[i] + 1
9    for k ← 1 to n do V_i[k] ← max(V_i[k], V_j[k])
10 end
```

inputTransition *Algorithm:* Let E_i be the current state estimate of $\mathcal{C}_i$. When $\mathcal{T}_i$ fires an input transition $\delta = \langle \ell_1, Q_{j,i}?m, \ell_2\rangle \in \Delta_i$, it also reads the information $\langle E_j, V_j, \delta'\rangle$ (where E_j is the state estimate of $\mathcal{C}_j$ before the execution of δ' by $\mathcal{T}_j$, V_j is the vector clock of $\mathcal{C}_j$ after the execution of δ' by $\mathcal{T}_j$, and $\delta' = \langle \ell'_1, Q_{j,i}!m, \ell'_2\rangle \in \Delta_j$ is the output corresponding to δ) tagging m, and the following operations are performed to update E_i:

- we update the state estimate E_j of $\mathcal{C}_j$ (this update is stored in $Temp$) by using the vector clocks to guess the possible behaviors of $\mathcal{T}$ between the execution of the transition δ' and the execution of δ. We consider three cases :
  - if $V_j[i] = V_i[i]$: $Temp \leftarrow \mathsf{Post}^{\mathcal{T}}_{\delta}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta'}(E_j)))$. In this case, thanks to the vector clocks, we know that $\mathcal{T}_i$ has executed no transition between the execution of δ' by $\mathcal{T}_j$ and the execution of δ by $\mathcal{T}_i$. Thus, only transitions in $\Delta \setminus \Delta_i$ could have occurred during this period. We then update E_j as follows. We compute (i) $\mathsf{Post}^{\mathcal{T}}_{\delta'}(E_j)$ to take into account the execution of δ' by $\mathcal{T}_j$, (ii) $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta'}(E_j))$ to take into account the transitions that could occur between the execution of δ' and the execution of δ, and (iii) $\mathsf{Post}^{\mathcal{T}}_{\delta}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta'}(E_j)))$ to take into account the execution of δ.
  - else if $V_j[j] > V_i[j]$: $Temp \leftarrow \mathsf{Post}^{\mathcal{T}}_{\delta}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\mathsf{Post}^{\mathcal{T}}_{\delta'}(E_j))))$. Indeed, in this case, we can prove (see Theorem 1) that if we reorder the transitions executed between the occurrence of δ' and the occurrence of δ in order to execute the transitions of Δ_i before the ones of Δ_j, we obtain a correct update of E_i. Intuitively, this reordering is possible, because there is no causal relation between the events of $\mathcal{T}_i$ and the events of $\mathcal{T}_j$, that have occurred between δ' and δ. So, in this reordered sequence, we know that, after the execution of δ, only transitions in $\Delta \setminus \Delta_j$ could occur followed by transitions in $\Delta \setminus \Delta_i$.
  - else $Temp \leftarrow \mathsf{Post}^{\mathcal{T}}_{\delta}(\mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta'}(E_j)))$. Indeed, in this case, the vector clocks do not allow us to deduce information regarding the behavior of $\mathcal{T}$ between the execution of δ' and the execution of δ. Therefore, to have a correct state estimate, we update E_j by taking into account all the possible behaviors of $\mathcal{T}$ between the execution of δ' and the execution of δ.
- we update the estimate E_i to take into account the execution of δ: $E_i \leftarrow \mathsf{Post}^{\mathcal{T}}_{\delta}(E_i)$.
- we intersect $Temp$ and E_i to obtain a better state estimate: $E_i \leftarrow E_i \cap Temp$.
- vector clock V_i is incremented to take into account the execution of δ and subsequently is set to the component-wise maximum of V_i and V_j. This last operation allows us to take into account the fact that any event that precedes the sending of m should also precede the occurrence of δ.

C. Properties

State estimate algorithms should have two important properties: soundness and completeness. Completeness means that the current state of the global system is always included in the state estimates computed by each controller. Soundness means that all states included in the state estimate of $\mathcal{C}_i$ ($\forall i \in [1..n]$) can be reached by one of the sequences of actions that are compatible with the local observation of $\mathcal{T}_i$.

We first introduce some additional notations and a lemma used in the proof of Theorem 1. Let $s = \vec{x}_0 \xrightarrow{e_1} \vec{x}_1 \xrightarrow{e_2} \ldots \xrightarrow{e_m} \vec{x}_m$ be an execution of $\mathcal{T}$. When an event e_k is executed in the sequence s, the state estimate of *each* controller $\mathcal{C}_i$ is denoted by E^k_i. This state estimate is defined in the following way: if e_k has not been executed by $\mathcal{T}_i$, then $E^k_i \stackrel{\text{def}}{=} E^{k-1}_i$. Otherwise, E^k_i is computed by $\mathcal{C}_i$ according to Algorithm 2 or 3.

Lemma 2: Given a transition $\delta_i = \langle \ell_i, Q_{t,i}?m_i, \ell'_i\rangle \in \Delta_i$ (with $t \neq i$), and a set of states $B \subseteq X$, then $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(B))) = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_i}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(B))$.

Theorem 1: SE-algorithm is complete if the Reach *operator computes an overapproximation of the reachable states. In other words, SE-algorithm satisfies the following property: for any execution* $\vec{x}_0 \xrightarrow{e_1} \vec{x}_1 \xrightarrow{e_2} \dots \xrightarrow{e_m} \vec{x}_m$ *of* $\mathcal{T}$, $\vec{x}_m \in \bigcap_{i=1}^{n} E_i^m$.

Proof (Sketch): We prove[5] this theorem by showing, by induction on the length m of an execution $\vec{x}_0 \xrightarrow{e_1} \vec{x}_1 \xrightarrow{e_2} \dots \xrightarrow{e_m} \vec{x}_m$ of $\mathcal{T}$, that $\forall i \in [1..n] : \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_m) \subseteq E_i^m$. By abuse of notation, we identify a state $\vec{x}_m$ and the singleton $\{\vec{x}_m\}$ in the proofs. Since $\vec{x}_m \in \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_m)$, we have that $\vec{x}_m \in E_i^m$.

- **Base case ($m = 0$):** According to Algorithm 1, $\forall i \in [1..n] : E_i^0 = \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_0)$.
- **Induction step:** We suppose that the property holds for $k \leq m$ and we prove that $\forall j \in [1..n] : \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\vec{x}_{m+1}) \subseteq E_j^{m+1}$. For that, we suppose that the event e_{m+1} has been executed by $\mathcal{T}_i$ and we consider two cases:

1) $\delta_{e_{m+1}}$ is an output on the queue $Q_{i,k}$ (with $k \neq i \in [1..n]$): We consider two sub-cases:

a) $j = i$: We know that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_m) \subseteq E_i^m$ (induction hypothesis) and the set $E_i^{m+1} = \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_i^m))$ (see Algorithm 2). Moreover, we have that:

$\vec{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_m)$
$\Rightarrow \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\vec{x}_m) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_m))$
$\Rightarrow \vec{x}_{m+1} \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_m))$, as $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\vec{x}_m) = \vec{x}_{m+1}$
$\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_{m+1}) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_m)))$
$\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_{m+1}) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_i^m))$, by induction hypothesis
$\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_{m+1}) \subseteq E_i^{m+1}$, by definition of E_i^{m+1}

b) $j \neq i$: we prove the property by induction as in the previous case.

Note that since we compute an overapproximation of E_j^{m+1} ($\forall j \in [1..n]$), the inclusion we proved remains true[6].

2) $\delta_{e_{m+1}}$ is an input from the queue $Q_{k,i}$ (with $k \neq i \in [1..n]$): Again, we consider two sub-cases:

a) $j = i$: By Algorithm 3, the set $E_i^{m+1} = Temp \cap \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_i^m)$ (in our algorithm, the set $Temp$ can have three possible values). To prove that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_{m+1}) \subseteq E_i^{m+1}$, we first prove that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_{m+1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_i^m)$ and next we show that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_{m+1}) \subseteq Temp$. The first inclusion is proved as follows:

$\vec{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_m)$
$\Rightarrow \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\vec{x}_m) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_m))$
$\Rightarrow \vec{x}_{m+1} \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_m))$, as $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\vec{x}_m) = \vec{x}_{m+1}$
$\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_{m+1}) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_m)))$
$\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_{m+1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_m))$, by Lemma 2
$\Rightarrow \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_{m+1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_i^m)$, by induction hypothesis

To prove the second inclusion, we must consider three possibilities which depend on the definition of $Temp$. Let e_t (with $t \leq m$) be the output (executed by $\mathcal{T}_k$ with $k \neq i \in [1..n]$) corresponding to the input e_{m+1}:

A) $Temp = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})))$ and $V_k[i] = V_i[i]$ (as a reminder, V_k represents the vector clock of $\mathcal{T}_k$ after the occurrence of the event e_t and V_i represents the vector clock of $\mathcal{T}_i$ before the occurrence of the event e_{m+1}): We first prove that

$$\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_t) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})) \quad (3)$$

Next, since $V_k[i] = V_i[i]$, we know that, between the moment where e_t has been executed and the moment where e_m has been executed, the vector clock $V_i[i]$ has not been modified. Thus, during this period no transition of $\mathcal{T}_i$ has been executed. In consequence, we have that $\vec{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_t)$ and hence $\vec{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}))$ by (3). Finally, from this inclusion, we can deduce that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_{m+1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})))$, which proves the property.

B) $Temp = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}))))$ and $V_k[k] > V_i[k]$: first, we prove that:

$$\vec{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\vec{x}_t)) \quad (4)$$

For that, we consider the subsequence $se = \vec{x}_t \xrightarrow{e_{t+1}} \vec{x}_{t+1} \xrightarrow{e_{t+2}} \dots \xrightarrow{e_m} \vec{x}_m$ of the execution $\vec{x}_0 \xrightarrow{e_1} \vec{x}_1 \xrightarrow{e_2} \dots \xrightarrow{e_m} \vec{x}_m$, and we show that se can be reordered to obtain a new sequence where the events of $\mathcal{T}_i$ are executed before the ones of $\mathcal{T}_k$ and where $\vec{x}_m$ remains reachable. To prove that such a reordered sequence can be obtained we first prove that the events in se executed by $\mathcal{T}_k$ do not causally depend on the events in se executed by $\mathcal{T}_i$. Then we use Lemma 1, that allows us to swap two consecutive events without modifying the reachability when these events are not causally dependent, to reorder the events of $\mathcal{T}_i$ and $\mathcal{T}_k$. Finally, from (4), we can deduce that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_{m+1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_k}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}))))$.

C) $Temp = \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})))$: first, we prove that:

$$\mathsf{Reach}^{\mathcal{T}}_{\Delta}(\vec{x}_t) \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})) \quad (5)$$

[5]The proofs of Theorems 1 and 2 are quite technical and composed of several cases. In the sketch of these proofs, we present the different cases: for the first ones, we fully explain the techniques and the approaches used to solve them, but for the last ones, we are more concise, since they are based on similar resolution methods. We proceed in this way to give the intuition of the complete resolution of the proof.

[6]Note that if we compute an underapproximation of E_j^{m+1}, the inclusion does not always hold.

Next, since the events $e_{t+1}, \ldots, e_m$ leading to $\vec{x}_m$ from the state $\vec{x}_t$ correspond to transitions which belong to Δ we have that $\vec{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta}(\vec{x}_t)$ and hence $\vec{x}_m \subseteq \mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1}))$ by (5). Finally, from this inclusion, we can deduce that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_{m+1}) \subseteq \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(\mathsf{Reach}^{\mathcal{T}}_{\Delta}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_t}}(E_k^{t-1})))$.

Thus, for each definition of $Temp$, we have that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_{m+1}) \subseteq Temp$ and hence $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_{m+1}) \subseteq E_i^{m+1}$.

b) $j \neq i$: The proof is similar to the one given in the case where $\delta_{e_{m+1}}$ is an output.

Thus, for each $j \in [1..n]$, we have that $\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\vec{x}_{m+1}) \subseteq E_j^{m+1}$. Moreover, since we compute an overapproximation of E_j^{m+1} ($\forall j \in [1..n]$), this inclusion remains true.

■

Theorem 2: SE-algorithm is sound if the Reach *operator computes an underapproximation of the reachable states. In other words, SE-algorithm satisfies the following property: for any execution $\vec{x}_0 \xrightarrow{e_1} \vec{x}_1 \xrightarrow{e_2} \ldots \xrightarrow{e_m} \vec{x}_m$ of $\mathcal{T}$, $E_i \subseteq \{x' \in X | \exists \overline{\sigma} \in P_i^{-1}(P_i(\sigma_{e_1}.\sigma_{e_2}\ldots\sigma_{e_m})) : \vec{x}_0 \xrightarrow{\overline{\sigma}} x'\}$ ($\forall i \leq n$) where $\forall k \in [1,m]$, σ_{e_k} is the action that labels the transition corresponding to e_k.*

Proof (Sketch): We prove by induction on the length m of the sequences of events $e_1, \ldots, e_m$ executed by the system that $\forall i \in [1..n] : E_i^m \subseteq \{x_r \in X | \exists \overline{\sigma} \in P_i^{-1}(P_i(\sigma_{e_1}.\sigma_{e_2}\ldots\sigma_{e_m})) : \vec{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$ where $\delta_{e_k} = \langle \ell_{e_k}, \sigma_{e_k}, \ell'_{e_k} \rangle$ is the transition corresponding to e_k, for each $k \in [1,m]$:

- **Base case ($m = 0$):** It is proved by showing that $\forall i \in [1..n] : E_i^0 = \{x_r \in X | \exists \overline{\sigma} \in P_i^{-1}(P_i(\epsilon)) : \vec{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$.
- **Induction step:** We suppose that the property holds for $k \leq m$ and we prove that $\forall j \in [1..n] : E_j^{m+1} \subseteq \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\ldots\sigma_{e_{m+1}})) : \vec{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$. We suppose that e_{m+1} has been executed by $\mathcal{T}_i$ and we consider two cases:
  1) $\delta_{e_{m+1}}$ is an output: We consider two sub-cases:
     a) $i \neq j$: The property is proved from the induction hypothesis $E_j^m \subseteq \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\ldots\sigma_{e_m})) : \vec{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$ by using the fact that $E_j^{m+1} = E_j^m$ (since $\mathcal{C}_j$ does not update its state estimate) and that $P_j(\sigma_{e_1}.\sigma_{e_2}\ldots\sigma_{e_m}) = P_j(\sigma_{e_1}.\sigma_{e_2}\ldots\sigma_{e_{m+1}})$, because $\sigma_{e_{m+1}} \notin \Sigma_j$.
     b) $i = j$: We have to prove that $E_j^{m+1} = \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_j^m)) \subseteq \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\ldots\sigma_{e_{m+1}})) : \vec{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$. This can be done by showing that if a state $\vec{x} \in \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_j}(\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_j^m))$, $\vec{x} \in \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\ldots\sigma_{e_{m+1}})) : \vec{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$.

     Note that since we compute an underapproximation of E_j^{m+1}, the inclusion we proved remains true.
  2) $\delta_{e_{m+1}}$ is an input: We consider again two sub-cases. For the first case (i.e., $i \neq j$), the proof is similar to the one given in the case where $\delta_{e_{m+1}}$ is an output. For the second case (i.e., $i = j$), we must prove that $\mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_j^m) \cap Temp \subseteq \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\ldots\sigma_{e_{m+1}})) : \vec{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$ (see Algorithm 3). This can be done by showing that if a state $\vec{x} \in \mathsf{Post}^{\mathcal{T}}_{\delta_{e_{m+1}}}(E_j^m)$, then $\vec{x} \in \{x_r \in X | \exists \overline{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1}.\sigma_{e_2}\ldots\sigma_{e_{m+1}})) : \vec{x}_0 \xrightarrow{\overline{\sigma}} x_r\}$. Again, since we compute an underapproximation of E_j^{m+1}, the inclusion remains true.

■

If we compute an underapproximation of the reachable states, our state estimate algorithm is sound but not complete. If we compute an overapproximation of the reachable states, our state estimate algorithm is complete but not sound. Since we only need completeness to solve the control problem, we define in section VI an effective algorithm for the distributed problem by computing overapproximations of the reachable states.

VI. Computation by Means of Abstract Interpretation of Distributed Controllers for the Distributed Problem

In this section, we first define a semi-algorithm for the distributed problem which uses SE-algorithm as sub-algorithm. Next, we explain how to extend it by using abstract interpretation techniques to obtain an effective algorithm.

A. Semi-Algorithm for the Distributed Problem

Our algorithm, which synthesizes a distributed controller $\mathcal{C}_{\mathsf{di}}$ for the distributed problem, is composed of two parts:

- **Offline part:** We compute the set $I(Bad)$ of states of the global system $\mathcal{T}$ that can lead to Bad by a sequence of uncontrollable transitions. Next, we compute, for each local controller $\mathcal{C}_i$, a control function $\mathcal{F}_i$ which gives, for each action σ of $\mathcal{T}_i$, the set of states of $\mathcal{T}$ that can lead to $I(Bad)$ by a transition labeled by σ. This information is used by $\mathcal{C}_i$, in the online part, to define its control policy.
- **Online part:** During the execution of $\mathcal{T}$, each local controller $\mathcal{C}_i$ uses the SE-algorithm to obtain its own state estimate E_i and computes from this information the actions to be forbidden.

These two parts are formalized as follows.

Offline Part. The set $I(Bad)$ of states of $\mathcal{T}$ leading uncontrollably to Bad is given by $\mathsf{Coreach}^{\mathcal{T}}_{\Delta_{uc}}(Bad)$ which, as a reminder, is defined by $\mathsf{Coreach}^{\mathcal{T}}_{\Delta_{uc}}(Bad) = \bigcup_{n\geq 0}(\mathsf{Pre}^{\mathcal{T}}_{\Delta_{uc}})^n(Bad)$ (see (2)). Alternatively, it is defined as the least fixpoint of the function $\lambda B.Bad \cup \mathsf{Pre}^{\mathcal{T}}_{\Delta_{uc}}(B)$. Since this function is continuous as a composition of continuous functions, the Knaster-Tarski and Kleene's theorems [37], [25] ensure that the least fixpoint exists, so $I(Bad) = \mathsf{Coreach}^{\mathcal{T}}_{\Delta_{uc}}(Bad)$.

Next, we define, for each local controller $\mathcal{C}_i$, the control function $\mathcal{F}_i : \Sigma_i \times 2^X \to 2^X$, which gives, for each action $\sigma \in \Sigma_i$ and set $B \subseteq X$ of states to be forbidden, the set $\mathcal{F}_i(\sigma, B)$ of global states in which the action σ must be forbidden. This set corresponds, more precisely, to the greatest set $\mathcal{O}$ of states

of $\mathcal{T}$ such that, for each state $\vec{x} \in \mathcal{O}$, there exists a transition labeled by σ leading to B from $\vec{x}$:

$$\mathcal{F}_i(\sigma, B) \stackrel{\text{def}}{=} \begin{cases} \mathsf{Pre}^{\mathcal{T}}_{\mathsf{Trans}(\sigma)}(B) & \text{if } \sigma \in \Sigma_{i,c} \\ \emptyset & \text{otherwise} \end{cases} \quad (6)$$

We compute, for each action $\sigma \in \Sigma_i$, the set $\mathcal{F}_i(\sigma, I(Bad))$ ($\forall i \in [1..n]$). This information is used, during the execution of $\mathcal{T}$, by the local controller $\mathcal{C}_i$ to compute the actions to be forbidden.

Online Part. The local controller $\mathcal{C}_i$ is formally defined, for each state estimate $E \in 2^X$, by:

$$\mathcal{C}_i(E) \stackrel{\text{def}}{=} \{\sigma \in \Sigma_i \mid \mathcal{F}_i(\sigma, I(Bad)) \cap E \neq \emptyset\} \quad (7)$$

Thus, if E is the state estimate of $\mathcal{C}_i$, it forbids an action $\sigma \in \Sigma_i$ if and only if there exists a state $\vec{x} \in E$ in which the action σ must be forbidden in order to prevent the system $\mathcal{T}$ from reaching $I(Bad)$ (i.e., $\exists \vec{x} \in E : \vec{x} \in \mathcal{F}_i(\sigma, I(Bad))$).

During the execution of the system, when the subsystem $\mathcal{T}_i$ ($\forall i \in [1..n]$) executes a transition $\delta = \langle \ell_i, \sigma, \ell_i' \rangle$, the local controller $\mathcal{C}_i$ receives the following information:

- if $\sigma = Q_{j,i}?m$ (with $j \neq i \in [1..n]$), it receives σ, and the triple $\langle E_j, V_j, \delta' \rangle$ tagging m.
- if $\sigma = Q_{i,j}!m$ (with $j \neq i \in [1..n]$), it receives σ.

In both cases, since $\mathcal{C}_i$ knows that $\mathcal{T}_i$ was in the location ℓ_i before triggering σ, this controller can infer the fired transition. $\mathcal{C}_i$ then uses the SE-algorithm with this information to update its state estimate E_i and computes, from this estimate, the set $\mathcal{C}_i(E_i)$ of actions that $\mathcal{T}_i$ cannot execute.

The following theorem proves that this algorithm synthesizes correct controllers for the distributed problem.

Theorem 3: Given a set of forbidden states $Bad \subseteq X$, our distributed controller $\mathcal{C}_{\mathsf{di}} = \langle \mathcal{C}_i \rangle_{i=1}^n$ solves the distributed problem if $\vec{x}_0 \notin I(Bad)$.

Proof: We prove by induction on the length m of the sequences of transitions (these sequences begin in the initial state) that $I(Bad)$ is not reachable in the system $\mathcal{T}$ under the control of $\mathcal{C}_{\mathsf{di}}$, which implies that Bad is not reachable, because $Bad \subseteq I(Bad)$:

Base case ($m = 0$): Since $\vec{x}_0 \notin I(Bad)$, the execution of the system $\mathcal{T}$ under the control of $\mathcal{C}_{\mathsf{di}}$ starts in a state which does not belong to $I(Bad)$.

Induction step: We suppose that the proposition holds for the sequences of transitions of length less than or equal to m and we prove that this property remains true for the sequences of transitions of length $m+1$. By induction hypothesis, each state $\vec{x}_1$ reachable by a sequence of transitions of length m does not belong to $I(Bad)$ and we show that each transition $\delta \in \Delta$, which can lead to a state $\vec{x}_2 \in I(Bad)$ from this state $\vec{x}_1$ in $\mathcal{T}$, cannot be fired from $\vec{x}_1$ in the system $\mathcal{T}$ under the control of $\mathcal{C}_{\mathsf{di}}$. For that, we consider two cases and we suppose that δ is executed by $\mathcal{T}_i$ and is labeled by σ:

- if δ is controllable, then σ is forbidden by $\mathcal{C}_i$ in $\vec{x}_1$ and hence δ cannot be fired from $\vec{x}_1$. Indeed, the estimate E_i of $\mathcal{C}_i$ contains $\vec{x}_1$, because the SE-algorithm is complete. Moreover, we have that $\vec{x}_1 \in \mathcal{F}_i(\sigma, I(Bad))$, because $\vec{x}_1 \in \mathsf{Pre}^{\mathcal{T}}_{\delta}(\vec{x}_2)$ and $\vec{x}_2 \in I(Bad)$. Therefore, $\sigma \in \mathcal{C}_i(E_i)$ (by (7)), which implies that δ cannot be fired from $\vec{x}_1$.
- if δ is uncontrollable, then $\vec{x}_2 \in I(Bad)$, which is impossible by hypothesis.

Hence, in the system $\mathcal{T}$ under the control of $\mathcal{C}_{\mathsf{di}}$, the forbidden state $\vec{x}_2$ cannot be reached from $\vec{x}_1$ by the transition δ. ■

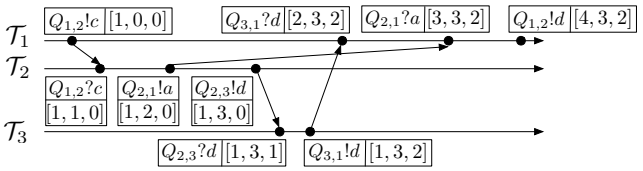

Fig. 3. An execution of the running example.

Example 1: We consider the sequence of actions of our running example of Figure 3. The set Bad is given by the set of global states where the location of $\mathcal{T}_1$ is A_{er}. Thus, $I(Bad) = Bad \cup \{\langle \ell_1, \ell_2, \ell_3, w_{1,2}, w_{2,1}, w_{2,3}, w_{3,1} \rangle | (\ell_1 = A_0) \wedge (w_{2,1} = a.M^*)\}$. At the beginning of the execution of $\mathcal{T}$, the state estimates of the subsystems are $E_1 = \{\langle A_0, B_0, D_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle\}$, $E_2 = \{\langle A_0, B_0, D_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_0, D_0, c, \epsilon, \epsilon, \epsilon \rangle\}$, and $E_3 = \{\langle A_0, B_0, D_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_0, D_0, c, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_1, D_0, \epsilon, b^*, \epsilon, \epsilon \rangle, \langle A_1, B_2, D_0, \epsilon, b^*(a + \epsilon), \epsilon, \epsilon \rangle, \langle A_1, B_3, D_0, \epsilon, b^*(a + \epsilon), d, \epsilon \rangle\}$. After the first transition $\langle A_0, Q_{1,2}!c, A_1 \rangle$, the state estimate of the controller $\mathcal{C}_1$ is not really precise, because a lot of things may happen without the controller $\mathcal{C}_1$ being informed: $E_1 = \{\langle A_1, B_0, D_0, c, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_1, D_0, \epsilon, b^*, \epsilon, \epsilon \rangle, \langle A_1, B_2, D_0, \epsilon, b^*a, \epsilon, \epsilon \rangle, \langle A_1, B_3, D_0, \epsilon, b^*(a + \epsilon), d, \epsilon \rangle, \langle A_1, B_3, D_1, \epsilon, b^*(a + \epsilon), \epsilon, \epsilon \rangle, \langle A_1, B_3, D_0, \epsilon, b^*(a + \epsilon), \epsilon, d \rangle\}$. However, after the second transition $\langle B_0, Q_{1,2}?c, B_1 \rangle$, the controller $\mathcal{C}_2$ has an accurate state estimate: $E_2 = \{\langle A_1, B_1, D_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle\}$. We skip a few steps and consider the state estimates before the sixth transition $\langle D_1, Q_{3,1}!d, D_0 \rangle$: E_1 is still the same, because the subsystem $\mathcal{T}_1$ did not perform any action, $E_3 = \{\langle A_1, B_3, D_1, \epsilon, b^*(a + \epsilon), \epsilon, \epsilon \rangle\}$, and we do not give E_2, because $\mathcal{T}_2$ is no longer involved. When $\mathcal{T}_3$ sends message d to $\mathcal{T}_1$, it tags it with E_3. Thus, $\mathcal{C}_1$ knows, after receiving this message, that there is a message a in the queue $Q_{2,1}$. It thus disables the action $A_2 \stackrel{Q_{1,2}!d}{\longrightarrow} A_0$, as long as this message a is not read (action $A_2 \stackrel{Q_{2,1}?a}{\longrightarrow} A_2$), to prevent the system from reaching the forbidden states. Note that if we consider the sequence of actions of Figure 3 without the sending and the reception of the message a, then when $\mathcal{T}_1$ reaches the location A_2 by executing the action $Q_{3,1}?d$, its controller $\mathcal{C}_1$ enables the actions $Q_{1,2}!d$, because it knows that no message a is in $Q_{2,1}$.

B. Effective Algorithm for the Distributed Problem

The algorithms described in the previous sections require the computation of (co-)reachability operators. Those operators cannot be computed exactly because of undecidability reasons. Abstract interpretation-based techniques [6] allows us to compute, in a finite number of steps, an *overapproximation* of the (co-)reachability operators, and thus of the set $I(Bad)$, and of the state estimates E_i.

Computation of (Co-)Reachability Sets by the Means of Abstract Interpretation. For a given set of global states

$X' \subseteq X$ and a given set of transitions $\Delta' \subseteq \Delta$, the reachability (resp. co-reachability) set from X' can be characterized by the least fixpoint $\mathsf{Reach}^{\mathcal{T}}_{\Delta'}(X') = \mu Y.F_{\Delta'}(Y)$ with $F_{\Delta'}(Y) = X' \cup \mathsf{Post}^{\mathcal{T}}_{\Delta'}(Y)$ (resp. $\mathsf{Coreach}^{\mathcal{T}}_{\Delta'}(X') = \mu Y.F_{\Delta'}(Y)$ with $F_{\Delta'}(Y) = X' \cup \mathsf{Pre}^{\mathcal{T}}_{\Delta'}(Y)$). Abstract interpretation provides a theoretical framework to compute efficient overapproximation of such fixpoints. The concrete domain i.e., the sets of states 2^X, is substituted by a simpler abstract domain Λ, linked by a *Galois connection* $2^X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \Lambda$ [6], where α (resp. γ) is the abstraction (resp. concretization) function. The fixpoint equation is transposed into the abstract domain. So, the equation to solve has the form: $\lambda = F^{\sharp}_{\Delta'}(\lambda)$, with $\lambda \in \Lambda$ and $F^{\sharp}_{\Delta'} \sqsupseteq \alpha \circ F_{\Delta'} \circ \gamma$ where $\sqsupseteq$ is the comparison operator in the abstract lattice. In that setting, a standard way to ensures that this fixpoint computation converges after a finite number of steps to some overapproximation λ_∞, is to use a *widening operator* ∇. The concretization $c_\infty = \gamma(\lambda_\infty)$ is an overapproximation of the least fixpoint of the function $F_{\Delta'}$.

Choice of the Abstract Domain. In abstract interpretation-based techniques, the quality of the approximation we obtain depends on the choice of the abstract domain Λ. In our case, the main issue is to abstract the content of the FIFO channels. Since the CFSM model is Turing-powerful, the language which represents all the possible contents of the FIFO channels may be recursively enumerable. As discussed in [22], a good candidate to abstract the contents of the queues is to use the class of regular languages, which can be represented by finite automata. Let us recall the main ideas of this abstraction.

Finite Automata as an Abstract Domain. We first assume that there is only one queue in the distributed system $\mathcal{T}$; we explain later how to handle a distributed system with several queues. With one queue, the concrete domain of the system $\mathcal{T}$ is defined by $X = 2^{L \times M^*}$. A set of states $Y \in 2^{L \times M^*}$ can be viewed as a map $Y : L \mapsto 2^{M^*}$ that associates a language $Y(\ell)$ with each location $\ell \in L$; $Y(\ell)$ therefore represents the possible contents of the queue in the location ℓ. In order to simplify the computation, we substitute the concrete domain $\langle L \mapsto 2^{M^*}, \subseteq \rangle$ by the abstract domain $\langle L \mapsto \mathrm{Reg}(M), \sqsubseteq \rangle$, where $\mathrm{Reg}(M)$ is the set of *regular languages* over the alphabet M and $\sqsubseteq$ denotes the natural extension of the set inclusion to maps. This substitution consists thus in abstracting, for each location, the possible contents of the queue by a regular language. Regular languages have a canonical representation given by finite automata, and each operation (union, intersection, left concatenation,...) in the abstract domain can be performed on finite automata.

Widening Operator. With our abstraction, the widening operator we use to ensure the convergence of the computation, is also performed on a finite automaton, and consists in quotienting the nodes[7] of the automaton by the *k-bounded bisimulation relation* $\equiv_k$; $k \in \mathbb{N}$ is a parameter which allows us to tune the precision: increasing k improves the quality of the abstractions in general. Two nodes are equivalent w.r.t. $\equiv_k$ if they have the same outgoing path (sequence of labeled transitions) up to length k. While we merge the equivalent nodes, we keep all transitions and obtain an automaton recognizing a larger language. Note that the number of equivalent classes of the k-bounded bisimulation relation is bounded by a function of k and of the size of the alphabet of messages. Therefore the number of states of the resulting automaton is also bounded. So, if we fix k and we apply this widening operator regularly, the fixpoint computation terminates (see [22] for more details and examples).

[7]The states of an automaton representing the queue contents are called nodes to avoid the confusion with the states of a CFSM.

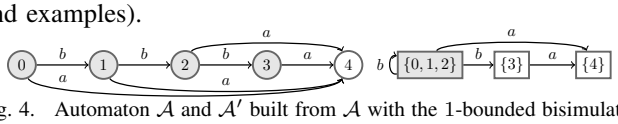

Fig. 4. Automaton $\mathcal{A}$ and $\mathcal{A}'$ built from $\mathcal{A}$ with the 1-bounded bisimulation relation $\equiv_1$

Example 2: We consider the automaton $\mathcal{A}$ depicted in Figure 4, whose recognized language is $a + ba + bba + bbba$. We consider the 1-bounded bisimulation relation i.e., two nodes of the automaton are equivalent if they have the same outgoing transitions. So, nodes $0, 1, 2$ are equivalent, since they all have two transitions labeled by a and b. Nodes 3 and 4 are equivalent to no other node since 4 has no outgoing transition whereas only a is enabled in node 3. When we quotient $\mathcal{A}$ by this equivalent relation, we obtain the automaton $\mathcal{A}'$ on the right of Figure 4, whose recognized language is b^*a. $\diamond$

When the system contains several queues $Q = \{Q_1, \ldots, Q_r\}$, their content can be represented by a concatenated word $w_1 \sharp \ldots \sharp w_r$ with one w_i for each queue Q_i and $\sharp$, a delimiter. With this encoding, we represent a set of queue contents by a finite automaton of a special kind, namely a QDD [4]. Since QDDs are finite automata, classical operations (union, intersection, left concatenation,...) in the abstract domain are performed as previously. We must only use a slightly different widening operator not to merge the different queue contents [22].

Effective Algorithm. The Reach and Coreach operators are computed using those abstract interpretation techniques: we proceed to an iterative computation in the abstract domain of regular languages and the widening operator ensures that this computation terminates after a finite number of steps [6]. So the Reach (resp. Coreach) operators always give an overapproximation of the reachable (resp. co-reachable) states, whatever the distributed system is. Finally, we define the distributed controller as in section VI-A by using the overapproximations $I'(Bad)$ and E'_i instead of $I(Bad)$ and E_i.

VII. Experiments

Our control algorithm has been implemented as a part of the McScM tool, and freely available at [29]. McScM's input is a CFSM model of the system. The set Bad is given by a set of locations and regular expressions describing what the queues should not contain. Our tool first computes an overapproximation of $I(Bad)$ according to the algorithms of sections VI. Then it starts an interactive simulation of the system. At each step, it displays the current state of the system and the transitions forbidden by the controller, and asks the user to choose a transition among the allowed ones. Then, it updates the current state of the system and the state estimates as in section VI and thus enables or disables the controllable transitions.

example	# subsystems	# channels	time [s]	memory [MB]	maximal size	average size
running example	3	4	7.13	5.09	143	73.0
c/d protocol	2	2	5.32	8.00	183	83.2
non-regular protocol	2	1	0.99	2.19	172	47.4
ABP	2	3	1.19	2.19	49	24.8
sliding window	2	2	3.26	4.12	21	10.1
POP3	2	2	3.08	4.12	22	8.5

TABLE I
TIME AND MEMORY CONSUMPTION OF A 100-STEPS RANDOM RUN

Experiment on the Running Example. On this example, our software computes the exact set $I(Bad)$ (see Example 1) if we set the widening parameter $k = 1$. We considered the sequences of events of Example 1 and the software validates the theory. The computation of $I(Bad)$ and execution of each sequence of events took less than 0.4s of run time and required 1.22 MB of memory on a standard laptop.

Experiment on the Connection/Disconnection Protocol. In this example taken from [22], an error occurs when the client and the server send close/disconnect message at the same time. Our controller solves the problem by not allowing the server to send disconnection messages. The computation of $I(Bad)$ took less than 0.1s and required 1.22 MB of memory.

Simulation. Instead of asking the user what transitions should be taken, our software can randomly choose them. Table I displays the time and memory consumption needed by a 100-steps random run on several examples of communication protocol. It also mentions the size (number of nodes) of the state estimate computed during this run.

Remark 2: Note that even though the state space is unbounded, state estimates are symbolical representations of sets of states, and their sizes do not depend on the number of states they represent. For example, a state estimate which represents a queue containing one or more messages 'a' (i.e. the infinite set of states a,aa,aaa,...) can be encoded by an automaton with only two nodes and two transitions. Thus, the state estimates always have a finite representation, and the experiments give the maximal and average size of this representation.

VIII. CONCLUSION AND FURTHER WORKS

We propose in this paper a novel framework for the control of distributed systems modeled as communicating finite state machines with reliable unbounded FIFO channels. Each local controller can only observe its subsystem but can communicate with the other controllers by piggy-backing extra information, such as state estimates, to the messages sent in the FIFO channels. Our algorithm synthesizes the local controllers that restrict the behavior of a distributed system in order to satisfy a global state avoidance property, e.g. to ensure that an error state is no longer reachable or to bound the size of the FIFO channels. We abstract the content of the FIFO channels by the same regular representation as in [22]; this abstraction leads to a safe effective algorithm. Even if we cannot have any theoretical guarantee about the permissiveness of the control (like a non-blocking property), we remind that this permissiveness depends on the quality of the abstraction. The more precise the abstraction is, the more permissive the control is. Our experiments show that our approach is tractable and allows a precise control.

As a further work, we intend to solve the main practical problem of our approach: we compute and send states estimates every time a message is sent. A more evolved technique would consist in the offline computation of the set of possible estimates. Estimates would be indexed in a table, available at execution time to each local estimator. A similar online method would be to use the memoization technique: when a state estimate is computed for the first time, it is associated with an index that is transmitted to the subsystem which records both values. If the same estimate must be transmitted, only its index can be transmitted and the receiver can find from its table the corresponding estimate. We still have to determine what is the most efficient technique, and evaluate how it improves the current implementation. We also believe that the work of decentralized control with communication and modular control with coordinator might be adapted in our framework in order to reduce the communication between controllers.

REFERENCES

[1] G. Barrett and S. Lafortune. Decentralized supervisory control with communicating controllers. *IEEE Transactions on Automatic Control*, 45(9):1620–1638, 2000.

[2] S. Bensalem, M. Bozga, S. Graf, D. Peled, and S. Quinton. Methods for knowledge based controlling of distributed systems. In *ATVA'10*, volume 6252 of *LNCS*, pages 52–66. Springer, 2010.

[3] G. Berry and G. Gonthier. The esterel synchronous programming language: Design, semantics, implementation. *Sci. Comput. Program.*, 19(2):87–152, 1992.

[4] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of qdds. In *SAS '97: Proceedings of the 4th International Symposium on Static Analysis*, pages 172–186, London, UK, 1997. Springer-Verlag.

[5] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[6] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL'77*, pages 238–252, 1977.

[7] P. Darondeau. Distributed implementations of Ramadge-Wonham supervisory control with petri nets. In *44th IEEE Conference on Decision and Control*, pages 2107–2112, Sevilla, Spain, December 2005.

[8] C. J. Fidge. Timestamps in message-passing systems that preserve the partial ordering. In *Proc. of the 11th Australian Computer Science Conference (ACSC'88)*, pages 56–66, February 1988.

[9] P. Gastin, N. Sznajder, and M. Zeitoun. Distributed synthesis for well-connected architectures. *Formal Methods in System Design*, 34(3):215–237, 2009.

[10] B. Gaudin and H. Marchand. An efficient modular method for the control of concurrent discrete event systems: A language-based approach. *Discrete Event Dynamic System*, 17(2):179–209, 2007.

[11] B. Genest. On implementation of global concurrent systems with local asynchronous controllers. In *CONCUR*, volume 3653 of *LNCS*, pages 443–457, 2005.

[12] A. Genon, Th. Massart, and C. Meuter. Monitoring distributed controllers: When an efficient ltl algorithm on sequences is needed to model-check traces. In *FM*, volume 4085 of *Lecture Notes in Computer Science*, pages 557–572. Springer, 2006.
[13] K. Iraishi. On solvability of a decentralized supervisory control problem with communication. *IEEE Transactions on Automatic Control*, 54(3):468–480, March 2009.
[14] C. Jard, Jéron T, Jourdan G.-V, and J.-X. Rampon. A general approach to trace-checking in distributed computing systems. In *ICDCS*, pages 396–403, 1994.
[15] S. Jiang and R. Kumar. Decentralized control of discrete event systems with specializations to local control and concurrent systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 30(5):653–660, October 2000.
[16] G. Kalyon, T. Le Gall, H. Marchand, and Th. Massart. Symbolic supervisory control of distributed systems with communications. Research Report 8260, INRIA, 2013. http://hal.inria.fr/hal-00801840.
[17] G. Kalyon, T. Le Gall, H. Marchand, and Th. Massart. Global state estimates for distributed systems. In *31th IFIP International Conference on FORmal TEchniques for Networked and Distributed Systems, FORTE*, volume 6722 of *LNCS*, pages 198–212, June 2011.
[18] G. Kalyon, T. Le Gall, H. Marchand, and Th. Massart. Synthesis of communicating controllers for distributed systems. In *50th IEEE Conference on Decision and Control and European Control Conference*, Orlando, USA, December 2011.
[19] G. Kalyon, Th. Massart, C. Meuter, and L. Van Begin. Testing distributed systems through symbolic model checking. In *FORTE*, volume 4574 of *LNCS*, pages 263–279, 2007.
[20] J. Komenda and J.H. van Schuppen. Supremal sublanguages of general specification languages arising in modular control of discrete-event systems. In *44th IEEE Conference on Decision and Control*, pages 2775–2780, 2005.
[21] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.
[22] T. Le Gall, B. Jeannet, and T. Jéron. Verification of communication protocols using abstract interpretation of fifo queues. In *11th International Conference on Algebraic Methodology and Software Technology, AMAST '06*, LNCS, July 2006.
[23] S.-H. Lee and Wong K.C. Structural decentralized control of concurrent discrete-event systems. *European Journal of Control*, 8(5), 2002.
[24] F. Lin, K. Rudie, and S. Lafortune. Minimal communication for essential transitions in a distributed discrete-event system. *IEEE Transactions on Automatic Control*, 52(8):1495–1502, June 2007.
[25] J. C. Martin. *Introduction to Languages and the Theory of Computation*. McGraw-Hill Higher Education, 1997.
[26] Th. Massart. A calculus to define correct tranformations of lotos specifications. In *FORTE*, volume C-2 of *IFIP Transactions*, pages 281–296, 1991.
[27] F. Mattern. Virtual time and global states of distributed systems. In *Proceedings of the Workshop on Parallel and Distributed Algorithms*, pages 215–226, North-Holland / Elsevier, 1989.
[28] Antoni W. Mazurkiewicz. Trace theory. In *Advances in Petri Nets*, pages 279–324, 1986.
[29] McScM, a Model Checker for Symbolic Communicating Machines - version 1.2. http://altarica.labri.fr/forge/projects/mcscm/wiki/.
[30] A. Muscholl and I. Walukiewicz. A lower bound on web services composition. In *FOSSACS'07*, pages 274–286, Berlin, Heidelberg, 2007. Springer-Verlag.
[31] Wuxu Peng and S. Puroshothaman. Data flow analysis of communicating finite state machines. *ACM Trans. Program. Lang. Syst.*, 13(3):399–442, July 1991.
[32] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *FOCS*, pages 746–757. IEEE Computer Society, 1990.
[33] P.J. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems*, 77(1):81–98, 1989.
[34] K. Ricker, L. Rudie. Know means no: Incorporating knowledge into discrete-event control systems. *IEEE Transactions on Automatic Control*, 45(9):1656–1668, September 2000.
[35] K. Rudie and W.M. Wonham. Think globally, act locally: decentralized supervisory control. *IEEE Transaction on Automatic Control*, 31(11):1692–1708, November 1992.
[36] A. Sen and V. K. Garg. Detecting temporal logic predicates on the happened-before model. In *IPDPS*, 2002.
[37] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
[38] S. Tripakis. Decentralized control of discrete event systems with bounded or unbounded delay communication. *IEEE Trans. on Automatic Control*, 49(9):1489–1501, 2004.
[39] S. Xu and R. Kumar. Distributed state estimation in discrete event systems. In *ACC'09: Proceedings of the 2009 conference on American Control Conference*, pages 4735–4740. IEEE Press, 2009.
[40] T. Yoo and S. Lafortune. A general architecture for decentralized supervisory control of discrete-event systems. In *Proc of 5th Workshop on Discrete Event Systems, WODES 2000*, Ghent, Belgium, August 2000.

Gabriel Kalyon received a Master Degree in Computer Science at Université Libre de Bruxelles, Belgium in 2006 . In 2010 received a a Ph.D. degree for a thesis he did inside a collaboration between the Formal Methods and Verification Group of the Computer Science Department at the Université Libre de Bruxelles, Belgium and the VerTeCs Group at INRIA Rennes, France. His research interests include Supervisory Control, Testing and Data Structures for efficient Model-Cheching.

Tristan Le Gall received a Ph.D. degree in computer science from Université de Rennes 1, France, in 2008. In 2008-2010, he held a post-doc position at Université Libre de Bruxelles, Belgium. Since 2010, he is a junior researcher at CEA LIST, France. His research interests include Supervisory Control, Static Analysis, Model-Checking of Infinite-State Systems and Game Theory.

Thierry Massart is professor in the Formal Methods and Verification Group in the Computer Sciences Department of the Université Libre de Bruxelles, Belgium since 1991. He received a PhD from the same University in 1990. His research is mainly related to formal specification and verification of distributed systems. In particular, the reliable design of distributed systems, the design of efficient model-checking and testing techniques for distributed systems, the controller synthesis for infinite state systems.

Hervé Marchand received a Ph.D. degree in computer science from the Université de Rennes 1, France in 1997. In 1998, he spent one year at the University of Michigan. Since 1998, he holds an INRIA research scientist position (Rennes, France). His research interests include Supervisory Control, automatic test generation and diagnosis of Discrete Events Systems and their applications to security. He is also interested in high-level languages for reactive and real-time systems programming. Between January 2010 and December 2012, he was Associate Editor of the IEEE Transactions on Automatic Control journal.