Coordination of Components in a Distributed Discrete-Event System

Ahmed Khoumsi
Department of Electrical and Computer Engineering
Université de Sherbrooke, Canada J1K–2R1
Ahmed.Khoumsi@USherbrooke.ca

Abstract

We propose a method for coordinating local components that observe a distributed discrete-event system \( \mathcal{R} \) and execute actions depending on the current state of \( \mathcal{R} \). Coordination is achieved by allowing the components to communicate with one another. This communication helps the components to distinguish states of \( \mathcal{R} \) when necessary. This study is illustrated in distributed supervisory control.

1. Introduction

Discrete event systems (DES) have their behaviors defined by the possible sequences of events they can execute. This paper deals with distributed DES, that execute events in several sites distant with each other. We consider the case of a distributed DES \( \mathcal{R} \) that has a distinct component for each of its sites. A component observes uniquely the events in its site, and may have to execute actions depending on the current (global) state of \( \mathcal{R} \). The problem is that a component may be unable to distinguish some (global) states of \( \mathcal{R} \) just from its (local) observations. We solve this problem by permitting the components to exchange (coordination) messages with each other (see Fig. 1). This study can be helpful in areas requiring surveillance of systems, such as supervisory control [15], conformance testing [8, 22], and failure diagnosis [13, 20].

2. Coordination problem to be solved

2.1. Necessity of coordination, Actions

We consider a distributed DES \( \mathcal{R} \) that has a distinct component for each of its sites. A component observes uniquely the (local) events in its site, and thus, may be unable to determine the current (global) state of \( \mathcal{R} \). Generically, a component knows that \( \mathcal{R} \) is in one of the states that belong to a set of states \( K \), and the latter depends on what has been observed by the component. Ideally, \( K \) consists of a single state, and in the worst case, \( K \) contains all the states of \( \mathcal{R} \).

We consider the case where during the evolution of \( \mathcal{R} \) (by executions of events), each component may have to execute actions that depend on the current state of \( \mathcal{R} \). Actions are instantaneous, that is, their duration is negligible and can be approximated by 0. This is not restrictive, because an action \( \theta \) with a nonnegligible duration can be represented by two instantaneous actions: the beginning and the termination of \( \theta \). Here are some examples of (instantaneous) actions: to activate or stop an alarm; to detect or loose an obstacle; to enable or disable an event; to establish a connection, or to terminate or modify an established connection; to start regulating the speed of a dynamic systems, or to stop or modify an ongoing regulation. It is worth noting that an action (executed by a component) is different from an event (executed by \( \mathcal{R} \) and observed by a component). But an action can be related to one or several events: for example, an action may consist of preventing an event.

Figure 1. Communicating components in a distributed DES.
A problem arises when a component cannot distinguish two states \(q_1\) and \(q_2\) of \(\mathcal{R}\) whereas the actions associated to the two states are different. More generally, this type of problem occurs when a component cannot distinguish \(n\) \((n \geq 2)\) states associated to different actions. An approach to solve this problem is that the components coordinate themselves by exchanging (coordination) messages (see Fig. 1). Before to define formally the problem to be solved, we need to present Automata with Actions (AwA) used to model \(\mathcal{R}\) and the components.

### 2.2. Automata with Actions (AwA)

Let \(E\) be a finite set of events and \(A\) be a finite sets of actions. An Automaton with Actions (AwA) over \(E\) and \(A\) is defined by \((S, E, T, A, F, S^0)\), where \(S\) is a finite set of states and \(S^0 \in S\) is the initial state. \(T \subseteq S \times E \times S\) is a transition relation; a transition is thus defined by \(Tr = [q; e; r]\) where: \(q\) and \(r\) are origin and destination states, and \(e\) is an event. \(F\) is a feedback function: \(Q \longrightarrow 2^A\) (denotes the set of subsets of \(A\)). Intuitively, \(F(q)\) indicates the actions to be executed each time the state \(q\) is reached.

**Remark 2.1** A AwA with an empty \(A\) is a classical finite state automaton (FSA).

**Hypothesis 2.1** The considered AwA are deterministic, that is, we cannot have two transitions with the same origin state, the same event, and different destination states.

**Definition 2.1** Let \(E\) and \(A\) be finite sets of events and actions, resp. A trace over \(E\) and \(A\) is a finite sequence \(\lambda = a_1^{\cdot} e_1^{\cdot} A_1^{\cdot} e_2^{\cdot} \cdot \cdot \cdot A_n^{\cdot} e_n^{\cdot} A_1^{n+1}\), where \(e_i \in E\) and \(A_i \subseteq A\) \((A'_i\) can be empty). Let an H-trace denote a trace whose all events belongs to \(H \subseteq E\).

**Definition 2.2** The language of a AwA \(\mathcal{K}\) (denoted \(L_\mathcal{K}\)) is the set of traces that can be executed by \(K\) from its initial state. Note that \(L_\mathcal{K}\) is prefix closed.

**Definition 2.3** Two AwA \(U\) and \(V\) are said equivalent (written \(U \equiv V\)) iff \(L_U = L_V\).

**Definition 2.4** Let \(\lambda = A_1^{\cdot} e_1^{\cdot} A_2^{\cdot} \cdot \cdot \cdot A_n^{\cdot} e_n^{\cdot} A_1^{n+1}\) be a trace defined over \(E\) and \(A\), and let \(H \subseteq E\). \(H\) is hidable in \(\lambda\) iff for every \(H\)-trace \(A_1^{j+1} \cdot \cdot \cdot A_p^{j+1}\) contained in \(\lambda\) \((1 \leq j \leq p \leq n)\), we have: \(A_1^{j+1}, A_2^{j+2}, \cdot \cdot \cdot A_p^{j+1}\) empty, and \(A_i\) or \(A_p^{j+1}\) empty. In such a case, we hide \(H\) in \(\lambda\) by replacing every \(H\)-trace \(A_1 \cdot \cdot \cdot A_p^{j+1}\) by its \(A_i\) or \(A_p^{j+1}\), necessarily the nonempty one if any. The result is denoted \(\lambda^H\). \(H\) is hidable in a language \(L\) iff \(H\) is hidable in every trace \(\lambda \in L\). In such a case, we define: \(L^H = \{\mu : \mu = \lambda^H\text{ for some }\lambda \in L\}\).

**Lemma 2.1** Let \(K\) be an AwA defined over \(E\) and \(A\), and \(L\) be its language. If \(H \subset E\) is hidable in \(L\), then there exists an AwA (denoted \(K^H\)) whose language is \(L^H\). We will say that \(H\) is hidable in \(K\).

Intuitively, \(\lambda^H\) (resp. \(L^H\), \(K^H\)) models the same behavior as \(\lambda\) (resp. \(L\), \(K\)), but as it is observed through a mask hiding the events of \(H\). The fact that \(H\) is hidable in a trace (resp. language) \(X\) of AwA, guarantees that \(X^H\) is a trace (resp. language) of AwA.

### 2.3. Model of the distributed DES \(\mathcal{R}\)

For simplicity, we consider \(\mathcal{R}\) distributed in two sites identified by 1 and 2, and the corresponding components are denoted \(\mathcal{R}_1\) and \(\mathcal{R}_2\) respectively. \(\mathcal{R}\) is modeled by a AwA \(R = (S, \Sigma_1 \cup \Sigma_2, T, \Gamma_1 \cup \Gamma_2, F, S^0)\). \(\Sigma_1 \cup \Sigma_2\) is the alphabet of \(R\), where \(\Sigma_1\) and \(\Sigma_2\) are finite sets of events occurring in sites 1 and 2 respectively, and \(\Sigma_1 \cap \Sigma_2 = \emptyset\). \(\Gamma_1\) and \(\Gamma_2\) are finite sets of actions executed by \(\mathcal{R}_1\) and \(\mathcal{R}_2\) (in sites 1 and 2) respectively, and \(\Gamma_1 \cap \Gamma_2 = \emptyset\). Henceforth, an event or action \(e\) of \(\Sigma_1\) or \(\Gamma_1\) is written \(e_j\).

Fig. 2 represents a AwA \(R\) where: \(\Sigma_1 = \{a_1, b_1, c_1\}, \Sigma_2 = \{c_2, d_2\}, \Gamma_1 = \{a_1, b_1, \gamma_1, \delta_1\}, \Gamma_2 = \{\mu_2, \nu_2\}\). States are represented by nodes and identified by numbers, and the initial state is indicated by an incoming arrow. Actions associated to a state are indicated inside its corresponding node. A transition \([q; e\; \sigma; r]\) is represented by an arrow linking \(q\) to \(r\) and labeled by \(\sigma\). Intuitively, \(\mathcal{R}\) evolves by executing events as specified in \(R\), and:

- \(\mathcal{R}_1\) observes the events \(a_1, b_1\) and \(c_1\) and:
  - executes the actions \(a_1, b_1, \gamma_1, \delta_1\) in State 1;
  - executes the actions \(\gamma_1, \delta_1\) when \(\mathcal{R}\) reaches State 2;
  - executes the actions \(\gamma_1, \delta_1\) when \(\mathcal{R}\) reaches State 3;
  - executes the action \(\delta_1\) when \(\mathcal{R}\) reaches State 5.
- \(\mathcal{R}_2\) observes the events \(c_2\) and \(d_2\), and execute all the actions of \(\Gamma_2\) in the initial State 1.

![Figure 2. Example of AwA \(R\) modeling \(\mathcal{R}\)](image)

### 2.4. Model of the components, and some operators

#### 2.4.1 Models of the components \(\mathcal{R}_1\) and \(\mathcal{R}_2\)

As we have explained in Sects 1 and 2.1, the components will have to exchange messages. Given the set of events...
Σ_1 ∪ Σ_2 and the set of actions Γ_1 ∪ Γ_2 of R, each component R_i (i = 1, 2) is modeled by an AwA R_i = (Q_i, Σ_i ∪ Λ_i, ξ_i, Γ_i, ψ_i, q_0^i). Q_i is a finite set of states and q_0^i ∈ Q_i is the initial state. Σ_i ∪ Λ_i is the alphabet of R_i, where Σ_i is the set of events executed by R and observed by R_i (see Sect. 2.3) and Λ_i is the set of sendings and receptions by R_i to and from the other component. Let s_i^k(k) denote the sending by R_i (for R_j) of a message containing information k, and r_i^j(k) the reception by R_j of a message (from R_i) containing information k. Γ_i is a finite set of actions executed by R_i (see Sect. 2.3). ξ_i is a transition relation and ψ_i is a feedback function. As we will see in Sect. 3, each R_i satisfies the following hypotheses:

**Hypothesis 2.2** For every transition Tr = [q; s_i^k(k); r_i] of R_i: no action is associated to q, q has not another outgoing transition than Tr, and every incoming transition of q is labeled by an event of Σ_i.

**Hypothesis 2.3** For every transition Tr = [q; r_i^j(k); r_j] of R_j, q is not the destination state of any transition labeled by an event of Σ_i.

Fig. 3 represents two AwAs R_1 and R_2 where: Σ_1 = {a_1, b_1, c_1}. Γ_1 = {α_1, β_1, γ_1, δ_1, π_1, δ_1}. Σ_2 = {e_2, d_2}. Γ_2 = {μ_2, ν_2}. Λ_1 = {s_i^2(3), r_i^2(5)}. Λ_2 = {s_i^2(5), r_i^2(3)}. The aim is to synchronize R_1 ⊗ R_2 with R.

![Figure 3. Example of AwAs R_1 and R_2](image)

**2.4.2 Synchronized product of R_1 and R_2**

Given R_i = (Q_i, Σ_i ∪ Λ_i, ξ_i, Γ_i, ψ_i, q_0^i), (i = 1, 2), we now define the synchronized product of R_1 and R_2 as a AwA denoted R_1 ⊗ R_2 that models the joint behavior of R_1 and R_2. Intuitively:

* Each R_i observes the events of Σ_i.

* After each observation, R_i executes, if any, the actions of Γ_i associated to the state reached.

* Every s_i^k(k) of R_i is executed simultaneously to r_j^j(k) of R_j, for i, j = 1, 2 and i ≠ j. This simultaneous execution is denoted trans_{i,j}(k) and indicates a transmission of k from R_i to R_j.

* When a trans_{i,j}(k) is possible, it is executed before any other event. This is conformant with Hyp. 3.1 that will be used in Sect. 3, where communication is assumed immediate and instantaneous. In Sect. 4.2, we discuss the case where Hyp. 3.1 is not satisfied.

Formally, R_1 ⊗ R_2 = (Q, Σ_1 ⊃ Σ_2 ⊃ Λ, ξ, Γ_1 ⊃ Γ_2, ψ, q_0^i), such that: Q = Q_1 × Q_2, q_0^i = (q_0^1, q_0^2). Λ is a set of elements trans_{i,j}(k) such that s_i^k(k) ∈ Λ_i and r_j^j(k) ∈ Λ_j for i, j = 1, 2 and i ≠ j. The set of transitions ξ and the feedback function ψ are constructed as follows:

**Step 1:** ψ(q; q_i^k_1, q_j^k_2) = ψ(q_i^k_1) ∪ ψ(q_j^k_2) and

* ∀q_1, r_1 ∈ Q_1, ∀q_2 ∈ Q_2, ∀r_i ∈ Σ_i s.t. [q_1; x; r_1] ∈ ξ_i: [(q_1; x; q_2); (r_1; q_2)] ∈ ξ and ψ((r_1; q_2)) = ψ(r_1).

* ∀q_2, r_2 ∈ Q_2, ∀q_1 ∈ Q_1, ∀r_j ∈ Σ_j s.t. [q_2; y; r_2] ∈ ξ_j: [(q_2; y; q_1); (r_2; q_1)] ∈ ξ and ψ((r_2; q_1)) = ψ(r_2).

* ∀q_1, r_1 ∈ Q_1, ∀q_2, r_2 ∈ Q_2, ∀s_i^k_1(k) ∈ Λ_i, ∀s_j^k_2(k) ∈ Λ_j s.t. [(q_1; s_i^k_1(k); r_1) ∈ ξ_i] and [(q_2; s_j^k_2(k); r_2) ∈ ξ_j]: [(q_1; q_2); trans_{i,j}(k); (r_1; r_2)] ∈ ξ and ψ((r_1; r_2)) = ψ(r_1) ∪ ψ(r_2).

**Step 2:** For every transition ([q; trans_{i,j}(k); r] ∈ ξ), we remove every other transition whose origin state is q. This step implements the fact that when a communication has to be done, it is executed before any other possible event. This is conformant with Hyp. 3.1 of Sect. 3. Then, ξ and Q are updated by removing unreachable states.

**2.4.3 Synchronization of R_1 ⊗ R_2 with R**

The aim is to synchronize R_1 ⊗ R_2 = (Q, Σ_1 ⊃ Σ_2 ⊃ Λ, ξ, Γ_1 ⊃ Γ_2, ψ, q_0^i) with R = (S, Σ_1 ⊃ Σ_2 ⊃ T, Γ_1 ⊃ Γ_2, F, s^0). The result is denoted (R_1 ⊗ R_2)R. Intuitively, each event of Σ_i is observed by R_i only if it is accepted by R, for i = 1, 2; actions of R are ignored. More formally, (R_1 ⊗ R_2)R = (Q_s, Σ_1 ⊃ Σ_2 ⊃ Λ, ξ_s, Γ_1 ⊃ Γ_2, ψ_s, q_s^0) such that: Q_s = Q × S, q_s^0 = (q_0^1, s^0), and ψ_s is a feedback function defined by ψ_s((q; r); x) = ψ((q; r)). The set of transitions ξ_s is defined as follows:

* ∀q, q’ ∈ Q, ∀r ∈ S, ∀x ∈ Λ, ((q; r); x; (q’; r)) ∈ ξ_s if ((q; x; q’; r)) ∈ ξ.
Lemma 2.2 Hyps. 2.2 and 2.3 imply that for every transition \( Tr = [q; \text{trans}_{1,j}(k); r] \) of \( R_1 \otimes R_2 \) and \( (R_1 \otimes R_2)_R \) no action is associated to \( q \); \( q \) has not another outgoing transition than \( Tr \); and every incoming transition of \( q \) is labeled by an event of \( \Sigma_i \).

Fig. 4 represents \( (R_1 \otimes R_2)_R \) for the \( R_1 \) and \( R_2 \) of Fig. 3.

![Figure 4. \( (R_1 \otimes R_2)_R \) for the \( R_1 \) and \( R_2 \) of Fig. 3](image)

2.4.4 Hiding communications

Lemma 2.3 The set of events \( \text{trans}_{1,j}(k) \) is hidden in (the language of) \( (R_1 \otimes R_2)_R \).

From Lemmas 2.1 and 2.3, we deduce that if \( L \) is the language of the AwA \( (R_1 \otimes R_2)_R \) and \( C \) is its set of communications, then there exists a AwA denoted \( (R_1 \otimes R_2)_R^C \) whose language is \( L^C \).

Lemma 2.4 \( (R_1 \otimes R_2)_R^C \) is constructed by applying the following four operations to every transition \( Tr = [q; \text{trans}_{1,j}(k); r] \) of \( (R_1 \otimes R_2)_R \):

- \( Tr \) is removed and its origin and destination states \((q \text{ and } r)\) are merged into a single state \( q\rightarrow r \);
- \( q\rightarrow r \) becomes the destination state of the transitions whose destination state was \( q \) or \( r \);
- \( q\rightarrow r \) becomes the origin state of the transitions whose origin state was \( r \);
- we associate to \( q\rightarrow r \) the actions (if any) that were associated to \( q \) or \( r \).

If we hide communications in the \( (R_1 \otimes R_2)_R \) of Fig. 4, we obtain the AwA \( (R_1 \otimes R_2)_R^C \) of Fig. 2.

2.5. Formalization of the problem to be solved

We have now all the ingredients to formalize our problem: Given an automaton \( R \), the objective is to construct two AwA \( R_1 \) and \( R_2 \) such that: \( (R_1 \otimes R_2)_R^C \equiv R \).

Intuitively, we have to design two components \( \mathcal{R}_1 \) and \( \mathcal{R}_2 \) that observe a distributed system and execute actions as specified in \( R \). Note that this objective is satisfied by the \( R, R_1, R_2 \) of Figs. 2 and 3.

3. Coordination with negligible reaction and communication delays

3.1. Principle and algorithm of coordination

Given an AwA \( R \), let us propose a procedure for constructing two AwAs \( R_1 \) and \( R_2 \) that satisfies our objective. The simplest way to achieve coordination is that each component \( \mathcal{R}_i \) inform the other \( \mathcal{R}_j \) of all its observations (i.e., of all events executed in its site). When communication is costly, the most desirable way is that the components exchange only necessary information. In this section, we propose a simple coordination method using the following principle and hypothesis:

If in \( R \), an event \( x_i \) observed by \( \mathcal{R}_i \) leads to a state \( r \) : associated to an action executed by \( \mathcal{R}_j \), or from which an event can be observed by \( \mathcal{R}_j \).

Then after the observation of \( x_i \), \( \mathcal{R}_i \) sends a message which must be received by \( \mathcal{R}_j \) before any of its execution or observation that follows \( x_i \). The message indicates the current global state \( r \).

Hypothesis 3.1 The following two delays are negligible:
1) reaction delays of components to send messages, and
2) communication delays.

Here is an algorithm that synthesizes \( R_1 \) and \( R_2 \) from \( R \) based on the above coordination principle.

**INPUT:** \( R = (S, \Sigma_1 \cup \Sigma_2, T, \Gamma_1 \cup \Gamma_2, F, s^0) \).
**RESULT:** \( R_i = (Q_i, \Sigma_i \cup \Lambda_i, \xi_i, \Gamma_i, \psi_i, q_i^0), \) for \( i = 1, 2 \).
**BEGIN**

1. For \( i := 1 \) to 2:
2. \( R_i := R; \)
3. For every state \( s \in Q_i \): \( \psi_i(s) := \psi_i(s) \cap \Gamma_i \)
4. **EndFor**

5. For every transition \( Tr \) of \( R \):
6. Let \( Tr = [q; x; r] \)
7. Let \( i, j = 1, 2 \) s.t. \( x \) is observed by \( \mathcal{R}_i \), and \( i \neq j \)
8. If \( (\exists [r; u; s] \in T \) s.t. \( u \in \Sigma_j \) And \((q \neq r)\) Or \((\psi_j(r) \neq 0)\)) Then:
   9. In \( R_i \): replace \([q; x; r] \) by \([q; x; q\rightarrow r[s_i^j(r); r]] \)
   10. In \( R_j \): replace \([q; x; r] \) by \([q; r_j^i(r); r] \)
11. Else: In \( R_j \): replace \([q; x; r] \) by \([q; \epsilon; r] \)
12. **EndIf**
13. **EndFor**
14. For \( i := 1 \) to 2:
15. Hide events \( \epsilon \) of \( R_i \)
16. Determine \( R_i \)
17. Minimize \( R_i \)
18. **EndFor**
**END**
3.2. Explanations and properties of the synthesis algorithm

In the sequel, $L_{i\rightarrow j}$ denotes the part of the algorithm consisting of Lines $i, i+1, \ldots, j$; and $L_i$ denotes $L_{i-1}$.

$L_{1-4}$ : Each $R_i$ is initialized by $R_i$ without actions of $R_j$.

$L_{5-10}$ : The aim is to construct $R_1$ and $R_2$ by adding communications according to our coordination principle. All transitions of $R$ are considered iteratively: one transition per iteration.

$L_{6-7}$ : at each iteration, $x$ denotes the event and $q, r$ the origin and destinations states of the considered transition $Tr$. $i$ denotes the site of $x$ (i.e., $x$ is observed by $R_i$), and $j$ denotes the other site.

$L_{8}$ formalizes a condition that is true when at least one of the following two cases is satisfied: 1) Transition $[q; x; r]$ is not a selfloop and can be followed by the observation of an event by $R_j$; 2) Transition $[q; x; r]$ is followed by the execution of action(s) by $R_j$.

$L_{8-10}$ : When the condition of $L_8$ is satisfied, a communication from $R_i$ to $R_j$ is inserted between $x$ and the following event or action. The sending and reception are realized in $L_9$ and $L_{10}$, respectively.

$L_9$ : in $R_i$, a sending by $R_i$ for $R_j$ is inserted after the event $x$. The sent message indicates the current global state $r$ of $R$.

$L_{10}$ : in $R_j$, the event $x$ is replaced by the reception by $R_j$ of a message coming from $R_i$. The received message indicates the current global state $r$ of $R$.

$L_{11}$ processes the case where the condition of $L_8$ is not satisfied: no communication is needed. We have just to replace in $R_i$ the event $x$ by a spontaneous event $e$.

$L_{14-15}$ : Events $e$ are hidden in each $R_i$, and then the latter is determinized and minimized. These three operations necessitate more explanations.

Let a sending (resp. reception, empty) transition denote a transition labeled by an event $s_i^1(k)$ (resp. $r_i^1(k)$, $e$)

### 3.2.1 Hiding empty transitions in $L_{15}$

Let $R_1$ and $R_2$ be the AwAs constructed by $L_{1-15}$.

**Lemma 3.1** The event $e$ is hideable in each $R_i$.

**Lemma 3.2** In each $R_i$:

- the destination state of an empty transition is associated to no action.
- the destination state of a non-selfloop empty transition is the origin state of only reception or empty transitions.
- the origin state of an empty transition is the destination state of only sending, reception, or empty transitions.

Hiding empty transitions in a finite state automaton (FSA) (i.e., AwA without actions) can be done using the well-known natural projection [3, 7]. Due to Lemma 3.2, we can adapt the natural projection to hide empty transitions in the AwA $R_i$ by using the following Lemma:

**Lemma 3.3** We can hide empty transitions in each $R_i$ by:

- applying the natural projection of FSA and ignoring (but not removing) actions; and then
- associating the actions of $(F(k) \cap \Gamma_i)$ to every destination state of $s_i^1(k)$ or $r_i^1(k)$.

Thus, if we apply the above operations to $R_i$, its language $L_i$ becomes $L_i^\emptyset$.

#### 3.2.2 Determinization in $L_{16}$

**Lemma 3.4** The AwA obtained after the hiding of $L_{15}$ can be determinized like a FSA, i.e., ignoring (but not removing) the actions associated to states.

#### 3.2.3 Minimization in $L_{17}$

The objective of minimization is to “merge” equivalent states of an automaton. When the latter is a FSA, then well-known minimization algorithms [3, 7] can be applied, where the following definition of equivalent states is used:

Two states $q_1$ and $q_2$ of a FSA are said equivalent when every event sequence is executable from $q_1$ iff it is executable from $q_2$.

**Lemma 3.5** A AwA can be minimized by using the same minimization algorithms of FSA, provided that the definition of equivalent states is adapted as follows: Two states $q_1$ and $q_2$ of a AwA are said equivalent when every trace (see Def. 2.1) is executable from $q_1$ iff it is executable from $q_2$.

That is, two states $q_1$ and $q_2$ of a AwA are equivalent iff: 1) every event sequence is executable from $q_1$ iff it is also executable from $q_2$ (equivalence in FSA); and 2) two states $r_1$ and $r_2$ are associated to the same actions if they are reached by the same event sequence from $q_1$ and $q_2$, respectively. We have also the following lemma:

**Lemma 3.6** The synthesis algorithm generates two AwAs $R_1$ and $R_2$ that satisfy hypotheses 2.2 and 2.3.

### 3.3. Correctness and application of the synthesis algorithm

**Theorem 1** Given a AwA $R$, the synthesis algorithm generates two AwAs $R_1$ and $R_2$ that satisfy the objective of Sect. 2.5, that is: $(R_1 \land R_2)^R_1 = R$.

If we apply the synthesis algorithm to the $R$ of Fig. 2, we obtain the $R_1$ and $R_2$ of Fig. 3. For the sake of clarity, these $R$, $R_1$, $R_2$ are represented together in Fig. 5. Here are some explanations on how this result is obtained:
4. Improvement and applicability of the coordination method

In this section, we improve the coordination method by removing some useless communications, and discuss on its applicability for nonnegligible reaction delays and communication delays.

4.1. Reducing communications

Clearly, our coordination method generates less communications than the one where components inform each other of all what they observe. Let us see why it may nevertheless generate some unnecessary communications. We have explained in Sect. 3 that the principle of coordination is as follows:

- Transition \([1; a_1; 2]\) of \(R\) is executed in Site 1 and followed only by events and actions in the same site (event \(b_1\) and actions \(\gamma_1, \delta_1\)), and thus, no communication needs to follow this transition.

- Transition \([2; b_1; 3]\) of \(R\) is executed in Site 1 and can be followed by the event \(d_2\) in Site 2, and thus, a communication from \(R_1\) to \(R_2\) must follow this transition. The sending part of this communication corresponds to the transition \([3; 3−4; s_1^2(3); 3−4]\) in \(R_1\), and the reception part corresponds to the transitions \([1−2; r_2^2(3); 3−2]\) and \([3−2; r_2^4(3); 3−2]\) in \(R_2\).

- Transition \([3; c_1; 2]\) of \(R\) (similarly to \([1; a_1; 2]\)) is executed in Site 1 and followed only by events and actions in the same site, and thus, no communication needs to follow this transition.

- Transition \([3; d_2; 4]\) of \(R\) is executed in Site 2 and followed only by event \(c_2\) in the same site, and thus, no communication needs to follow this transition.

- Transition \([4; c_2; 5]\) of \(R\) is executed in Site 2 and followed by events and actions in Site 1 (events \(b_1\) and \(c_1\), and action \(\delta_1\)), and thus, a communication from \(R_2\) to \(R_1\) must follow this transition. The sending and reception parts of this communication correspond to the transitions \([4−5; s_2^1(5); 5]\) and \([3−4; r_1^2(5); 5]\) in \(R_2\) and \(R_1\) respectively.

- Transitions \([5; b_1; 3]\) and \([5; c_1; 3]\) of \(R\) (similarly to \([2; b_1; 3]\)) are executed in Site 1 and can be followed by the event \(d_2\) in Site 2, and thus, a communication from \(R_1\) to \(R_2\) must follow these transitions. The sending and reception parts of this communication correspond to the transitions \([5−3; s_1^2(3); 3−4]\) and \([5; r_2^2(3); 3−2]\) in \(R_1\) and \(R_2\) respectively.

Figure 5. Example of application of the synthesis algorithm

If in \(R\), the observation by \(R_i\) of an event \(x_i\) (in Site \(i\)) is followed in Site \(j\) by:

1) the observation by \(R_j\) of an event \(y_j\), or
2) the execution by \(R_j\) of an action \(\xi_j\);

Then a communication from \(R_i\) to \(R_j\) is inserted between \(x_i\) and \(y_j\) or \(\xi_j\). Each transmitted message contains the current global state of \(R\).

Communication due to the above 1) (i.e., \(x_i\) followed by \(y_j\)) guarantees that each component knows the global state of \(R\) when it has to send a message (containing the global state). When a message due to 2) (i.e., \(x_i\) followed by \(\xi_j\)) is received by \(R_j\), the latter is informed of the global state and deduces that it must execute the action \(\xi_j\). In some cases, \(R_j\) does not need to know the current global state to deduce the action(s), if any, it has to execute. It may be sufficient that the component knows that the current state belongs to a given set of states \(K\). Let us now propose a method to reduce the communications generated in \(R_i\) and \(R_j\), by identifying and removing some useless communications.

Lemma 4.1 A communication defined by \((s^i_j(k), r^i_j(k))\) is useless when \(r^i_j(k)\) occurs only in transitions:

- whose destination state is associated to no action, and
- whose origin state has no other outgoing transition.

When \((s^i_j(k), r^i_j(k))\) is useless, we also say that the corresponding sending and reception transitions are useless in \(R_i\) and \(R_j\), respectively.

Lemma 4.2 The set of useless (sending, reception) transitions of each \(R_i\) is hidable.

From Lemmas 2.1 and 4.2, we deduce that if \(L\) is the language of the AwA \(R_i\) and \(U\) is its set of useless transitions, then there exists a AwA whose language is \(L^U\).
Lemma 4.3 We hide useless transitions from $R_1$: by applying to useless sending transitions the same operations as in Lemma 2.4, by removing useless selfloop reception transitions, and by applying the following operations to every useless non-selfloop reception transition $Tr$:

- the destination state $r$ of $Tr$ is split into two states $r_1$ and $r_2$ such that: $r_1$ becomes the destination state of $Tr$, $r_2$ becomes the destination state of the other transitions whose destination state was $r$, and $r_1$ and $r_2$ become the origin states of the transitions whose origin state was $r$.
- $Tr$ is removed and its origin and destination states ($q$, $r_1$) are merged into a single state $q_{\rightarrow r_1}$; 
- $q_{\rightarrow r_1}$ becomes the destination state of all the transitions whose destination state was $q$;
- $q_{\rightarrow r_1}$ becomes the origin state of all the transitions whose origin state was $r_1$; 
- we associate to $q_{\rightarrow r_1}$ the actions (if any) that were associated to $q$.

If we hide useless communications from the ($R_1$, $R_2$) of Figure 5, we obtain the ($R_1$, $R_2$) of Figure 6. Note that in this example, we obtain a minimal communication.

Figure 6. Reducing communications in the components of Fig. 5

4.2. Adaptation when reaction delays and communication delays are nonnegligible

We have seen that the principle of coordination is as follows: if in $R$, an event $x_i$ in site $i$ is followed by an event $y_j$ (resp. an action $\xi_j$) in site $j$, then a sending by $R_i$ must follow the event $x_i$ and be received by $R_j$ before $y_j$ (resp. $\xi_j$). If the delay between $x_i$ and the reception of the message is equal to a value $\tau$, this implies that the delay between $x_i$ and $y_j$ or $\xi_j$ must be greater than $\tau$. In order to respect this requirement, we have assumed that $\tau$ is negligible and can be approximated by 0. When $\tau$ has a nonnegligible upper bound $D$, a sufficient condition for the applicability of our coordination method is thus stated by the following lemma:

Lemma 4.4 The coordination method of Sect. 3 is applicable if the distributed system $R$ stays during at least $D$ in every state requiring communication, i.e., satisfying the condition of $L_S$ of the synthesis algorithm (see explanations of $L_S$ in Sect. 3.2).

5. Application in supervisory control of DES

Our coordination method can be applied to distributed conformance testing, where the components are testers and their actions consist of selecting inputs to be applied and generating verdicts [8]. The coordination method can also be applied to distributed supervisory control of DES, where the components are supervisors and their actions consists of enabling and disabling events [15]. In more advanced control systems, actions can also consist of forcing events, specifically in the presence of timing constraints [4, 11]. And in yet more advanced control systems, actions can also consist, for example, of starting or terminating jobs defined by continuous variables and differential equations, such as starting regulating the speed of a dynamic system, or modifying or terminating an ongoing regulation.

Let us see in more detail how our coordination method can be applied in supervisory control of DES. The inputs of the problem are a plant $P$ and a specification $S$. In the centralized case (i.e., all events executed in the same site), the objective is to synthesize a supervisor that restricts the behavior of $P$ (by disabling events) so that it respects $S$. The supervisor must be optimal in the sense that it disables events only when necessary. In the distributed case (i.e., events executed in different sites), we propose to proceed in three steps: 1) we assume the centralized case (i.e., sites are not distinguished) and we synthesize a supervisor Sup; 2) we use our coordination method to synthesize supervisors Sup1 and Sup2 from Sup; and then 3) we remove some useless communications as indicated in Sect. 4.1. This approach is illustrated by the plant and the specification represented on Fig. 7. After Step 1, if $b_1$ is the only uncontrollable event, we obtain the supervisor represented on the same Figure. In our representation, we have used the following semantics of enabling and disabling: an event that has been enabled (resp. disabled), remains enabled (resp. disabled) until its disabling (resp. enabling). Note that the obtained supervisor is similar to $R$ of Fig. 5, provided that: $a_1$, $\beta_1$, $\gamma_1$, $\delta_1$ mean respectively: "enable $a_1$", "enable $b_1$", "disable $c_1$", "enable $d_1$"; $\gamma_1$, $\delta_1$ mean respectively: "disable $c_1$", "disable $d_1$"; $\nu_2$, $\nu_2$ mean respectively: "enable $e_2$", "disable $e_2$".

Therefore, after Step 3 we obtain the supervisors Sup1 and Sup2 of Fig. 8 which are similar to the components of Fig. 6.

6. Conclusion

Interesting work on coordination has been done in decentralized supervisory control [24, 23, 1, 16, 17, 2, 18, 19] and diagnosis [6, 21]. We have developed a new coordination method which is better than [19] (the latter being the
We plan to implement the coordination method and then apply it to concrete nontrivial systems, for example in conformance testing and in supervisory control. We also plan to study its generalization for real-time DES.

References