Towards Distributed Control of Discrete-Event Systems[⋆]

Philippe Darondeau[1] and S.L. Ricker[2]

[1] INRIA Rennes-Bretagne Atlantique, Campus de Beaulieu, Rennes, France
darondeau@irisa.fr
[2] Mathematics & Computer Science, Mount Allison University, Sackville NB, Canada
lricker@mta.ca

Abstract. To initiate a discussion on the modeling requirements for distributed control of discrete-event systems, a partially-automated region-based methodology is presented. The methodology is illustrated via a well-known example from distributed computing: the dining philosophers.

1 Decentralized, Asynchronous, and Distributed DES Control

In this section, we explain our understanding of the gradual evolution of control for discrete-event systems (DES) from centralized control to more advanced forms, including distributed control, an area still in its infancy but one that may soon become vital for practical applications of the theory.

Ramadge and Wonham's theory of non-blocking supervisory control for DES was introduced in [23] and developed soon after in [18, 24] to cover situations in which only some of the events generated by the DES are observed, for instance "in situations involving decentralized control" [24]. The basic supervisory control problem consists of deciding, for a DES with observable/unobservable and controllable/uncontrollable events, whether a specified behavior may be enforced on the DES by some admissible control law. This basic supervisory control problem is decidable. Moreover, if every controllable event is also observable, then one can effectively compute the largest under-approximation of the specified behavior that can be enforced by the supervisory control.

After the work done in [28] and extended in [35], *decentralized control* refers to a form of control in which several local supervisors, with different subsets of observable events and controllable events, cooperate in rounds. In each round, the local supervisors compute a set of authorized events and the DES performs authorized events until one is observed by some local supervisor, which puts an end to the

[⋆] Work partially supported by the EC-FP7 under project DISC (Grant Agreement INFSO-ICT-224498) and NSERC.

round. In each round, the cooperation between the supervisors may result from an implicit synchronization or arbitration, e.g., in conjunctive coordination, or it may be obtained by applying distributed algorithms, e.g., to reach a consensus on the set of authorized events. In any case, the controlled DES is governed by a global clock, whose ticks are the events whose occurrences start the rounds. This global clock may not be a problem for embedded system controllers, especially when they are designed using synchronous programming languages like Lustre [12]. Yet global clocks are impractical for widely distributed systems or architectures, not to mention for distributed workflows or web services. For this reason Globally Asynchronous Locally Synchronous (GALS) architectures have been developed [22]. The existence of non-blocking decentralized supervisory control for the basic supervisory control problem is decidable in the special case where the closed behavior of a DES is equal to the closure of its marked behavior, but it is undecidable in the general case without communication [16, 30].

Decentralized control can be extended to the case where supervisors communicate [3]. When the correct control decision cannot be taken by any of the local supervisors at the end of one of the cooperative rounds of decision making, communication between supervisors can be introduced. The basic idea of this class of problems is to ensure that supervisors receive relevant information about system behavior that they cannot directly observe so that (at least) one local supervisor can make the correct control decision. Further, it is often desirable to synthesize an optimal communication protocol, either from a set-theoretic (e.g., [27]) or quantitative (e.g., [26]) perspective. Thus, supervisors make control decisions based on not only their partial observations, but also on information received from other supervisors.

For the most part, work in this area has been devoted to the study of synchronous communication protocols. There are several different approaches: build a protocol using a bottom-up approach [3, 25]; given a correct communication protocol, use a top-down approach to reduce it to one that is optimal from a set-theoretic perspective [27, 32]; and distribute an optimal centralized solution among a set of communicating supervisors [19]. Bounded and unbounded communication delays have also been explored [31, 13]. The communication channel is modeled as a FIFO buffer; however, there is no notion of optimality since the communication protocol for each supervisor is to simply send all of its observations to the other supervisors. In the case of unbounded delay, the problem is undecidable [31, 13].

Decentralized control with communication extends the basic decentralized control problem in the general direction of distributed control; however, the synthesis of asynchronous communication protocols remains largely unexamined. Of some interest is the class of models where asynchronous interaction does not require the intervention of an *arbitrator* [17]. At present, though, none of the strategies for synthesizing communication protocols in decentralized control of DES meets the conditions of asynchronous communication required for distributed control.

Asynchronous control was investigated in [21] in the framework of recognizable trace languages and asynchronous automata [37, 20, 8]. In an asynchronous automaton, states are vectors of local states, and transitions are defined on *partial* states, i.e., projections of states on a subset of locations. Transitions that concern disjoint subsets of locations produce independent events, hence asynchronous automata have *partially ordered runs* (*ergo* there is no global clock). In [21], a DES is an asynchronous automaton where an uncontrollable event has exactly one location while a controllable event may have several locations, and every event is observed in all locations concerned in the generating transition. The goal is to construct for all locations *local supervisors* that jointly enforce a specified behavior on the DES, i.e., a specific subset of partially ordered runs. Local supervisors cooperate in two ways. First, a controllable event with multiple locations cannot occur in the controlled DES unless it is enabled by all supervisors in these locations. Second, at each occurrence of a controllable event, all local supervisors concerned with this event synchronize and communicate to one another all information that they have obtained so far by direct observation of local events and from prior communication with other supervisors. The information available to the supervisor in location ℓ is therefore the ℓ-view of the partially ordered run of the DES, i.e., the causal past of the last event with location ℓ. The local control in location ℓ is a map from the ℓ-view of runs to subsets of events with location ℓ, including all uncontrollable events that are observable at this location. Implementing asynchronous supervisors, as they are defined above, on distributed architectures requires implementing first a protocol for synchronizing the local supervisors responsible for an event whenever the firing of this event is envisaged. One of the main contributions of [21] is the identification of conditions under which one can effectively decide upon the existence of asynchronous supervisors, which, moreover, can always be translated to *finite* asynchronous automata.

A different form of asynchronous control has been examined in [5]. Here the goal is not to compute from scratch an asynchronous supervisor but, rather, to transform a centralized (optimal and non-blocking) supervisor into an equivalent system of local supervisors with *disjoint* subsets of controllable events. The data for the problem are a DES G and a centralized supervisor S, defined over a set Σ of observable/unobservable and controllable/uncontrollable events, plus a partition $\Sigma = \uplus_{\ell \in \mathcal{L}} \Sigma_\ell$ of Σ over a finite set of locations. All local supervisors have the set of events Σ, but the local supervisor S_ℓ in location ℓ is the sole supervisor that can disable controllable events at location ℓ. The states of S_ℓ are the cells of a corresponding *control cover*, i.e., a covering of the set of states of S by *cells* with the following properties. First, if two transitions from a cell are labelled by the same event, then they must lead to the same cell. Second, if two states x and x' are in the same cell, then for any reachable states (x, q) and (x', q') of $S \times G$ and for any event e with location ℓ, if e is enabled at q and q' in G, then e should be coherently enabled or disabled at x and x' in S. Third, two states in a cell should be consistent w.r.t. marking information. Supervisor localization based on control covers is universal in the sense that it always succeeds in transforming a centralized supervisor S into an equivalent family of local supervisor S_ℓ. It is worth noting that a local supervisor S_ℓ with language $L(S_\ell)$ may be replaced equivalently by a local supervisor with the language $\pi(L(S_\ell))$ for any natural projection operator $\pi : \Sigma^* \to (\Sigma'_\ell)^*$ such that $\Sigma_\ell \subseteq \Sigma'_\ell \subseteq \Sigma$ and $L(S_\ell) = \pi^{-1}\pi(L(S_\ell))$. In the end, for any event $e \in \Sigma$, whenever e occurs in the controlled DES, all local supervisors S_ℓ such that $e \in \Sigma'_\ell$ should perform local transitions labelled by e *in a synchronized way.*

The approaches taken in [21] and [5] differ significantly. The former approach is based on asynchronous automata and on the assumption that several local supervisors may be responsible for the same controllable event. The latter approach is based on product automata and on the assumption that exactly one local supervisor is responsible for each controllable event. Both approaches, though, rely on similar requirements for the final implementation of the asynchronous supervisors. In both approaches, the only way for the local supervisors to communicate with one another is to synchronize on shared events. As a matter of fact, widely distributed architectures do not generally provide protocols for performing synchronized transitions in different locations.

By *distributed control*, we mean a situation in which the local supervisors and the DES cooperate as follows. First, the set of observable or controllable events of the DES is partitioned over the different locations ($\Sigma = \uplus_{\ell \in \mathcal{L}} \Sigma_\ell$), and for each location $\ell \in \mathcal{L}$, every event in Σ_ℓ results from the synchronized firing of two e-labelled transitions, in the DES and in the local supervisor S_ℓ, respectively. Second, the set of events of each local supervisor S_ℓ is the union of Σ_ℓ and a set X_ℓ of auxiliary *send* and *receive* events, used to communicate with the other local supervisors in *asynchronous message passing mode.* Messages sent from one location to another are never lost; however, they may overtake one another and the arrival times cannot be predicted. Distributed controllers of this type, already suggested in [6], can be implemented almost directly on *any* distributed architecture. A slightly different type of distributed controllers, in which the local supervisors communicate by FIFO buffers, has recently been proposed in [15] together with synthesis algorithms for avoiding forbidden states in infinite systems. Prior work along the same line was presented in [34], where the goal was distributed state estimation in finite DES instead of distributed control.

In this paper, we do not intend to present a new theory, but, rather, we use a well-known example with which to illustrate the concept of *distributed control*, an area that is not yet fully understood, with the hope of motivating further research in this direction. The example chosen is the $n = 3$ version of the classical n dining philosophers problem, where the local supervisors are in the forks and both hands of each philosopher may act concurrently. The techniques that we use for constructing distributed controllers for this problem are drawn or adapted from [6], and they are briefly recalled in the next section before they are put into practice.

2 Distributed Controller Synthesis Based on Petri Net Synthesis

In this section, we describe the background of Petri net based controller synthesis and its extension to distributed controller synthesis.

Given a DES G over a set of events Σ, called the *plant*, let *Spec* be a specification of the desired behavior of G. Let $\Sigma = \Sigma_o \uplus \Sigma_{uo} = \Sigma_c \uplus \Sigma_{uc}$, where Σ_o and Σ_c are the sets of observable and controllable events, respectively. Henceforth, we assume that every controllable event is observable, i.e., $\Sigma_c \subseteq \Sigma_o$. Then the supervisory control problem consists of constructing a supervisor S over the set of events Σ_o such that the partially synchronized product of S and G, usually denoted by $S \times G$ but here denoted by (S/G), satisfies *Spec* and the following *admissibility* condition holds: for every reachable state (x, q) of (S/G), if an uncontrollable event $e \in \Sigma_{uc}$ is enabled at state q in G, then the event e should also be enabled at state x in S.

Let $S = (X, \Sigma_o, \delta, x_o)$ be a deterministic supervisor satisfying the above requirements, where X is the set of states, $x_0 \in X$ is the initial state, and $\delta : X \times \Sigma_o \to X$ is the partial transition map. For simplicity, we do not consider marked states here. We say that S is *Petri net definable* (PND) if it is isomorphic to the reachability graph of a Petri net system N with the set of transitions $T = \Sigma_o$. Let us recall some definitions.

Definition 1. *A* Petri net *is a bi-partite graph* (P, T, F)*, where* P *and* T *are* finite *disjoint sets of vertices, called* places *and* transitions*, respectively, and* $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ *is a set of directed edges with non-negative integer weights. A* marking *of* N *is a map* $M : P \to \mathbb{N}$*. A transition* $t \in T$ *is* enabled at *a marking* M *(denoted by* $M[t\rangle$*) if* $M(p) \geq F(p, t)$ *for all places* $p \in P$*. It* t *is enabled at* M*, then it can* be fired*, leading to the new marking* M' *(denoted by* $M[t\rangle M'$*) defined by* $M'(p) = M(p) + F(t, p) - F(p, t)$ *for all* $p \in P$*.*

Definition 2. *A* Petri net system N *is a tuple* (P, T, F, M_0) *where* M_0 *is a marking of the underlying net* (P, T, F)*, called the initial marking. The* reachability set $RS(N)$ *of* N *is the set of all markings reached by sequences of transitions from* M_0*. The* reachability graph $RG(N)$ *of* N *is the transition system* $(RS(N), T, \delta, M_0)$ *with the partial transition map* $\delta : RS(N) \times T \to RS(N)$ *defined as* $\delta(M, t) = M'$ *iff* $M[t\rangle M'$ *for all markings* M *and* M' *in* $RS(N)$*.*

Thus, S is PND if there exists a Petri net system $N = (P, T, F, M_0)$ with the set of transitions $T = \Sigma_o$ and a bijection $\varphi : X \to RS(N)$ such that $\varphi(x_0) = M_0$ and for all $x \in X$ and $t \in \Sigma_o$, $\delta(x, t)$ is defined if and only if $M[t\rangle M'$ for $M = \varphi(x)$ and for some marking M', and then $M' = \varphi(x')$ where $x' = \delta(x, t)$.

PND controllers comprised of *monitor places* for PND plants date back to the work in [7, 36]. Monitor places are linear combinations of places of the net which define the plant, and they are added to the existing places of this net in order to constrain its behavior. PND controllers for DES given as labelled transition systems proceed from a similar idea with one significant difference: since G has no places, the places of the controller net N are synthesized directly from $Spec$ (the specification) using the theory of regions [1]. This controller net synthesis technique was inaugurated in [10]. In that work, the specification $Spec$ was the induced restriction of G on a subset of "good" states. The theory of regions may be applied both to state-oriented specifications given by transition systems and to event-oriented specifications given by languages. It is shown in [6] how the theory can be applied to the supervisory control problem with tolerance stated as follows:

Given a plant G and two prefix-closed regular languages L_{min} and L_{max}, such that $L_{min} \subseteq L_{max} \subseteq L(G)$, where $L(G)$ denotes the language generated by G, decide whether there exists and construct a Petri net controller N such that

$$L_{min} \subseteq L((RG(N)/G)) \subseteq L_{max}$$

After attempts to apply the theory to significant examples found in the literature (e.g., workflows, train systems), we convinced ourselves that in most cases where supervisory control is needed, the main objective is to avoid deadlocks or to enforce home states or both, but it is simply impossible to express such objectives by tolerance specifications because L_{min} is not known beforehand. The example developed in the next section also illustrates this situation.

At this stage, it may seem odd to search for PND controllers instead of general controllers defined by labelled transition systems. Indeed, the Petri nets that we consider are labelled injectively (their set of transitions T is equal to the set Σ_o of the observable events), and it is well-known that languages of injectively labelled and bounded Petri nets form a *strict subset* of the regular languages. So, while one loses generality, one gains a more compact representation of controllers, which appears to be poor compensation. Fortunately, things change in a radical way when one takes distributed control into account, since one can tailor controller synthesis to distributed Petri nets that can be converted to asynchronously communicating automata, as we explain below.

To begin with, we recall from [2] the definition of distributed Petri nets and their automated translation into asynchronous message passing automata.

Definition 3 (Distributed Petri net system). *A* distributed *Petri net system over a set of locations $\mathcal{L}$ is a tuple $N = (P, T, F, M_0, \lambda)$ where (P, T, F, M_0) is a Petri net system and $\lambda : T \to \mathcal{L}$ is a map, called a* location map*, subject to the following constraint: for all transitions $t_1, t_2 \in T$ and for every place $p \in P$, $F(p, t_1) \neq 0 \wedge F(p, t_2) \neq 0 \Rightarrow \lambda(t_1) = \lambda(t_2)$.*

In a distributed Petri net, two transitions with different locations cannot compete for tokens, hence *distributed conflicts cannot occur*, which makes the effective implementation easy. Two transitions with different locations may, though, *send* tokens to the same place. Implementing a distributed Petri net system N means producing an asynchronous message passing automaton ($AMPA$) behaving like N up to branching bisimulation (see Appendix for precise definitions). Given any non-negative integer bound B, let $RG_B(N)$ denote the induced restriction of the reachability graph $RG(N)$ of N on the subset of markings bounded by B, i.e., markings M such that $M(p) \leq B$ for all places $p \in P$. Transforming N into a B-bounded $AMPA$ with a reachability graph branching bisimilar to $RG_B(N)$ may be done as follows (see [2] for a justification).

- Given $N = (P, T, F, M_0, \lambda)$, extend $\lambda : T \to \mathcal{L}$ to $\lambda : (T \cup P) \to \mathcal{L}$ such that $\lambda(p) \neq \lambda(t) \Rightarrow F(p,t) = 0$ for all $p \in P$ and $t \in T$ (this is always possible by Def. 3).
- For each location $\ell \in \mathcal{L}$, construct a net system $N_\ell = (P_\ell, T_\ell, F_\ell, M_{\ell,0})$, called a *local net*, as follows.
  - $P_\ell = \{p \mid p \in P \wedge \ell = \lambda(p)\} \cup \{(p,\ell) \mid p \in P \wedge \ell \neq \lambda(p)\}$,
  - $T_\ell = \{t \in T \mid \lambda(t) = \ell\} \cup$
    $\{\ell\,!\,p \mid p \in P \wedge \ell \neq \lambda(p)\} \cup \{\ell\,?\,p \mid p \in P \wedge \ell = \lambda(p)\}$,
  - $F_\ell(p,t) = F(p,t)$ and $F_\ell(t,p) = F(t,p)$,
  - $F_\ell(t,(p,\ell)) = F(t,p)$,
  - $F_\ell((p,\ell), \ell\,!\,p) = 1$ and $F_\ell(\ell\,?\,p, p) = 1$,
  - $M_{\ell,0}(p) = M_0(p)$.

  In each local net N_ℓ, places (p,ℓ) are local clones of places p of other local nets. Whenever a transition $t \in T$ with location ℓ produces tokens for a *distant* place $p \in P$ ($\lambda(t) = \ell \neq \lambda(p)$), the transition $t \in T \cap T_\ell$ produces tokens in the local clone (p,ℓ) of p. These tokens are removed from the local clone (p,ℓ) of p by the auxiliary transition $\ell\,!\,p$, each firing of which models an asynchronous emission of the *message* p. Symmetrically, for any place p of N with location ℓ, each firing of the auxiliary transition $\ell\,?\,p$ models an asynchronous reception of the *message* p, resulting in one token put in the corresponding place $p \in P_\ell$.
- For each location ℓ, compute $RG_B(N_\ell)$. The desired $AMPA$ is the collection of local automata $\{A_\ell = RG_B(N_\ell) \mid \ell \in \mathcal{L}\}$. Each transition of an automaton A_ℓ is labelled either with some $t \in T \cap T_\ell$ or with some asynchronous message emission $\ell\,!\,p$ or with some asynchronous message reception $\ell\,?\,p$. A message p sent from a location ℓ' by a transition $\ell'\,!\,p$ of the automaton $A_{\ell'}$ is automatically routed towards the automaton A_ℓ with the location $\ell = \lambda(p)$, where it is received by a transition $\ell\,?\,p$ of the automaton A_ℓ. No assumption is made on the relative speed of messages, nor on the order in which they are received.

Closing the parenthesis, let us return to controllers. Let G be a DES over a set of events $\Sigma = \Sigma_o \uplus \Sigma_{uo} = \Sigma_c \uplus \Sigma_{uc}$ where $\Sigma_c \subseteq \Sigma_o$. Let $Spec$ be a specification of the desired behavior of G. Let $\mathcal{L}$ be a finite set of locations (or sites). Let $\lambda : \Sigma_o \to$

$\mathcal{L}$ be a location map, specifying for each observable event e the location $\lambda(e)$ in which it may be observed and possibly controlled. Let $S = (X, \Sigma_o, \delta, x_o)$ be a finite-state admissible supervisor for G, such that (S/G) satisfies the specification *Spec*. Thus, for every reachable state (x, q) of (S/G), if $e \in \Sigma_{uc}$ is enabled in state q in G, then e is also enabled in state x in S. We say that S is a *distributed Petri net definable* supervisor (DPND) if it is isomorphic to the reachability graph of a distributed Petri net $N = (P, T, F, M_0, \lambda)$ with a set of transitions $T = \Sigma_o$. In view of this isomorphism, since X is a finite set of states, there must exist a finite bound B such that $M(p) \leq B$ for all places $p \in P$ and for all reachable markings M of N. Therefore, $S = (X, \Sigma_o, \delta, x_o)$ may be translated equivalently to an $AMPA = \{A_\ell = RG_B(N_\ell) \mid \ell \in \mathcal{L}\}$, realizing, in the end, *fully distributed control.*

Therefore, the approach to distributed control that we suggest is to search for admissible DPND controllers using Petri net synthesis techniques (see [1] for a survey). With the adaptation proposed in [2], these techniques allow us to answer the following types of questions:

- Given a finite automaton over Σ_o and a location map $\lambda : \Sigma_o \to \mathcal{L}$, can this automaton be realized as the reachability graph of a distributable Petri net $N = (P, T, F, M_0, \lambda)$ with set of transitions $T = \Sigma_o$?
- Given a regular language over Σ_o and a location map $\lambda : \Sigma_o \to \mathcal{L}$, can this language be realized as the set of firing sequences of a distributable Petri net $N = (P, T, F, M_0, \lambda)$ with set of transitions $T = \Sigma_o$?

The type of net synthesis techniques that should be applied when solving the distributed supervisory control problem depends upon the specification *Spec* of the control objective. A method to derive distributed supervisors for the basic supervisory control problem with tolerances was described in [6]. This method cannot be applied when the control objective is avoiding deadlocks or enforcing home states. In fact, we do not know of any systematic method to cope with such problems. The case study presented in next section aims at motivating work in this direction. To give a flavour of the approach, let us set ourselves in the simple case where G is given by a Petri net. One proceeds by a series of trial and error. At each step, one removes states from the reachability graph of G, and checks that it is enough to add distributed monitor places to confine the behavior to the remaining states. The distribution constraints on monitor places are represented in the synthesis procedure by sign constraints depending upon the chosen location of the monitor place.

Note that AMPA derived from distributed Petri nets are a *strict subclass* of AMPA. Thus, the distributed controller synthesis techniques that we propose may fail even though the distributed supervisory control problem may be solved using more general AMPA. This point illustrates, if it was not yet totally clear, that the field for research on distributed controller synthesis is wide open.

3 Distributed Controllers for Three Dining Philosophers

We would now like to experiment with our Petri net based synthesis method for distributed controllers on a toy example. We have chosen the famous problem of the dining philosophers [14]. The reasons for this choice are twofold. On the one hand, the problem is easily understood. On the other hand, the size of this problem can be adjusted to accommodate our partially-automated processing techniques by decreasing the usual number of philosophers, which is five, to three philosophers.

Let us recall the statement of the problem. Three philosophers φ_1, φ_2, and φ_3 are sitting at a table with a bowl of spaghetti in the center. Three forks f_1, f_2, f_3 are placed on the table, such that philosopher ϕ_i has the fork f_i on his right and the fork $f_{i+1 \mod 3}$ on his left (see Fig. 1(a)). A philosopher alternates periods of eating and periods of thinking. To eat, he needs both the fork to his direct left and the fork to his direct right. Therefore, he tries to grab them one after the other while thinking. A philosopher who thinks with a fork in each hand stops thinking and starts eating after a finite delay. A philosopher who eats eventually stops eating, puts down the forks, and starts thinking again after a finite delay. The basic problem is to avoid deadlock, i.e., the situation in which every philosopher has taken one fork. A classical solution is to let all philosophers but one, say φ_1, take first the fork to their right, and to let φ_1 take first the fork to his left. An augmented problem is to avoid the starvation of any philosopher.

The dining philosophers problem was used in [33] to illustrate the use of supervisory control techniques for removing deadlocks from multi-threaded programs with lock acquisition and release primitives. In that work, the emphasis was on optimal control, not on distribution. In this paper, we do the opposite, i.e., we give priority to distributed control over optimal control. Moreover, *we intend that local controllers will be embedded in forks*, not in philosopher threads. The idea is that resource managers have fixed locations while processes using resources are mobile. To make things precise, consider the PND plant G defined by the Petri net N (shown in Fig. 1(b)).

Places $f1f$, $f2f$, and $f3f$ are the resting places of the forks $f1$, $f2$, and $f3$, respectively (fif should be read as "f_i is free"), and they are initially marked with one token each. For $j \in \{1, 2, 3\}$ and $i \in \{j, j+1 \mod 3\}$, "philosopher φ_j can take fork f_i when it is free" (transition tji). After this, "philosopher φ_j holds fork f_i" (condition jhi). A philosopher φ_j with two forks may "start eating" (transition sej). A philosopher φ_j who is eating (condition ej) can "start thinking" (transition stj), and then forks f_j and $f_{j+1 \mod 3}$ return again to the table. With respect to distribution, there are four locations as follows. For each fork f_i, $i \in \{1, 2, 3\}$, the two transitions which compete for this fork, namely tii and tji with $j = i - 1 \mod 3$, are located on a site i specific to this fork. All other transitions are located on a default site 4 that does not matter.

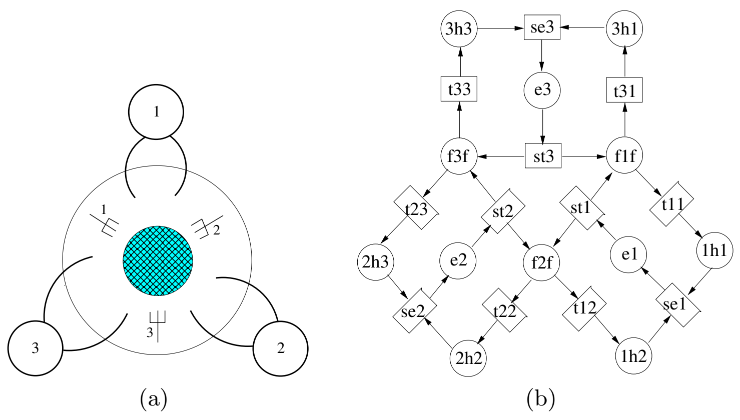


Fig. 1. Modeling the Dining Philosophers Problem:(a) three philosophers; (b) the uncontrolled plant

The control objective is to avoid deadlocks. For simplicity, we assume that all transitions are observable. The controllable transitions are the transitions which consume resources, i.e., the transitions tji (philosopher φ_j takes fork f_i). The control objective should be achieved by distributed control and, more precisely, by a distributed Petri net. The set of transitions T of the controller net may be any set included in $\{sej, stj, tjj, tji \mid 1 \leq j \leq 3 \wedge i = j+1 \mod 3\}$. The location map $\lambda : T \to \{1,2,3,4\}$ is naturally the induced restriction of the map defined by $\lambda(tij) = j$ and $\lambda(sej) = \lambda(stj) = 4$ for $1 \leq j \leq 3$.

The PND plant G under consideration, i.e., the reachability graph of N, has two sink states M_1 and M_2, defined by $M_1(3h1) = M_1(1h2) = M_1(2h3) = 1$ and $M_2(1h1) = M_2(2h2) = M_2(3h3) = 1$, respectively (letting $M_1(p) = 0$ or $M_2(p) = 0$ for all other places p). If one disregards *distributed* control, it is quite easy to eliminate these two deadlocks by adding two monitor places p_1 and p_2, the role of which is to disable the instances of the transitions $t31, t12, t23$ and $t11, t22, t33$ that reach M_1 and M_2 in one step, respectively. The monitor places may be chosen so that no other transition instance is disabled, hence they do not cause new deadlocks. The PND controller consisting of the monitor places p_1 and p_2 realizes the optimal control of G where the objective is deadlock avoidance. mbox
This solution is straightforward because N is one-safe, i.e., for every reachable marking M and for every place p of N, $M(p) \in \{0,1\}$. For any one-safe net with set of places P, the set of reachable markings is a convex subset of $\{0,1\}^P$. As a consequence, a reachable marking M_x may always be separated from all other reachable markings by a hyperplane. Any separating hyperplane induces a monitor place that disables all (instances of) transitions crossing this hyperplane in a fixed direction. Therefore, for one-safe nets, optimal control can always be realized by PND controllers [9]. The two monitor places computed by SYNET

[29] are $p_1 = 2 + st1 + st2 + st3 - t31 - t12 - t23$ (i.e., p_1 is initially marked with 2 tokens, there are flow arcs with weight 1 from $st1$, $st2$ and $st3$ to p_1, and that there are flow arcs with weight 1 from p_1 to $t31$, $t12$ and $t23$), and $p_2 = 2 + st1 + st2 + st3 - t11 - t22 - t33$. This optimal controller is unfortunately not a distributed controller.

3.1 A distributed controller avoiding deadlocks

The reachability graph G of N has 36 states and 78 transitions. An inspection of G reveals the two subgraphs shown in Fig. 2, where the initial state is numbered 0 and the deadlock states M_1 and M_2 are numbered 25 and 11, respectively. All transitions that lead to a deadlock state in one step are, in fact, represented in Fig. 2. Note that all squares in the figure are *distributed diamonds*, i.e., the north/south edges and the west/east edges of a square are always labelled by two events which belong to different locations.

To avoid reaching the deadlock states M_1 and M_2, one can proceed as follows. First, all seven instances of the transitions $t31$ and $t22$ indicated in Fig. 2 are removed (manually) from the reachability graph G of N. Let G_1 be the reachable restriction of the resulting subgraph of G. G_1 has 23 states, none of which is a sink state.

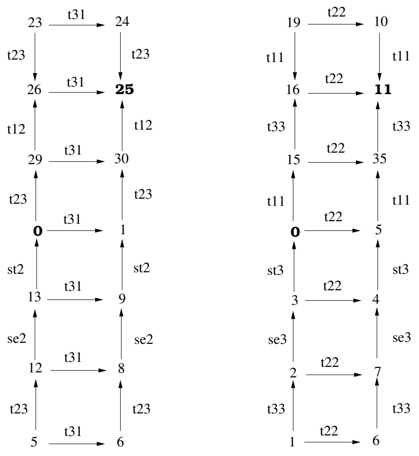
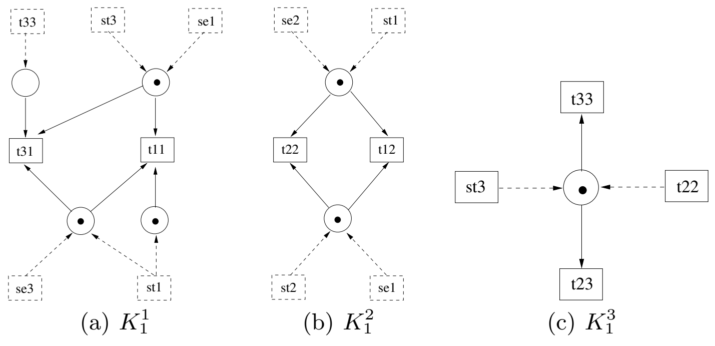


Fig. 2. Two ladder-shaped subgraphs leading to deadlock

If this problem is feasible, the software SYNET will compute a distributed controller net K_1 such that G_1 is isomorphic to $RG(K_1)/G$. This is the case for our example, and SYNET produces a DPND controller with components K_1^1, K_1^2, K_1^3, K_1^4 to plug in the respective locations 1 to 4. K_1^4 does nothing but send the other components asynchronous signals informing them of the occurrences of the *start eating* and *start thinking* events sej and stj. The local controllers K_1^1, K_1^2, K_1^3 are depicted in Fig. 3. For each controller, the asynchronous flow from other local controllers is indicated by dashed arrows.



(a) K_1^1 (b) K_1^2 (c) K_1^3

Fig. 3. Local controllers for our example

Note that K_1^2 sends messages to K_1^3 (indicating that $t22$ has occurred) and K_1^3 sends messages to K_1^1 (indicating that $t33$ has occurred) but K_1^1 does not send any message to the other components. In the design of the distributed controller K_1, we have privileged the transitions $t31$ and $t22$. As a result, in the initial state of $RG(K_1)/G$, philosopher φ_1 can grab both forks f_1 and f_2 *in parallel* while philosophers φ_2 and φ_3 can only fight over fork f_3. Similar controllers may be obtained by choosing $t12$ and $t33$, or $t23$ and $t11$, instead of $t31$ and $t22$. Unfortunately, neither K_1 nor such controllers can avoid starvation. In $RG(K_1)/G$, for each philosopher there actually exists a reachable state in which he may act fast enough to prevent the other two from ever eating! Finally, note that all places of the local controller nets K_1^1, K_1^2 and K_1^3 stay bounded by 1 in all reachable states of $RG(K_1)/G$. Therefore, to transform K_1 into an equivalent asynchronous message passing automaton $AMPA$, as indicated in Sect. 2, it suffices to compute the bounded reachability graph $RG_B(K_1)$ for the bound $B = 1$.

3.2 Another DPND controller avoiding deadlocks

A quite different distributed controller K_2 may be obtained by shifting the focus to the transitions $t31$ and $t11$. By inspecting the reachability graph G of N again, one can locate the two subgraphs shown in Fig. 4(a). All squares in the figure are distributed diamonds. A first attempt to avoid the deadlock states 25 and 11 consists of manually removing from G all occurrences of the transitions $t31$ and $t11$ indicated in Fig. 4(a). Unfortunately, the two deadlock states can still be reached after these transitions have been removed. In a second effort, the occurrences of the transitions $t12$ and $t33$ indicated in Fig. 4(b) are manually removed from G. Let G_2 be the reachable restriction of the resulting subgraph of G. G_2 has 22 states, none of which is a sink state. From G_2 SYNET produces a distributed controller net K_2 such that G_2 is isomorphic to $RG(K_2)/G$. K_2 has four components K_2^1, K_2^2, K_2^3, K_2^4 to plug in the respective locations 1 to 4. K_2^2, K_2^3 and K_2^4 do nothing but send K_2^1 asynchronous signals informing this local controller of the occurrences of the sets of events $\{t12\}$, $\{t33\}$ and $\{se1, st1, se3, st3\}$, respectively. The local controller K_2^1 is depicted in Fig. 5(a).

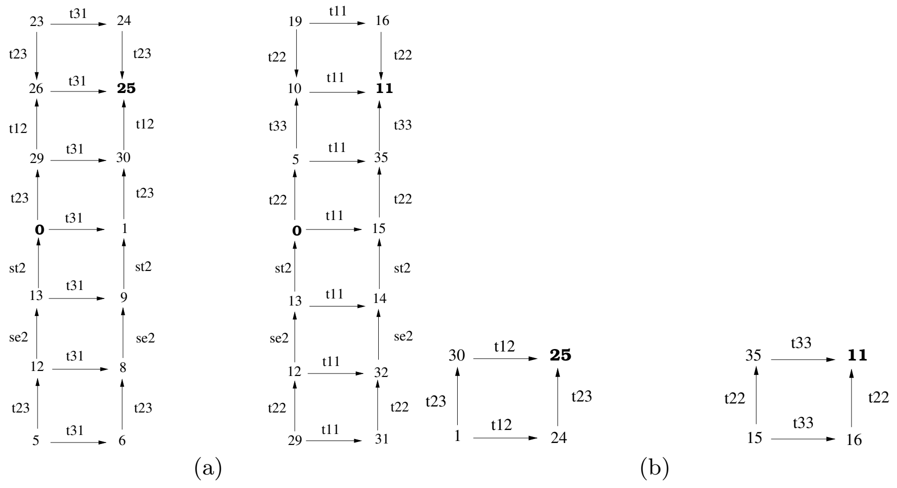


Fig. 4. Subgraphs leading to deadlock

Note that now, all control decisions are taken in location 1! In the initial state of $RG(K_2)/G$, philosopher φ_2 can take both forks f_2 and f_3 *in parallel*, while philosophers φ_1 and φ_3 can only compete with φ_2 to get forks f_2 and f_3, respectively. Note that all places of the local controller nets K_2^1 stay bounded by 1 in all reachable states of $RG(K_2)/G$. With respect to starvation, K_2 and all similar controllers have exactly the same drawbacks as K_1. K_1 and K_2 are incomparable controllers, and we suspect, but have not verified, that they are both maximally permissive amongst DPND controllers for the deadlock avoidance problem.

3.3 A distributed controller avoiding starvation

Using SYNET we produced a DPND controller K_3 that avoids starvation (and not just deadlocks). Specifically, we designed the right controller by examining the structure of cycles in the reachability graph of G, and used SYNET to confirm that it could be implemented with a distributed Petri net. The reachable state graph $RG(K_3)/G$ is shown in Fig. 5(b). Unfortunately, parallelism disappears completely, and there remains only one state where a choice is possible. It would be interesting to examine the same problem for a larger number of philosophers, but fully-automated strategies are necessary for this purpose.

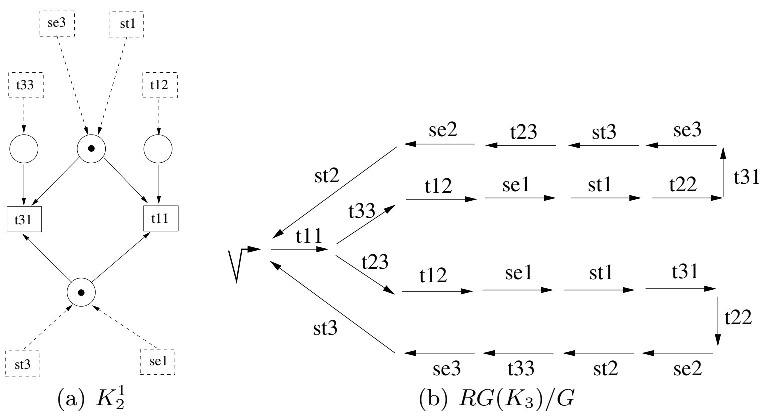


Fig. 5. (a) An unfair controller (b) A fair controller

4 Conclusion

To move towards a theory of distributed control of DES, we have proposed a mixture of asynchronous control and communication. A significant advantage of the proposed methodology is that the way of encoding the information to be exchanged is automated: the messages sent asynchronously are names of places of a Petri net produced by synthesis. Yet there remains a sizeable amount of work to be done: a significant disadvantage of the methodology is at present the lack of a general theorem and a fully-automated controller synthesis method.

References

1. E. Badouel and P. Darondeau. Theory of Regions. In *Lectures on Petri Nets I: Basic Models*, Advances in Petri Nets, LNCS 915, pages 529-586, 1998.
2. E. Badouel, B. Caillaud and P. Darondeau. Distributing Finite Automata Through Petri Net Synthesis. *Formal Aspects of Computing* 13: 447-470, 2002.
3. G. Barrett and S. Lafortune. Decentralized Supervisory Control with Communicating Controllers. *IEEE Trans. Autom. Control*, 45(9), 1620-1638, 2000.
4. D. Brand and P. Zafiropulo. On Communicating Finite-State Machines. *J. of the ACM*, 30(2):323-342, 1983.
5. K. Cai and W.M. Wonham. Supervisor Localization: A Top-Down Approach to Distributed Control of Discrete-Event Systems. *IEEE Trans. Autom. Control*, 55(3), 605-618, 2010.
6. P. Darondeau. Distributed Implementation of Ramadge-Wonham Supervisory Control with Petri Nets. In *CDC-ECC 2005*, pages 2107-2112.
7. A. Giua, F. Di Cesare and M. Silva. Generalized Mutual Exclusion Constraints on Nets with Uncontrollable Transitions. In *IEEE-SMC 1992*, pages 974-979.
8. B. Genest, H. Gimbert, A. Muscholl and I. Walukiewicz. Optimal Zielonka-Type Construction of Deterministic Asynchronous Automata. In *ICALP 2010* (2), LNCS 6199, pages 52-63.
9. A. Ghaffari, N. Rezg and X. Xie. Algebraic and Geometric Characterization of Petri Net Controllers Using the Theory of Regions. In *WODES 2002*, pages 219-224.

10. A. Ghaffari, N. Rezg and X. Xie. Design of Live and Maximally Permissive Petri Net Controller Using the Theory of Regions. *IEEE Trans. Robot. Autom.* 19: 137-142, 2003.
11. R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *J. of the ACM* 43(3): 555-600, 1996.
12. N. Halbwachs, P. Caspi, P. Raymond and D. Pilaud. The synchronous dataflow programming language Lustre. *Proc. IEEE*, 79(9):1305-1320, 1991.
13. K. Hiraishi. On Solvability of a Decentralized Supervisory Control Problem with Communication. *IEEE Trans. Autom. Control*, 54(3), 468-480, 2009.
14. C.A.R. Hoare. Communicating Sequential Processes. *CACM* 21(8): 666-677, 1978.
15. G. Kalyon, T. Le Gall, H. Marchand and T. Massart. Synthesis of Communicating Controllers for Distributed Systems. *Private communication*, 2011.
16. H. Lamouchi and J. Thistle. Effective Control Synthesis for DES under Partial Observations. In *CDC 2000*, pages 22-28.
17. L. Lamport. Arbiter-Free Synchronization. *Distrib. Comput.* 16(2/3): 219-237, 2003.
18. F. Lin and W.M. Wonham. On observability of discrete-event systems. *Info. Sci.* 44:173-198, 1988.
19. A. Mannani and P. Gohari. Decentralized Supervisory Control of Discrete-Event Systems Over Comunication Networks. *IEEE Trans. Autom. Control*, 53(2):547-559, 2008.
20. M. Mukund and M. Sohoni. Gossiping, Asynchronous Automata and Zielonka's Theorem. Report TCS-94-2, Chennai Mathematical Institute, 1994.
21. P. Madhusudan, P.S. Thiagarajan, S. Yang. The MSO Theory of Connectedly Communicating Processes. In *FSTTCS 2005*, LNCS 3821, pages 201-212.
22. D. Potop-Butucaru and B. Caillaud. Correct-by-Construction Asynchronous Implementation of Modular Synchronous Specifications. In *ACSD 2005*, pages 48-57.
23. P.J. Ramadge and W.M. Wonham. Supervisory Control of a Class of Discrete Event Processes. *SIAM J. Control Optim.*, 25:206-230, 1987.
24. P.J. Ramadge and W.M. Wonham. The Control of Discrete Event Systems. *Proc. of the IEEE, Special issue on Dynamics of Discrete Event Systems*, 77:81-98, 1989.
25. S.L. Ricker and B. Caillaud. Mind the Gap: Expanding Communication Options in Decentralized Discrete-Event Control. In *CDC 2007*, pages 5924-5929.
26. S.L. Ricker. Asymptotic Minimal Communication for Decentralized Discrete-Event Control. In *WODES 2008*, pages 486-491.
27. K. Rudie, S. Lafortune and F. Lin. Minimal Communication in a Distributed Discrete-Event System. *IEEE Trans. Autom. Control*, 48(6):957-975, 2003.
28. K. Rudie and W.M. Wonham. Think Globally, Act Locally: Decentralized Supervisory Control. *IEEE Trans. Autom. Control*, 37(11):1692-1708, 1992.
29. B. Caillaud. http://www.irisa.fr/s4/tools/synet/
30. J.G. Thistle. Undecidability in Decentralized Supervision. *Syst. Control Lett.*, 54: 503-509, 2005.
31. S. Tripakis. Decentralized Control of Discrete Event Systems with Bounded or Unbounded Delay Communication. *IEEE Trans. Autom. Control*, 49(9):1489-1501, 2004.
32. W. Wang, S. Lafortune and F. Lin. Minimization of Communication of Event Occurrences in Acyclic Discrete Event Systems. *IEEE Trans. Autom. Control*, 53(9):2197-2202, 2008.
33. Y. Wang, S. Lafortune, T. Kelly, M. Kudlur and S. Mahlke. The Theory of Deadlock Avoidance via Discrete Control. In *POPL 2009*, pages 252-263.

34. X. Xu and R. Kumar. Distributed State Estimation in Discrete Event Systems. In *ACC 2009*, pages 4735-4740.
35. T.S. Yoo and S. Lafortune. A General Architecture for Decentralized Supervisory Control of Discrete-event Systems. *Discrete Event Dyn. Syst.*, 12(3):335-377, 2002.
36. K. Yamalidou, J. Moody, M. Lemmon and P. Antsaklis. Feedback Control on Petri Nets Based on Place Invariants. *Automatica* 32(1): 15-28, 1996.
37. W. Zielonka. Notes on Finite Asynchronous Automata. *RAIRO Informatique Théorique et Applications*, 21:99-135, 1987.

Appendix

Our Asynchronous Message Passing Automata (AMPA) differ from the communicating automata originally introduced by Brand and Zafiropulo [4] in that communications are not FIFO. Given a finite set of locations $\mathcal{L}$, a *B-bounded AMPA* is a collection of DFA $\{A_\ell \mid \ell \in \mathcal{L}\}$ together with a finite set of messages P, where $\lambda : P \to \mathcal{L}$ specifies the address for each message. Each $A_\ell = (Q_\ell, \Sigma_\ell \uplus \Sigma^!_\ell \uplus \Sigma^?_\ell, \delta_\ell, q_{0,\ell})$ is a DFA with a partial transition map δ_ℓ, and initial state $q_{0,\ell}$, where $\Sigma^!_\ell = \{\ell!p \mid p \in P \wedge \lambda(p) \neq \ell\}$ and $\Sigma^?_\ell = \{\ell?p \mid p \in P \wedge \lambda(p) = \ell\}$. The actions in Σ_ℓ are observable. The communication actions in $\Sigma^!_\ell \cup \Sigma^?_\ell$ are unobservable.

The dynamics of a B-bounded AMPA are defined by a transition system $RG(AMPA)$ constructed inductively from an initial configuration $\langle \overline{q_0}, \overline{m_0} \rangle$ as follows:

- $\overline{q_0}$ is an $\mathcal{L}$-indexed vector with entries $q_{0,\ell}$ for all $\ell \in \mathcal{L}$,
- $\overline{m_0}$ is an P-indexed vector with null entries for all $p \in P$,
- From any configuration $\langle \overline{q}, \overline{m} \rangle$, where $\overline{q}$ is an $\mathcal{L}$-indexed vector with entries $q_\ell \in Q_\ell$, for all $\ell \in \mathcal{L}$, and $\overline{m}$ is a P-indexed vector of integers with entries $m_p \geq 0$ for all $p \in P$, there is a transition $\langle \overline{q}, \overline{m} \rangle \xrightarrow{\sigma} \langle \overline{q'}, \overline{m'} \rangle$ in the following three cases:
  - $q'_\ell = \delta_\ell(q_\ell, \sigma)$ for some $\ell \in \mathcal{L}$ and $\sigma \in \Sigma_\ell$, $q'_k = q_k$ for all $k \neq \ell$ and $\overline{m'} = \overline{m}$;
  - $q'_\ell = \delta_\ell(q_\ell, \sigma)$ for some $\ell \in \mathcal{L}$ and $\sigma = \ell!p \in \Sigma^!_\ell$, $q'_k = q_k$ for all $k \neq \ell$, $m'_p = m_p + 1 \leq B$, and $m'_r = m_r$ for all $r \neq p$;
  - $q'_\ell = \delta_\ell(q_\ell, \sigma)$ for some $\ell \in \mathcal{L}$ and $\sigma = \ell?p \in \Sigma^?_\ell$, $q'_k = q_k$ for all $k \neq \ell$, $m'_p = m_p - 1 \geq 0$, and $m'_r = m_r$ for all $r \neq p$;

Branching bisimulation was defined by van Glabbeek and Weijland [11] for processes with a single unobservable action τ. The following is an adaptation of the original definition to processes defined by automata with several unobservable actions. Let $\Sigma = \Sigma_o \cup \Sigma_{uo}$ be a set of labels, where Σ_o and Σ_{uo} are the subsets of observable and unobservable labels, respectively. Let $A = (Q, \Sigma, \delta, q_0)$ and $A' = (Q', \Sigma, \delta, q'_0)$ be two automata over Σ. A and A' are *branching bisimilar* if there exists a symmetric relation $R \in Q \times Q' \cup Q' \times Q$ such that $(q_0, q'_0) \in R$ and whenever $(r, s) \in R$

- if $\delta(r, \sigma) = r'$ and $\sigma \in \Sigma_{uo}$, then $(r', s) \in R$;
- if $\delta(r, \sigma) = r'$ and $\sigma \in \Sigma_o$, then there exists a sequence $\sigma'_1 \ldots \sigma'_k \in \Sigma^*_{uo}$ (where $k = 0$ means an empty sequence) such that if one lets $\delta(s, \sigma'_1 \ldots \sigma'_j) = s'_j$ for $j \leq k$, and $\delta(s'_k, \sigma) = s'$, then s' and states s'_j are effectively defined and $(r', s') \in R$.