

Formal Modeling and Supervisory Control of Reconfigurable Robot Teams

Kiriakos Kiriakidis¹ and Diana Gordon-Spears²

¹ Department of Weapons and Systems Engineering
United States Naval Academy
105 Maryland Avenue
Annapolis, MD 21402
kiriakid@novell.nadn.navy.mil

² Computer Science Department
College of Engineering
University of Wyoming
Laramie, WY 82071
dspears@cs.uwo.edu

In *Proceedings of the FAABS'02 Workshop*

Abstract. Teams of land-based, airborne, or submerged robots constitute a new breed of robotic systems for which the issue of controlled behavior arises naturally. In this paper, we model the dynamics of a reconfigurable (adaptable) robot team within the formalism of the discrete event system framework from control theory. The adaptation to be handled is one or more robots switching offline. This paper presents a novel method for learning and verification following adaptation – to restore supervision and assure the behavior of the team.

1 Introduction

Adaptability is key to survival in dynamic, real-world situations. Successful task achievement must be accomplished by graceful recovery from unexpected circumstances, e.g., by “steering” a system back on course to achieve its mission despite failures. The contribution of this paper is in presenting a novel, practical method for multi-robot systems to adapt “safely,” i.e., to remain within the bounds of specified properties that ensure successful task completion. We assume that a Finite State Automaton (FSA) models the behavior of an individual robot. To capture the collective behavior of a robot team whose composition changes, we construct an *event-varying* Discrete Event System (DES) as the team’s model. To control the natural behavior of such DES, we follow the approach of supervisory control theory [10,13,17]. Adaptation is achieved with machine learning, behavioral assurance is tested with formal verification, in particular, automata-theoretic model checking [11], and recovery is ensured via a novel pair of algorithms. The primary focus of this paper is on formalizing and implementing an approach to successful recovery. In particular, we explore how a reconfigurable robot team can recover effective supervisory control to ensure

task achievement when faced with unexpected robotic failures or re-grouping, i.e., a shift in the team composition. This paper presents a substantial extension of the earlier research of Kiriakidis and Gordon reported at FAABS'00 [9].

Note that the method of supervisory control of DES addresses a number of related problems, albeit on the typical assumption that the system is fixed and known [3, 15, 16]. At present, the literature offers only a few works on adaptive or robust supervisory control to tackle the problem of uncertainty in the DES model [2, 12, 18]. Furthermore, although this prior work advances the development of a theory for uncertain DES, the literature lacks any design methods for emerging engineering applications. Here, we present a practical engineering method that has been tested on a team of robots.

There are numerous important potential applications of our approach. One example is teams of planetary rovers that adapt to unanticipated planetary conditions while remaining within critical mission parameters. Another example is automated factory robot teams that adapt to equipment failures but continue operation within essential tolerances and other specifications. As a third example, supportive groups of automated military equipment transport vehicles would require both flexibility and behavioral assurance. Finally, consider the application of groups of robots for hazardous waste cleanup. This is yet another example illustrating the need for both responsiveness to failures and behavioral predictability.

2 Problem Formulation

In this paper, we consider robots that, in addition to equipment (e.g., sensors) necessary to carry out tasks, comprise a receiver and transmitter device as a means of communication with a coordinator (supervisor). We shall refer to a collection of robots that are collaborating with a common coordinator as a *robotic group*. The coordinator shapes the group's collective behavior by imposing cooperation between individual robots. In turn, cooperation implies that the robotic group possesses a set of desired *properties*, characteristic of its functionality. For example, a desired property of a delivery system whose function is to transport an object from point A to point Z is that delivery at Z eventually follows delivery at A.

In general, the robotic group may be operating on its own or as part of a larger system of peer groups. To model a particular group, we recognize that its behavior is associated with two classes of *events*. First, there are events that describe, in a well-defined sense, the tasks that each robot executes. The coordinator can manipulate those events to ensure multi-robot coordination. Second, there are events unpreventable by the coordinator such as when individual units (robots) switch coordinators or fail during operation. To update the state of the robotic group, the supervisor needs to know the event that a robot has just executed in order to identify its current task or determine that it has gone offline. Moreover, the structure of the event-driven model needs to be able to capture

the new situation when the coordinator loses (or gains) a robot. To provide for this, we shall assume a model that is also event-varying.

In this paper, we are going to focus our attention on the behavior of a robotic group and address the problem of maintaining the desired properties of the group in spite of unavoidable events. We shall employ the theory of automata, express each desired property as a sequence of events, and embed the resulting sequences in a desired language. To guarantee the desired language, we synthesize the group's coordinator, in the context of the supervisory control method [1, 14], so that it adapts to changes that have occurred in the group.

3 Formally Modeling a Reconfigurable Robot Team as an Event-Varying DES

Each robot has autonomy to complete a task (e.g., to transport an object from point A to point B). A robot is in a certain *state*, s , while a task is occupying it or it is waiting to start a new task. An *event*, σ , occurs when a robot starts a new task; thus, events coincide with state transitions. Before it starts a task and after it has completed one, the robot normally notifies the supervisor. Upon the completion of its current task, a robot is ready to execute another event.

The supervisor/coordinator decides which events are enabled and accordingly notifies the robots. Clearly, these events constitute an observable and controllable model. There are also uncontrollable events. A robot may be unable to start or finish a task. For example, a unit may have broken down or started to collaborate with another supervisor. These uncontrollable events are not part of the model per se, but their occurrence necessitates changes in the model. These events, however, are observable. If a robot starts to cooperate with another group, it will notify its old supervisor. For a robot that has broken down but is unable to notify its supervisor, the supervisor will assume that the robot is offline after a certain amount of time has elapsed.

Let us formalize the modeling of a robot group using the DES framework. The basic DES framework was developed in Ramadge and Wonham [14] and is further explained in [10]. We extend the basic framework here to handle uncontrollable events where robots may go offline. Each unit (robot), i , has its own alphabet, Σ_i , which comprises controllable events (its actions) only, and each admits an FSA model, G_i , $i \in \mathcal{I}_M$, where \mathcal{I}_M is an index set and M the number of units in the group initially. We also define the group alphabet, $\Sigma = \bigcup \Sigma_i$. Formally, an FSA is a quintuple $G = (X, \Sigma, \delta, s_0, X_m)$ where X is the set of states, δ the transition function, s_0 the initial state, X_m the set of marked (task completion) states. Strings in the the language of G , $L(G)$, and strings in the marked language of G , $L_m(G)$ (i.e., strings terminating in a marked state), are sequences of events. Marked states signify task completion. To take into account that robots may go offline, we define the FSA A , that is based on the following alphabet of all the uncontrollable events:

$$\Sigma_\lambda = \{\lambda_{i_k} \mid i_k \in \mathcal{I}_M, \text{ where } k \in \mathcal{I}_N, i_k \neq i_{k'} \text{ for } k \neq k', k' \in \mathcal{I}_N\}$$

where $N \leq M$ is the maximum number of units that may go offline. The occurrence of the event λ_{i_k} coincides with the loss of the robot number i_k . In accordance with the FSA A , each uncontrollable event occurs only once. After the event λ_{i_k} the robotic group comprises only $M - k$ robots.

Before any uncontrollable event occurs, the natural language of the robotic group stems from the following parallel composition:¹

$$G_\epsilon := G_1 \parallel G_2 \parallel \dots \parallel G_M$$

where ϵ is the empty string. When an uncontrollable event occurs, the parallel composition changes. After the uncontrollable event λ_{i_1} , $i_1 \in \mathcal{I}_M$, occurs, the model becomes

$$G_{\lambda_{i_1}} := G_1 \parallel \dots \parallel G_{i_1-1} \parallel G_{i_1+1} \parallel \dots \parallel G_M$$

Upon the occurrence of the uncontrollable events $\lambda_{i_2}, \dots, \lambda_{i_k}$, where $k \leq N$, the respective models are as follows:

$$G_{\lambda_{i_1} \lambda_{i_2}} = (G_{\lambda_{i_1}})_{\lambda_{i_2}}, \dots, G_{\lambda_{i_1} \dots \lambda_{i_k}} = (G_{\lambda_{i_1} \dots \lambda_{i_{k-1}}})_{\lambda_{i_k}} \quad (1)$$

Let r be a string of events in the language generated by A , i.e., $r \in L(A)$ identifies a sequence of units that goes offline. Then, the set of all possible models $\mathcal{G} = \{G_r \mid r \in L(A)\}$. Consider the FSA-valued function $G : (\Sigma \cup \Sigma_\lambda)^* \rightarrow \mathcal{G}$, where for the empty string, ϵ , the model of robotic group $G(\epsilon) = G_\epsilon$. Suppose a string of events $q_k \in (\Sigma \cup \Sigma_\lambda)^*$ includes the string of uncontrollable events $r_k = \lambda_{i_1} \dots \lambda_{i_k}$. Then, the occurrence of an event σ affects the group's model as follows:

$$G(q_k \sigma) = \begin{cases} G(q_k), & \sigma \notin \Sigma_\lambda \\ G_{r_k \sigma}, & \sigma \in \Sigma_\lambda \end{cases}$$

for $k = 1, \dots, N-1$. This event-varying DES captures the changes in the behavior of the robotic group as individual robots go offline.

4 Adaptive supervisory control

As mentioned earlier, the coordinator of a robotic group restricts the group's natural behavior in order to ensure that the group possesses certain desired properties. In the context of supervisory control, the coordinator accomplishes this by specifying and executing the desired language. In Section 2, we introduced the notion of the desired language as one that guarantees a set of desired properties. Let P_1, \dots, P_L and D denote the FSAs of desired properties and the current desired language, respectively. The current desired language, $L(D)$, satisfies the set of desired properties, $\mathcal{P} = \{P_1, \dots, P_L\}$, iff $L(D) \subseteq L(P_i)$, for each $i \in \mathcal{I}_L$ or, for brevity, $L(D) \subseteq L(\mathcal{P})$. The languages $L(D)$ and $L(\mathcal{P})$ consist of strings with events in the alphabets $\Sigma^D \subseteq \Sigma$ and $\Sigma^{\mathcal{P}} \subset \Sigma$, respectively.

¹ Because the alphabet Σ_i may be different for each robot i , this parallel composition is a shuffle product [14].

Before any uncontrollable event occurs, $L(D_\epsilon)$ is the set of all allowable action sequences for the robotic group G_ϵ . In general, the initial desired language FSA, D_ϵ , is constructed by an expert engineer. First, the desired language must be such that $L(D_\epsilon) \subseteq L(G_\epsilon)$. Second, it ought to satisfy each desired property, P_i , $i \in \mathcal{I}_L$. For example, P_i might state that an object that has been picked up must eventually be delivered. Third, the expert engineer may require that the initial desired language meet performance criteria more specific but less essential than the desired properties. Without supervision the robotic system may be unable to satisfy the desired properties, for, in general, $L(\mathcal{P}) \subseteq L(G_\epsilon)$.

Based on the event-varying DES that models the robotic group, we propose a learning algorithm to modify the desired language specified to the supervisor after each occurrence of an uncontrollable event. Upon execution of the uncontrollable event λ_{i_1} , the learning algorithm removes the events that pertain to the i_1 -th robot from the desired language FSA, D_ϵ . In turn, the algorithm repairs the resulting (possibly fractured or disconnected) FSA and yields a candidate desired language. A verification algorithm checks the candidate in order to ensure that the desired properties, \mathcal{P} , are still valid. For verification, we use the method of Automata-Theoretic (AT) model checking [11]. In brief, AT model checking assumes D and P are expressed as FSAs, and it checks whether $L(D) \subseteq L(P)$.

If the verification is successful, then the candidate is indeed a desired language, whose FSA we denote as $D_{\lambda_{i_1}}$. Clearly, it holds that $L(D_{\lambda_{i_1}}) \subseteq L(G_{\lambda_{i_1}})$. If the verification fails, we resort to an alternate learning algorithm that results in a smaller language, which, however, guarantees the desired properties. The procedure above repeats itself during a string of uncontrollable events r and, thus, generates a succession of desired language FSAs $\mathcal{D} = \{D_r \mid r \in L(A), \emptyset \neq L(D_r) \subseteq L(\mathcal{P}) \text{ and } L(D_r) \subseteq L(G_r)\}$. The supervisor synthesis step derives from the following result.

Proposition 1. Consider the event-varying DES (1) presented in Section 3, and the succession of desired languages \mathcal{D} . Suppose that the supervisor for (1) is the FSA-valued function $S : (\Sigma \cup \Sigma_\lambda)^* \rightarrow \mathcal{D}$ where $S(\epsilon) = D_\epsilon$ and

$$S(q_k \sigma) = \begin{cases} S(q_k), & \sigma \notin \Sigma_\lambda \\ D_{r_k \sigma}, & \sigma \in \Sigma_\lambda \end{cases} \quad (2)$$

Then for $k = 1, \dots, N - 1$, the supervised language $L[G(q_k \sigma) \| S(q_k \sigma)] \subseteq L(\mathcal{P})$.

Proof: Consider the desired language $L(D_r)$ for each $r \in L(A)$. First, from the hypothesis, $L(D_r)$ is non-empty and $L(D_r) \subseteq L(G_r)$. Second, $L(D_r)$ is prefix closed, i.e., $pr[L(D_r)] = L(D_r)$, for it is a generated language [1]. Third, $L(D_r)$ is controllable with respect to $L(G_r)$, for there are no uncontrollable events in $L(G_r)$. Then, for $\sigma \in \Sigma_\lambda$, the supervisor in (2) yields $L[G(q_k \sigma) \| S(q_k \sigma)] = L(D_{r_k \sigma})$; see [10]. Similarly, for $\sigma \notin \Sigma_\lambda$, $L[G(q_k \sigma) \| S(q_k \sigma)] = L(D_{r_k})$. ■

5 Adaptive supervisory control

This section addresses the situation wherein the robots must adapt. As mentioned earlier, the coordinator of a robotic group restricts the group's natural

behavior in order to ensure that the group possesses certain desired properties. In the context of supervisory control, the coordinator accomplishes this by specifying and executing a desired language, i.e., one that guarantees a set of desired properties. Let P and D denote the FSAs of a desired property and language, respectively. Then, a desired language, $L(D)$, satisfies a set of desired properties, $\mathcal{P} = \{P_1, \dots, P_L\}$, iff $L(D) \subseteq L(P_i)$, for each $i \in \mathcal{I}_L$. This is ensured initially by formal verification.

Based on the event-varying DES that models the robotic group, we propose a learning algorithm to modify the desired language (which is used to generate the supervisor) after each occurrence of an uncontrollable event. At first, the desired language satisfies: $L(D_\epsilon) \subseteq L(G_\epsilon)$. Upon execution of the uncontrollable event λ_{i_1} , the learning algorithm removes the events that pertain to the i_1 -th robot from the desired language FSA, D_ϵ . In turn, an algorithm called Maxmend repairs the resulting (probably fractured) FSA to yield a candidate desired language that is a subset of $L(G_{\lambda_{i_1}})$. A verification algorithm (model checking) checks the candidate in order to verify that the desired properties, \mathcal{P} , are still valid. If the verification is successful, then the candidate is indeed a desired language, whose FSA we denote as $D_{\lambda_{i_1}}$. If the verification fails, we resort to an alternate learning algorithm, called Quickmend, that results in a smaller language, which, however, guarantees that the desired properties hold, without the need for verification again [4]. The procedure above repeats itself during a string of uncontrollable events r and, thus, generates a succession of desired languages $\mathcal{D} = \{D_r \mid r \in L(A), \emptyset \neq L(D_r) \subseteq L(\mathcal{P}) \text{ and } L(D_r) \subseteq L(G_r)\}$.

Figures 1 and 2 show our algorithm Maxmend for repairing the FSA D following learning. Our motivation in designing this algorithm was a desire to preserve as much of the original FSA as possible, including preserving the order of transitions. In these figures, $\Psi_{pre}^D(s)$ is the set of all pre-learning successor states of state s in FSA D , $\Psi_{post}^D(s)$ is the set of all post-learning successors of s , $\delta_{pre}^D(s, \sigma)$ is the particular pre-learning successor of state s for event σ , and $\delta_{post}^D(s, \sigma)$ is the post-learning successor of s for event σ . Also, Σ_{pre}^D is the set of all pre-learning events, Σ_{post}^D is the set of all post-learning events, and Σ_{Δ}^D is the set of events deleted by learning. In other words, $\Sigma_{post}^D = \Sigma_{pre}^D \setminus \Sigma_{\Delta}^D$. Finally, X^D is the set of all states in the FSA D .

Let us now proceed to describe procedure Maxmend in words. Prior to calling procedure “repair-method” (Figure 1), variable “visited” is initialized to false for every state in the FSA. Procedure repair-method is then called with the initial FSA state as parameter s . Procedure repair-method does a depth-first search through the set of all states that are accessible from the initial state after learning. For each state visited on the depth-first search, “visited” is set to true so that it is not re-visited. Each state s that is visited which has no post-learning successors is considered an “unlinked-vertex.” In this case, procedure “find-linked-vertex” (Figure 2) is called to find the first pre-learning descendant of the unlinked-vertex that has a post-learning successor. This descendant is considered to be the “linked-vertex.” Procedure “copy-connections” (Figure 2) sets the successors of the unlinked-vertex equal to the successors of the linked-

```

procedure repair-method (s)
visited(s) = true;
if (( $\Psi_{post}^D(s) == \emptyset$ ) and
( $\Psi_{pre}^D(s) \neq \emptyset$ )) then {
  unlinked-vertex = s;
  linked-vertex = 0;
  for each  $\tau \in \Sigma_{\Delta}^D$  do {
    if (( $\delta_{pre}^D(s, \tau) \neq 0$ ) and
( $\delta_{pre}^D(s, \tau) \neq s$ )) then {
      for each  $s' \in X^D$  do
        visited2(s') = 0;
      od
      find-linked-vertex( $\delta_{pre}^D(s, \tau)$ );
      exit for-loop; } }
    fi
  od
  if ((linked-vertex  $\neq$  0) and
(linked-vertex  $\neq$  unlinked-vertex)) then
    copy-connections(unlinked-vertex, linked-vertex); }
  fi
fi
for each  $\sigma \in \Sigma_{post}^D$  do
  if ((visited( $\delta_{post}^D(s, \sigma)$ ) == false) and
( $\delta_{post}^D(s, \sigma) \neq 0$ )) then
    repair-method( $\delta_{post}^D(s, \sigma)$ );
  fi
od
end

```

Fig. 1. The algorithm Maxmend for FSA repair with verification.

```

procedure find-linked-vertex (s)
visited2(s) = true;
for each  $\sigma \in \Sigma_{pre}^D$  do
  if ( $\delta_{post}^D(s, \sigma) \neq 0$ ) then {
    linked-vertex = s;
    exit-for-loop; }
  else if (( $\delta_{pre}^D(s, \sigma) \neq 0$ ) and
( $visited2(\delta_{pre}^D(s, \sigma)) == false$ )) then
    find-linked-vertex( $\delta_{pre}^D(s, \sigma)$ );
  fi
od
end

procedure copy-connections (s1, s2)
for each  $\sigma \in \Sigma_{post}^D$  do {
   $\delta_{post}^D(s1, \sigma) = \delta_{post}^D(s2, \sigma)$ ;
  if ( $\delta_{post}^D(s2, \sigma) == s2$ ) then
     $\delta_{post}^D(s1, \sigma) = s1$ ; }
fi
od
end

```

Fig. 2. The subroutines of the Maxmend algorithm for FSA repair.

vertex for all post-learning events. If time permits, following this repair method the FSA may be simplified by removing unused (i.e., inaccessible) states and grouping states into equivalence classes using a state minimization algorithm [8].

```

procedure repair-method ( $s$ )
visited( $s$ ) = true;
for each  $\sigma \in \Sigma_{pre}^D$  do
  if ( $(\delta_{pre}^D(s, \sigma) \neq 0)$  and ( $\text{visited}(\delta_{pre}^D(s, \sigma)) == \text{false}$ )) then
    repair-method( $\delta_{pre}^D(s, \sigma)$ );
  fi
od
if ( $\Psi_{post}^D(s) == \emptyset$ ) then
  for each  $s' \in X^D$  do
    for each  $\sigma \in \Sigma_{post}^D$  do
      if ( $\delta_{post}^D(s', \sigma) == s$ ) then
         $\delta_{post}^D(s', \sigma) = 0$ ;
      fi
    od
  od
fi
end

```

Fig. 3. The algorithm Quickmend for FSA repair without verification.

The disadvantage of this repair algorithm is that it requires re-verification to be sure the resulting FSA still satisfies the property after the repair has been done. The reason that re-verification is required is that the process of mending (repair) adds new FSA transitions, which results in new strings of events. There is no guarantee that these new strings will satisfy the desired properties. In the event that re-verification fails, i.e., it indicates that the property has been violated, we propose the following alternative.

Figure 3 shows the Quickmend algorithm for repairing the desired language FSA after learning occurs. Prior to calling the algorithm of Figure 3, “visited” is initialized to false for every state in the FSA. Procedure “repair-method” is then called with the initial FSA state as parameter s . Procedure repair-method does a depth-first search through the set of all pre-learning states that are accessible from the initial state, and “visited” is set to avoid re-visitation. When the algorithm has visited all states, it pops the recursion stack to re-visit the states beginning with the last (i.e., it then visits the states in reverse order). For each state s visited in reverse order, the algorithm tests whether s has any post-learning successors. If not, it deletes all post-learning pointers to that state. Optionally, the FSA may be further simplified, e.g., by removing unused states.

Time complexity analyses of these two novel algorithms Maxmend and Quickmend may be found in [5–7]. Once a satisfactory desired language is found, automatic generation of a new supervisor follows the method of [10, 14]. This results in effective recovery of supervisory control, as well as preservation of properties that ensure task achievement.

6 Application to a Delivery System

In the laboratory, we tested the adaptive supervisory control approach on a delivery system that comprises a group of three mobile robots (G_1 , G_2 , and G_3), as well as a coordinator (supervisor). The coordinator consists of a field computer, which stays near the group and executes the supervisor’s automaton, and a main computer that devises the automaton and uploads it to the field computer. The goal of the robotic group is to deliver a number of objects from location A to location D. A set of sensors allows each mobile robot to detect the object (e.g., a puck) and find the location to deliver it. The communication between the main and field computers, as well as the field computer and each robot, is based on infrared transceivers.

Each robot has autonomy to complete a task, but it needs the coordinator’s authorization before it starts a new task. The job of the coordinator’s field computer is to disable certain state transitions (events) and enable other events in order to constrain the group’s natural language so that it satisfies the following desired property:

P: If G_1 picks-up, then G_3 will eventually deliver.

Completion of the task originally involves coordination of all three of the robots. We then simulate the failure of robot G_2 . Despite this failure, the adaptive supervision method described in this paper guarantees that the above desired property is preserved, i.e., the task is successfully completed. In summary, using the method described in this paper, the robotic group adapts and also meets its objective.

7 Conclusions and Future Work

This paper describes a practical method for the modeling and control of teams of robots, despite dynamically changing team composition. The method consists of applying learning to adapt to the situation of units going offline, followed by algorithmic revisions and verification of the desired language FSA, which is used for supervisory control. The practicality of our method has been demonstrated on a team of actual robots under supervision.

Future work will extend the current approach to handle the case of new units dynamically joining a team (i.e., added units). In parallel, we will continue to validate our approach on a variety of real-world applications.

Acknowledgements

This work was supported by the Naval Academy Research Council and the Office of Naval Research Grants N0001499WR20020 and N0001499WR20010.

References

1. C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Boston, Massachusetts: Kluwer Academic Publishers, 2001.
2. Y.-L. Chen, S. Lafortune, and F. Lin, "How to reuse supervisors when discrete event systems evolve," in *Proceedings of the IEEE Conference on Decision and Control*, (San Diego, CA), pp. 1442–1448, 1997.
3. Y. Du and S. H. Wang, "Control of discrete-event systems with minimal switching," *International Journal of Control*, vol. 48, no. 3, pp. 981–991, 1988.
4. D. Gordon, "Asimovian adaptive agents," *Journal of Artificial Intelligence Research*, vol. 13, pp. 95–153, 2000.
5. K. Kiriakidis and D. Gordon, "Supervision of multiple-robot systems," in *Proceedings of the American Control Conference*, (Arlington, VA), June 2001, pp. 2117–2118.
6. D. Gordon and K. Kiriakidis, "Adaptive supervisory control of interconnected discrete event systems," in *Proceedings of the 2000 IEEE Conference on Control Applications*, (Anchorage, AK), Sept. 2000, pp. 935–940.
7. D. Gordon and K. Kiriakidis, "Design of adaptive supervisors for discrete event systems via learning," in *Proceedings of the ASME Dynamic Systems and Control Division, International Mechanical Engineering Congress and Exposition*, (Orlando, FL), Nov. 2000, pp. 365–370.
8. J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Menlo Park, California: Addison-Wesley, 1979.
9. K. Kiriakidis and D. Gordon, "Supervision of multiple-robot systems," abstract in *Lecture Notes in Artificial Intelligence, Volume 1871* (Proceedings of FAABS'00), Springer-Verlag, 2001.
10. R. Kumar and V. K. Garg, *Modeling and Control of Logical Discrete Event Systems*. Boston, Massachusetts: Kluwer Academic Publishers, 1995.
11. P. Kurshan, *Computer Aided Verification of Coordinating Processes*. Princeton, New Jersey: Princeton University Press, 1994.
12. F. Lin, "Robust and adaptive supervisory control of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 38, no. 12, pp. 1842–1852, 1993.
13. P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.
14. P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.
15. J. G. Thistle, "Supervisory control of discrete event systems," *Mathematical and Computer Modelling*, vol. 23, no. 11, pp. 25–53, 1996.
16. W. M. Wonham and P. J. Ramadge, "Modular supervisory control of discrete-event systems," *Mathematics of Control, Signals, and Systems*, vol. 1, no. 1, pp. 13–30, 1988.
17. W. M. Wonham and P. J. Ramadge, "On the supremal controllable sublanguage of a given language," *SIAM J. Control and Optimization*, vol. 25, no. 3, pp. 637–659, 1987.

18. S. Young and V. K. Garg, "Model uncertainty in discrete event systems," *SIAM J. Control and Optimization*, vol. 33, no. 1, pp. 208–226, 1995.