Proceedings of the 1999 IEEE
International Conference on Robotics & Automation
Detroit, Michigan • May 1999

Discrete Event Control with Active Events

Michael Heymann[*], Feng Lin[†] and George Meyer[‡]

Abstract

The traditional framework for discrete event control is extended to include the case of control with active events, in which both the user and the environment have evnets that they can trigger. A variety of liveness and safety specifications can be considered within this extended framework. A synthesis algorithm of minimally restrictive controllers is outlined.

1 Introduction

In the traditional paradigm of supervisory control theory for discrete event systems, all events are assumed to be triggered by the environment. To achieve the required behavior, the user has the ability to control the system only by disabling some of the events (which are called controllable events). The supervisory control problem is then to synthesize a supervisor which, through suitable disablement of controllable events, confines the system's behavior to within specified legal requirements (usually in a minimally-restrictive way) [1] - [7].

Although in the resultant mathematical theory of discrete event control, the precise nature of the events is not always crucial, the theory cannot always be adapted to alternate setups. In particular, when both the user and the environment can trigger events in the system, the nature of the control problem can change substantially, and the traditional supervisory control framework must be modified.

In the present paper we present an extended framework for supervisory control, in which certain events can be triggered by the user, certain events can be triggered by the environment, and both the user and the environment can disable a subset of the other's events. An example is provided to illustrate this situation. While control for safety specifications remains quite similar to the traditional setup, control for liveness is quite different since several versions of liveness can convincingly be defined. For example, it may be required that the user be able to complete tasks using only events that he/she can trigger and the environment cannot disable. This leads to a new framework for liveness control and to a suitable modification of safety control. The theory is developed and a synthesis algorithm is outlined.

2 Illustrative Example

Example 1 Consider the following simple manufacturing system. The system consists of five machines, five buffers and two conveyors, as shown in Figure 1.

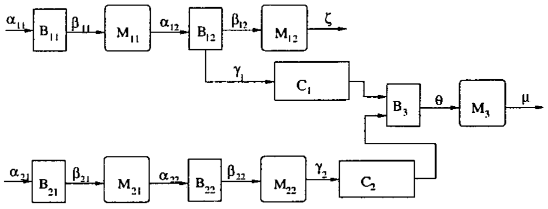

Figure 1: Manufacturing System

The system is controlled by two operators. The first operator O_1 has exclusive access to machines M_{11}, and M_{12}, buffers B_{11} and B_{12}, and conveyor C_1, and has shared access with operator O_2 to buffer B_3 and machine M_3. The second operator O_2 has exclusive ac-

[*]Department of Computer Science, Technion, Israel Institute of Technology, Haifa 32000, Israel
[†]Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202
[‡]NASA Ames Research Center, Moffett Field, CA 94035

cess to buffers B_{21} and B_{22}, machines M_{21} and M_{22}, and conveyor C_2. Operator O_2 has the ability to disable the operation of the conveyor C_1.

The system is operated in the following way: After entering a part into the buffer B_{11} (event α_{11}), operator O_1 can send it for processing to machine M_{11} (event β_{11}). Upon completion of processing in M_{11}, the part is deposited in buffer B_{12} (event α_{12}). Operator O_1 has then the choice either to send the part for further processing to machine M_{12} (event β_{12}) or, if not disabled, via the conveyor C_1 (event γ_1), to buffer B_3. If the part has been processed on machine M_{12}, then upon completion of processing, the part is ejected from the machine (event ζ). All the buffers have the capacity of exactly one part.

Operator O_2 can enter a part for processing into buffer B_{21} (event α_{21}), and subsequently take the part to be processed by M_{21} (event β_{21}). Thereafter the part is deposited in buffer B_{22} (event α_{22}), to be subsequently processed by M_{22} (event β_{22}). Thereafter, operator O_2 sends the part via conveyor C_2 (event γ_2) to buffer B_3. At any time, operator O_2 can feed a part from the buffer B_3 (provided it is not empty) to machine M_3 for processing (event θ) which, upon completion, is ejected from the machine (event μ). As stated, operator O_2 has the ability to disable γ_1 at any time or to enable it.

Note that the interpretation of controllable and uncontrollable events here is different from that in traditional supervisory control: From the viewpoint of operator O_1, an event is controllable if it can be executed (its occurrence enforced) by the controller (O_1 in this case) and cannot be disabled by the environment (O_2 in this case).

To study liveness properties of the manufacturing system, we designate the states where all parts are completely processed as the marked states. That is, we wish to guarantee that each part be processed successfully and completely.

Note that if we consider liveness to be the notion of nonblocking as in supervisory control, then the manufacturing system as described above is live: that is, from each state there exists a path to a marked state, and the system is nonblocking.

Upon a closer look, we find, however, that such liveness is at the mercy of the environment. In particular, there exist states from which, for the system to reach a marked state, depends on cooperation of the environment (i.e., on O_2). We shall call such reachability weak reachability.

To guarantee that a system is live even if its environment is not cooperative, we need to introduce a stronger notion of reachability: We require that, from any state, there exists a path to a marked state such that the path can be executed by the controller (that is, consists of only controllable events).

Based on this definition, the manufacturing system is not strongly reachable. Intuitively, to ensure strong reachability, O_1 must not be permitted to send the part to B_3.

With this illustrative example in mind, we shall develop the theory of active control for discrete event systems that deals with both weak and strong reachability, among other things.

3 Control With Active Events

In this section, we propose a new framework for modeling and control of discrete event systems in which both the user and the environment can trigger some of the events. We call such systems discrete event systems with active events. The system is modeled by an automaton

$$G = (\Sigma, Q, \delta, q_0, Q_m),$$

where the event set Σ, the state set Q, the transition function δ, and the initial state q_0 have their usual meaning as in discrete event control [7]. The system has two participants (players): the user and the environment[1]. As discussed earlier, events of the system are triggered either by the user or by the environment. Therefore, the event set Σ is partitioned as

$$\Sigma = \Sigma_{usr} \cup \Sigma_{env},$$

[1]The environment models everything other than the user, and may include other users.

where Σ_{usr} is the set of events that can be triggered by the user, while Σ_{env} is the set of events that can be triggered by the environment. In general the two event sets need not be disjoint, and there may be events that can be triggered both by the user and the environment. The event set Σ_{usr} consists of two disjoint subsets

$$\Sigma_{usr} = \Sigma^c_{usr} \dot{\cup} \Sigma^u_{usr},$$

where Σ^c_{usr} is the subset of the user-triggered events that can be disabled by the environment, and Σ^u_{usr} is the subset of the user-triggered events that cannot be disabled by the environment.

Similarly, the event set Σ_{env} is partitioned into two disjoint subsets

$$\Sigma_{env} = \Sigma^c_{env} \dot{\cup} \Sigma^u_{env}.$$

In traditional supervisory control of discrete event systems, it is assumed that all events are triggered by the environment (that is, $\Sigma_{usr} = \emptyset$) and that the user can only disable events (in Σ^c_{env}) from taking place. In this sense the control that the user can exert on the system is purely supervisory.

In the present setting the user has at his disposal in addition to the events from Σ^c_{env} that he can disable, also the events in Σ_{usr}, which we call *active* events, that he can trigger. As we shall see, several new issues must be addressed because of the introduction of active events.

We are still interested in supervisory control issues, in that we do not introduce any goal for our controller, other than to guarantee the safety and liveness of the system, from the point of view of the user, as to be further discussed in the next section. This goal must be achieved by a controller or supervisor through the disablement, or disallowing the execution of, certain events in G. Clearly, a controller cannot disable or disallow events in Σ^u_{env}. In other words, the controller can only disable or disallow the following "safety" events.

$$\Sigma_s = \Sigma - \Sigma^u_{env} = \Sigma^c_{env} \cup \Sigma_{usr} - \Sigma^u_{env}.$$

Formally, a controller is defined as a mapping γ

$$\gamma : L(G) \to 2^{\Sigma_s}$$

and operates as follows: After the occurrence of a sequence of events s, the controller "disables" the set of events $\gamma(s) \subseteq \Sigma_s$. Specifically, the subset of events $\gamma(s) \cap \Sigma^c_{env}$ is disabled from being triggered by the environment, and the subset of events $\gamma(s) \cap \Sigma_{usr}$ (which can be triggered by the user) is disallowed. On the other hand, events in $\Sigma - \gamma(s)$ can be triggered by the user or the environment if they are physically possible.

To specify the behavior of the controlled system γ/G, we study the language $L(\gamma/G)$ generated by the controlled system, which is given as follows.

- The empty string ϵ belongs to $L(\gamma/G)$
- After a sequence $s \in L(\gamma/G)$, $s\sigma \in L(\gamma/G)$ for $\sigma \in \Sigma$ if and only if σ is possible in $L(G)$ and is not disabled or disallowed by γ:

$$s\sigma \in L(\gamma/G) \Leftrightarrow s\sigma \in L(G) \wedge \sigma \notin \gamma(s).$$

Although our interpretation of the system G and controller γ are very different from those in traditional supervisory control, we have managed to keep their mathematical definition identical to those in traditional supervisory control, if we view Σ_s as the controllable events in traditional supervisory control. Therefore, the existence condition for our controller is characterized by the standard controllability condition [7].

Definition 1 A language $K \subseteq L(G)$ is said to be *controllable* with respect to Σ_s and $L(G)$ if

$$(\forall s \in \overline{K})(\forall \sigma \in \Sigma - \Sigma_s) s\sigma \in L(G) \Rightarrow s\sigma \in \overline{K}$$

where $\overline{K}$ is the prefix closure of K.

The following theorem, that characterizes conditions for the existence of a controller, is then the suitable restatement of a well known result (Proposition 5.1) of [6]:

Theorem 1 For a nonempty language $K \subseteq L(G)$, there exists a controller γ such that $L(\gamma/G) = K$ if and only if K is closed and controllable.

With this theorem, we can now discuss the specification and synthesis of controllers.

4 Safety and Liveness

As stated, the objective of our controller is to guarantee the safety and liveness of a system. Safety requirements specify what behaviors are not allowed in the system. One way of specifying safety is by stating a set $Q_b \subseteq Q$ of illegal states that the system must never enter. We call such a specification a *static* safety specification [2]. A more general safety requirement is a *dynamic* specification, given by a closed language $E = \overline{E}$ that describes the maximally allowed *safe* or *legal* behavior. (For obvious reasons, we assume that $E \subseteq L(G)$.) The safety requirement is then that the controlled system never exit E; that is

$$L(\gamma/G) \subseteq E.$$

It is not difficult to see that static safety specifications can always be stated as corresponding dynamic ones, and hence constitute a subset of the dynamic specifications.

On the other hand, the liveness requirements are stated as to guarantee that certain specified tasks can always be completed by the system. To this end, we specify a non-closed language M, called the *marked language*, that specifies the set of completed tasks[2]. Given a marked language M, the liveness requirement implies that every run of the controlled system must be extendable to a string of the marked language. In other words, liveness implies that every run can be extended to a completed task. Now in view of the classification of events allowed in the present framework, we can be somewhat more discriminating with respect to the question of what precisely we mean by the requirement of extendability to completed tasks. For example, in traditional supervisory control, liveness consists of the "nonblocking" condition, where task completion always implies the participation of the environment, since the user has no capability of triggering events. Thus, in the present setting, we may also be satisfied with a corresponding nonblocking condition that consists of the ability of the controlled system to complete tasks through the cooperative participation of the user and the environment. That is, every run can be completed to a string in M by concatenating to it suitable events from Σ. Alternatively, we may insist that the user be able to complete tasks by executing only the events that the user can trigger and the environment cannot disable; that is, events from Σ^u_{usr}. Another possibility is, that we may allow task completion by executing events from Σ_{usr}; that is, any user triggered events. Thus, we define a suitable set of *liveness events* Σ_l with respect to which task completion to strings in M must be possible. To state the liveness specification formally, we require that the language $L(\gamma/G)$ be *suffix completable* as defined below.

[2]This kind of specification is more general than specifying a set of marked states $Q_m \subseteq Q$, as is customary in traditional supervisory control.

Definition 2 A language $K \subseteq L(G)$ is said to be *suffix completable* with respect to Σ_l and M if

$$(\forall s \in \overline{K})(\exists t \in \Sigma_l^*) st \in \overline{K} \cap M.$$

In words, a language K is suffix completable, if every string in its prefix closure, can be completed to a string in the marked language M by concatenating to it some events from Σ_l. We define suffix completeness of K based on the prefix closure of K because $L(\gamma/G)$ is always closed. Clearly, with this definition, a language is suffix completable if and only if its prefix closure is suffix completable.

Note that, unlike in traditional supervisory control, where liveness or task completion is modeled by a set of marked states in G, we consider task completion as specified by M. This gives us more freedom in task specification, just like specifying E is more general than specifying a set of illegal states.

From the above discussion, it is now clear that in order to satisfy both safety and liveness requirements, the language generated by a controlled system, $L(\gamma/G)$, must (1) be contained in E and (2) be suffix completable with respect to Σ_l and M. However, there may exist many controllers that achieve this requirement. In view of the fact that our control objectives are supervisory in nature, in that we only want to guarantee safety and liveness (rather than some kind of preferred "optimal" performance), we would like, just as in traditional supervisory control, to find the minimally restrictive controller that permits the maximal possible legal behavior to survive. Clearly, such a minimally restrictive controller will generates a largest language K such that (1) K is closed and controllable (required by Theorem 1 for the existence of a controller); (2) K is contained in E (required by the safety specification); and (3) K is suffix completable (for liveness). Thus, in order to synthesize a minimally restrictive controller, let us proceed by defining the set of closed, controllable, and suffix completable sublanguage of E, as

$$\begin{aligned} CL(E) = \; & \{K \subseteq E : K \text{ is closed, controllable and} \\ & \text{suffix completable}\}. \end{aligned}$$

This set has a very nice property.

Theorem 2 $CL(E)$ is closed under (arbitrary) union.

By Theorem 2, we conclude that the supremal element of $CL(E)$ exists. We denote it by $E^{\uparrow}$. By Theorem 1, we know that a controller can always be synthesized such that $L(\gamma/G) = E^{\uparrow}$. Therefore, the problem of synthesizing the minimally restrictive controller that guarantees safety and liveness is reduced to the problem of finding $E^{\uparrow}$.

Before showing how to calculate $E^{\uparrow}$ in the next section, let us first remark about two related issues.

First, if we take $\Sigma_l = \Sigma$ and $M = L_m(G)$, then liveness is equivalent to the nonblocking requirement in traditional supervisory control, which is defined as

$$\overline{L_m(\gamma/G)} = L(\gamma/G).$$

To see this, note that since $L_m(\gamma/G) = L(\gamma/G) \cap L_m(G)$,

$$\begin{aligned} & \overline{L_m(\gamma/G)} = L(\gamma/G) \\ \Leftrightarrow \; & \overline{L(\gamma/G) \cap L_m(G)} = L(\gamma/G) \\ \Leftrightarrow \; & \overline{L(\gamma/G) \cap L_m(G)} \supseteq L(\gamma/G). \end{aligned}$$

On the other hand, liveness is equivalent to

$$\begin{aligned} & (\forall s \in L(\gamma/G))(\exists t \in \Sigma^*) st \in L(\gamma/G) \cap L_m(G) \\ \Leftrightarrow \; & L(\gamma/G) \subseteq \overline{L(\gamma/G) \cap L_m(G)}, \end{aligned}$$

which is same as nonblocking.

The second issue is about partial observation. If a controller can only observe events in some observable event set Σ_o, then a partial observation controller is defined as a map

$$\gamma : PL(G) \rightarrow 2^{\Sigma_s}$$

where $P : \Sigma^* \rightarrow \Sigma_o^*$ is the natural projection. It is known from [3] [4], that the condition for existence of a supervisor under partial observation is controllability and observability of the supervised language. Observability is defined as follows.

Definition 3 A language $K \subseteq L(G)$ is said to be *observable* with respect to Σ_o and $L(G)$ if

$$\begin{aligned} & (\forall s, s' \in \overline{K}) Ps = Ps' \\ & \Rightarrow (\forall \sigma \in \Sigma) s\sigma \in \overline{K} \wedge s'\sigma \in L(G) \Rightarrow s'\sigma \in \overline{K}. \end{aligned}$$

As in the case of full observation, the supervisor synthesis problem is reduced to finding a largest closed, controllable, observable, and suffix completable sublanguage of E. In other words, we are interested in the following set

$$\begin{aligned} COL(E) = \; & \{K \subseteq E : K \text{ is closed, controllable,} \\ & \text{observable and suffix completable}\}. \end{aligned}$$

Unfortunately, this set is not closed under union, as the union of two observable languages may not be observable. Hence the supremal element of $COL(E)$ may not exists and we may only find a maximal element of $COL(E)$. A controller can then be synthesized based on the maximal element. The algorithm to calculate a maximal element of $COL(E)$ is much more complicated and will not be discussed in this paper.

5 Controller Synthesis

In this section, we study the key to controller synthesis: How to find $E^{\uparrow}$. We first consider the case where specifications are static. As stated earlier, by a static specification, we mean that the safety and liveness requirements are given by two sets of states Q_b and Q_m. Here $Q_b \subseteq Q$ is the set of illegal states that the system must not visit. The corresponding legal language is then given by

$$E = \{s \in L(G) : (\forall t \leq s) \delta(q_o, t) \notin Q_b\},$$

where $t \leq s$ means t is a prefix of s. Similarly, $Q_m \subseteq Q$ is the set of marked states representing the completion of tasks. Hence the corresponding marked language is given by

$$M = \{s \in L(G) : \delta(q_o, s) \in Q_m\}.$$

To find $E^\uparrow$ in the static case, the algorithm proceeds along the following lines: At each step, there is a set BS of "bad states" (which initially is, of course, equal to Q_b). We "shrink" E, to render it controllable, by adding to the set BS the set of "uncontrollable states", from which there exists an uncontrollable path to BS. That is, we augment BS with

$$Q_{uc}(BS) = \{q \in Q - BS : (\exists s \in (\Sigma - \Sigma_s)^*)\delta(q, s) \in BS\}.$$

The states in this set are "uncontrollable" in the sense that when in any of these states, the system can execute a sequence of events that cannot be disabled or disallowed by the controller, that causes the system to enter a state in BS. Obviously, any string leading to $Q_{uc}(BS)$ cannot be in $E^\uparrow$, and hence the states in $Q_{uc}(BS)$ must be added to BS.

Next we shrink E further, to make it suffix completable, by adding the following "non-completable" states to BS:

$$Q_{nc}(BS) = \{q \in Q - BS : \neg(\exists s \in \Sigma_l^*)(\delta(q, s) \in Q_m \wedge (\forall t \leq s)\delta(q, t) \notin BS)\}.$$

These are states from which the system cannot reach a marked state through event sequences that consist only of events in Σ_l without intercepting illegal states in BS.

Clearly, the addition of the set $Q_{nc}(BS)$ to BS can generate new uncontrollable states from which a string in $(\Sigma - \Sigma_s)^*$ will lead to a state in BS. Therefore the above procedure of enlarging BS must be repeated until it converges (when no new states are added).

The formal algorithm is presented in the full version of the paper, available at www.ece.eng.wayne.edu/ ~flin.

How to handle dynamic specifications is also discussed in the full paper.

Acknowledgments

This research is supported in part by the National Science Foundation under grants ECS-9315344, INT-9602485 and NASA under grant NAG2-1043 and in part by the Technion Fund for Promotion of Research. The work by the first author was completed while he was visiting NASA Ames Research Center, Moffett Field, CA 94035, on a grant with San Jose State University.

References

[1] M. Heymann and F. Lin, 1994. On-line control of partially observed discrete event systems. *Discrete Event Dynamic Systems: Theory and Applications, 4(3)*, pp. 221-236.

[2] M. Heymann and F. Lin, 1998. Discrete event control of nondeterministic systems. control of nondeterministic systems, *IEEE Transactions on Automatic Control, 43(1)*, pp. 3-17.

[3] F. Lin. On controllability and observability of discrete event systems. *Ph. D. Thesis*, Department of Electrical Engineering, University of Toronto, 1987.

[4] F. Lin and W. M. Wonham, 1988. On observability of discrete event systems. *Information Sciences, 44(3)*, pp. 173-198.

[5] F. Lin and W. M. Wonham, 1990. Decentralized control and coordination of discrete event systems with partial observation. *IEEE Transactions on Automatic Control, 35(12)*, pp. 1330-1337.

[6] R. J. Ramadge and W. M. Wonham, 1987. Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization, 25(1)*, pp. 206-230.

[7] P. J. Ramadge and W. M. Wonham, 1989. The control of discrete event systems. *Proceedings of IEEE, 77(1)*, pp. 81-98.