Multiuser 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 events 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. Multiuser systems are also discussed.**

***Index Terms*—Discrete-event systems, manufacturing systems, multiagent, safety and liveness, supervisory control.**

I. 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]–[14].

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 note 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. We also apply this framework to multiuser systems.

II. Illustrative Example

Example 1: Consider the following simple manufacturing system. The system consists of two machines and a buffer, as shown in Fig. 1.

Manuscript received August 11, 2000; revised August 2, 2001. Recommended by Associate Editor R. S. Sreenivas. This work was supported in part by the National Science Foundation under Grants INT-9602485 and ITR-0082784, NASA under Grant NAG2-1043, and in part by the Technion Fund for Promotion of Research. The work of the first author was completed while he was visiting NASA Ames Research Center, Moffett Field, CA 94035 USA, on a grant with San Jose State University.

M. Heymann is with the Department of Computer Science, Technion—Israel Institute of Technology, Haifa 32000, Israel (e-mail: heymann@cs.technion.ac.il).

F. Lin is with the Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202 USA (e-mail: flin@ece.eng.wayne.edu).

G. Meyer is with the NASA Ames Research Center, Moffett Field, CA 94035 USA (e-mail: gmeyer@mail.arc.nasa.gov).

Publisher Item Identifier S 0018-9286(02)02072-X.

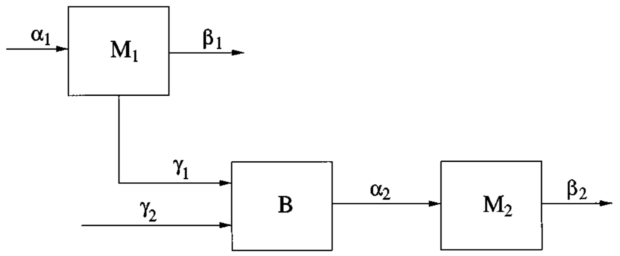

Fig. 1. Manufacturing System.

The system is controlled by two operators. The first operator O_1 has exclusive access to machine M_1 and has shared access with operator O_2 to the buffer B. Machine M_2 is fed from the buffer, and is operated by operator O_2. The buffer has the capacity of exactly one part.

The system is operated in the following way: Operator O_1 can send a part for processing to machine M_1 (event α_1). He/she can then choose to allow machine M_1 to complete processing the part, and then remove the finished part from the machine (event β_1) or, if the buffer is empty, he/she can remove the partially processed part from M_1 to the buffer B (event γ_1) for finishing on machine M_2.

Operator O_2 can feed a part directly into buffer B, provided the buffer is empty (event γ_2). At any time, he/she can feed a part from the buffer B (provided it is not empty) to machine M_2 for processing (event α_2) which, upon completion, is ejected from the machine (event β_2). Operator O_2 has the ability, at any time, to disable γ_1 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, it depends on cooperation of the environment (i.e., on O_2). We refer to such reachability as 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 that can be executed by the controller and cannot be disabled by the environment (that is, a path that consists only of 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 the buffer B.

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.

III. 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, the initial state q_0, and the marked state set Q_m have their usual meaning as in discrete-event control [14]. 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 expressed as

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

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

$$\Sigma_{\text{usr}} = \Sigma_{\text{usr}}^c \dot{\cup} \Sigma_{\text{usr}}^u$$

where Σ_{usr}^c is the subset of the user-triggered events that can be disabled by the environment and Σ_{usr}^u 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 $\Sigma_{\text{env}} = \Sigma_{\text{env}}^c \dot{\cup} \Sigma_{\text{env}}^u$.

In traditional supervisory control of discrete-event systems, it is assumed that all events are triggered by the environment (that is, $\Sigma_{\text{usr}} = \emptyset$) and that the user can only disable events (in Σ_{env}^c) 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/her disposal in addition to the events from Σ_{env}^c that he/she can disable, also the events in Σ_{usr}, which we call *active* events, that he/she 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 Section IV. This goal must be achieved by a controller or supervisor through the disablement, or disallowing the execution of, certain events in G. Clearly, the user has no authority over events in Σ_{env}^u, which he/she cannot prevent from occurring. In other words, the controller can only disable or disallow the following "safety" events.

$$\Sigma_s = \Sigma - \Sigma_{\text{env}}^u = \Sigma_{\text{env}}^c \cup \Sigma_{\text{usr}} - \Sigma_{\text{env}}^u.$$

Formally, a controller is defined as a mapping γ

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

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 $\gamma(s) \cap \Sigma_{\text{env}}^c$ is disabled from being triggered by the environment, and the subset of events $\gamma(s) \cap \Sigma_{\text{usr}}$ (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, that is

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

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 set of controllable events in traditional supervisory control. Therefore, the existence condition for our controller is characterized by the controllability condition [14], just as in traditional supervisory control.

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

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

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

The following theorem, that characterizes conditions for the existence of a controller, is then the suitable restatement of the well-known result of [13, Prop. 5.1].

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.

IV. 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 [6]. A more general safety requirement is a *dynamic* specification, given by a closed language $E = \bar{E}$ 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

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

On the other hand, the liveness requirements are stated so as to guarantee that certain specified tasks can always be completed by the system. To this end, we specify a nonclosed 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 a

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

[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.

string of 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_{usr}^u. 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.

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

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

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 as specifying a legal language 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 set of legal behaviors to survive. Clearly, such a minimally restrictive controller would generate the 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, it must first be shown that such a language actually exists. Let us proceed by defining the set of closed, controllable, and suffix completable sublanguage of E, as

$$\mathrm{CL}(E) = \{K \subseteq E : K \text{ is closed, controllable and suffix completable}\}.$$

This set has the following very nice property.

Theorem 2: $\mathrm{CL}(E)$ is closed under (arbitrary) union.

Proof: Let R be the union of an arbitrary set of languages in $\mathrm{CL}(E)$. The fact that R is closed and controllable is known from [13]. So we only need to show that R is suffix completable. To this end, let $s \in R$ be any string. Then $s \in K$ for some language $K \subseteq R$, and since K is suffix completable,

$$(\exists t \in \Sigma_l^*) st \in \bar{K} \cap M \Rightarrow (\exists t \in \Sigma_l^*) st \in \bar{R} \cap M.$$

Therefore, $(\forall s \in R)(\exists t \in \Sigma_l^*) st \in \bar{R} \cap M$ and, hence, R is suffix completable. ■

By Theorem 2, we conclude that the supremal element of $\mathrm{CL}(E)$ 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 characterized by the condition that

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

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

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

The last step is due to the fact that $\overline{L(\gamma/G) \cap L_m(G)} \subseteq \overline{L(\gamma/G)} \cap \overline{L_m(G)} = L(\gamma/G)$ is always true.

On the other hand, liveness is equivalent to the property

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

which is the 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 [8], [9], 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

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

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:

$$\mathrm{COL}(E) = \{K \subseteq E : K \text{is closed, controllable, observable and suffix completable}\}.$$

Unfortunately, this set is not generally closed under union, since the union of two observable languages may not be observable [8], [9]. Hence, the supremal element of $\mathrm{COL}(E)$ may not exist and we may only be able to find some maximal element of $\mathrm{COL}(E)$. A controller can then be synthesized based on this maximal element. An algorithm for finding a maximal element of $\mathrm{COL}(E)$ is much more complicated and will not be discussed in this note.

V. Controller Synthesis

In this section, we study the key to controller synthesis: How to find $E^{\uparrow}$. We consider two cases: (1) where the specification is static, and (2) the specification is dynamic.

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 of bad states (BS) (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

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

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 $Q_{\mathrm{uc}}(\mathrm{BS})$ cannot be in $E^\uparrow$, and hence the states in $Q_{\mathrm{uc}}(\mathrm{BS})$ must be added to BS.

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

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

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 $Q_{\mathrm{nc}}(\mathrm{BS})$ 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 precise algorithm is omitted here, but can be found in the full version of the note at www.ece.eng.wayne.edu/~flin.

The above algorithm shows how to calculate $E^\uparrow$ when the specification is static. In the general case, the specification may be dynamic and given by a legal language E and a marked language M. Our approach in this case is to translate the dynamic specification into a static specification by possibly enlarging the state set of G. The details can be found in the full version of the note.

VI. Multi-User Systems

In this section, we consider systems with n users. For each user, two sets of events are specified

Σ_i^d is the set of events that user i can disable;
Σ_i^t is the set of events that user i can trigger, $i = 1, 2, \ldots, n$.

We do not put any constraints on the sets Σ_i^d and Σ_i^t. Hence, we permit the possibility that an event can be triggered and/or disabled by more than one user.

If we consider the user and environment in the previous sections as user 1 and user 2, respectively, then in terms of Σ_i^d and $\Sigma_i^t, i = 1, 2$, we have

$$\Sigma_1^t = \Sigma_{\mathrm{usr}} \quad \Sigma_2^t = \Sigma_{\mathrm{env}} \quad \Sigma_1^d = \Sigma_{\mathrm{env}}^c \quad \Sigma_2^d = \Sigma_{\mathrm{usr}}^c.$$

Hence

$$\Sigma_{\mathrm{env}}^u = \Sigma_2^t - \Sigma_1^d \quad \Sigma_{\mathrm{usr}}^u = \Sigma_1^t - \Sigma_2^d.$$

What makes the multiuser problem different and interesting is the various modes of possible cooperation among users. For example, can a user rely on some other users for disablement of certain events that the user him/herself cannot disable? Or can a user rely on some other users to trigger certain events on his/her behalf that the user him/herself cannot trigger? The answers to such questions depend on the objective of the user and the degree of cooperation of the other users. Therefore, let us distinguish, from the viewpoint of user i, two sets of users

FA—the set of cooperative users;
EA—the set of noncooperative users.

As before, the objectives of the controller for user $i, i = 1, 2, \ldots, n$ is to achieve safety and liveness. For safety events Σ_{s_i} of user i, at least two cases can be discussed.

1) User i relies for his/her safety only on events that he/she can disable or disallow. That is, $\Sigma_{s_i} = \Sigma_i^d \cup (\Sigma_i^t - \cup_{j \neq i}\Sigma_j^t)$.
2) Or, more optimistically, user i relies for safety also on other (cooperative) users to disable or disallow events. $\Sigma_{s_i} = (\cup_{i \in \mathrm{FA}}\Sigma_i^d) \cup (\cup_{i \in \mathrm{FA}}\Sigma_i^t - \cup_{j \notin \mathrm{FA}}\Sigma_j^t)$.

On the other hand, the liveness events Σ_{l_i} can be specified differently, depending on the application at hand, as follows.

Case 1) At one extreme, user i wants to be absolutely sure and guarantee task completion without compromise. Therefore, he/she will rely only on the events that he/she can trigger but no other agents can disable: $\Sigma_{l_i} = \Sigma_i^t - (\cup_{j \neq i}\Sigma_j^d)$.
Case 2) In the second case, user i will rely on the events that he/she can trigger but no uncooperative agents can disable: $\Sigma_{l_i} = \Sigma_i^t - (\cup_{j \in \mathrm{EA}}\Sigma_j^d)$.
Case 3) In the third case, user i will rely on the events that the cooperative agents can trigger but no uncooperative agents can disable: $\Sigma_{l_i} = (\cup_{i \in \mathrm{FA}}\Sigma_i^t) - (\cup_{j \in \mathrm{EA}}\Sigma_j^d)$.
Case 4) Finally, at the other extreme, user i may rely on everyone's mercy to achieve liveness: $\Sigma_{l_i} = \Sigma$. (This last case is assumed in traditional supervisory control).

After determining Σ_{s_i} and Σ_{l_i}, we will then design a controller for agent user i that ensures safety and liveness in the same way as discussed in the previous sections.

Example 2: (revisited) We return now to the example presented in Section II and will show how controllers can be synthesized under various assumptions regarding the attitude of (or degree of cooperation between) its two users (operators).

The state transition diagram of the uncontrolled system is obtained by the synchronous composition of its three components as shown in Fig. 2, where the state labels XYZ mean: machine 1 is in state X ($X = \mathrm{Idle}$ or $X = \mathrm{Run}$), buffer is in state Y ($Y = \mathrm{Empty}, Y = \mathrm{Full}$ or $Y = \mathrm{Jammed}$) and machine 2 is in state Z. The dashed states (where the buffer is jammed) are illegal, and the legal subsystem (and language) can be obtained upon deletion from the system of these states and their associated transitions.

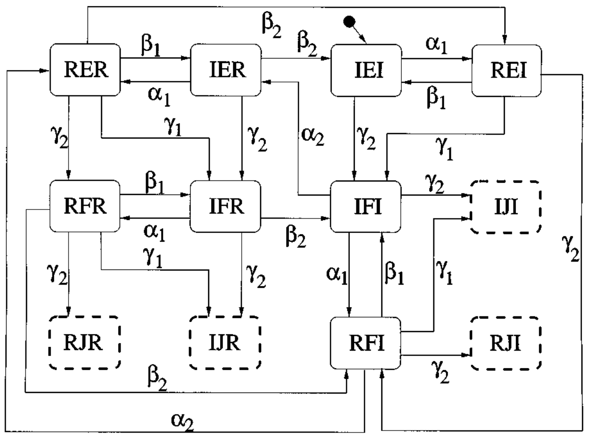

Fig. 2. Transition diagram of manufacturing system.

The sets of events that operators 1 and 2 can trigger are, respectively, $\Sigma_1^t = \{\alpha_1, \beta_1, \gamma_1\}$ and $\Sigma_2^t = \{\alpha_2, \beta_2, \gamma_2\}$. Similarly, the sets of events operators 1 and 2 can disable are, respectively, $\Sigma_1^d = \{\alpha_1, \beta_1, \gamma_1\}$ and $\Sigma_2^d = \{\alpha_2, \beta_2, \gamma_2, \gamma_1\}$.

If user 1 wants to insure safety (i.e., prevent the system from reaching any of the illegal states) and user 2 is not cooperative, then $\Sigma_{s_1} = \Sigma_1^d$. The legal language in this case is not controllable and the supremal controllable sublanguage is empty. However, if user 2 is cooperative, then $\Sigma_{s_1} = \Sigma$, and the legal language is controllable. In contrast, If user 2 wants to insure safety, then $\Sigma_{s_2} = \Sigma_2^d$ in case user 1 is uncooperative, and $\Sigma_{s_2} = \Sigma$ in case user 1 is cooperative. In both cases the legal language is controllable.

Let us now assume that the initial state IEI is the only marked state of the system, and that in addition to safety, the controller must satisfy the liveness condition specified by the marked language that consists of all the event strings that lead the system to this marked state.

Let us now consider the two-user control problem with the requirement that both safety and liveness must be satisfied. In case user 1 wants to achieve safety and liveness, only the situation where user 2 is cooperative with respect to safety is relevant. Let us further assume that user 2 is cooperative also with respect to liveness, in which case, $\mathrm{EA} = \emptyset$ and $\mathrm{FA} = \{1, 2\}$.

In case 1 (where $\Sigma_{l_i} = \Sigma_i^t - (\cup_{j \neq i} \Sigma_j^d)$), we obtain $\Sigma_{l_1} = \{\alpha_1, \beta_1\}$. By our synthesis algorithm, the resulting safe and live system consists of states IEI and REI.

In case 2 (where $\Sigma_{l_i} = \Sigma_i^t - (\cup_{j \in \mathrm{EA}} \Sigma_j^d)$), we obtain $\Sigma_{l_1} = \{\alpha_1, \beta_1, \gamma_1\}$. By our synthesis algorithm, the resulting safe and live system consists of states IEI and REI.

In case 3 (where $\Sigma_{l_i} = (\cup_{i \in \mathrm{FA}} \Sigma_i^t) - (\cup_{j \in \mathrm{EA}} \Sigma_j^d)$), we obtain $\Sigma_{l_1} = \Sigma$. By our synthesis algorithm, the resulting safe and live system consists of all the legal states.

In case 4 (where $\Sigma_{l_i} = \Sigma$), we obtain $\Sigma_{l_1} = \Sigma$. By our synthesis algorithm, the resulting safe and live system consists of all the legal states.

In a similar fashion, we can discuss how user 2 can achieve safety and liveness.

VII. Conclusion

We have introduced an extended framework for discrete-event control where, in addition to the events that can be triggered by the environment, the user has at his/her disposal a set of events that he/she can trigger. Both the user and the environment can each disable certain events of the other. We examined the control problem where both safety and liveness requirements can be specified in a somewhat more general setting than in the traditional discrete-event control framework. A particularly interesting generalization is obtained when the environment consists of (or includes) one or more additional users. This leads to a variety of interesting scenarios where the users have each their own control objectives (specifications) and capabilities.

References

[1] R. D. Brandt, V. Garg, R. Kumar, F. Lin, S. I. Marcus, and W. M. Wonham, "Formulas for calculating supremal controllable and normal sublanguages," *Syst. Control Lett.*, vol. 15, pp. 111–117, 1990.

[2] S. L. Chung, S. Lafortune, and F. Lin, "Limited lookahead policies in supervisory control of discrete-event systems," *IEEE Trans. Automat. Contr.*, vol. 37, pp. 1921–1935, Dec. 1992.

[3] M. Heymann, "Concurrency and discrete-event control," *IEEE Control Syst. Mag.*, vol. 10, no. 4, pp. 103–112, 1990.

[4] M. Heymann and G. Meyer, "An Algebra of Discrete Event Processes,", NASA Technical Memorandum 102 848, 1991.

[5] M. Heymann and F. Lin, "On-line control of partially observed discrete event systems," *Discrete Event Dyna. Syst.: Theory Appl.*, vol. 4, no. 3, pp. 221–236, 1994.

[6] ——, "Discrete event control of nondeterministic systems," *IEEE Trans. Automat. Contr.*, vol. 43, pp. 3–17, Jan. 1998.

[7] ——, "Nonblocking Supervisory Control of Nondeterministic Systems,", Technion, Israel, CIS Report 9620, 1996.

[8] F. Lin, "On Controllability and Observability of Discrete Event Systems," Ph. D. dissertation, Department of Electrical Engineering, Univ. of Toronto, Toronto, ON, Canada, 1987.

[9] F. Lin and W. M. Wonham, "On observability of discrete event systems," *Inform. Sci.*, vol. 44, no. 3, pp. 173–198, 1988.

[10] ——, "Decentralized supervisory control of discrete-event systems," *Inform. Sci.*, vol. 44, no. 3, pp. 199–224, 1988.

[11] ——, "Decentralized control and coordination of discrete event systems with partial observation," *IEEE Trans. Automat. Contr.*, vol. 35, pp. 1330–1337, Dec. 1990.

[12] ——, "Supervisory control of timed discrete event systems under partial observation," *IEEE Trans. Automat. Contr.*, vol. 40, pp. 558–562, Mar. 1994.

[13] R. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, 1987.

[14] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, pp. 81–98, Jan. 1989.

Extremum Seeking Control for Discrete-Time Systems

Joon-Young Choi, Miroslav Krstić, Kartik B. Ariyur, and Jin S. Lee

***Abstract*—We present an extremum seeking control algorithm for discrete-time systems applied to a class of plants that are represented as a series combination of a linear input dynamics, a static nonlinearity with an extremum, and a linear output dynamics. By using the two-time scale averaging theory, we derive a mild sufficient condition under which the plant output exponentially converges to an $O(\alpha^2)$ neighborhood of the extremum value, where α is the magnitude of modulation signal. The sufficient condition is related to positive realness of linear parts of the plant but only at the modulation frequency. The algorithm is illustrated with a brief simulation study.**

***Index Terms*—Averaging, discrete-time systems, extremum seeking.**

I. Introduction

Extremum seeking, a nonmodel based method of adaptive control, deals with systems where the reference-to-output map is uncertain but is known to have an extremum. The objective of extremum seeking is to find the set point that achieves the extremum.

Krstić and Wang [1] presented the first stability analysis for an extremum seeking system applied to a general nonlinear dynamical plant. Their analysis used averaging and singular perturbations. Krstić [2]

Manuscript received February 19, 2001; revised July 23, 2001. Recommended by Associate Editor L. Y. Wang. This work was supported by Grants from Air Force Office of Scientific Research, the Office of Naval Research, the National Science Foundation, and the Ministry of Education, Korea through the BK21 program of the ECE division at POSTECH.

J.-Y. Choi and J. S. Lee are with the Department of Electronic and Electrical Engineering, Pohang University of Science and Technology, Pohang 790-784, Korea (e-mail: cjy@postech.ac.kr).

M. Krstić and K. B. Ariyur are with the Department of Mechanical and Aerospace Engineering, University of California at San Diego, La Jolla, CA 92093-0411 USA (e-mail: krstic@ucsd.edu).

Publisher Item Identifier S 0018-9286(02)02073-1.