On the Synthesis of Safe Control Policies in Decentralized Control of Discrete Event Systems

Kurt Rohloff and Stéphane Lafortune

Abstract— **State estimation and safe controller synthesis for a general form of decentralized control architecture for discrete event systems is investigated. For this architecture, controllable events are assigned to be either "conjunctive" or "disjunctive". A new state estimator that accounts for past local control actions when calculating the set of estimated system states is presented. The new state estimator is applied to a previous general decentralized control law. The new control method generates a controlled language at least as large as that generated by the original method if a safety condition is satisfied. An algorithm for generating locally maximal control policies for a given state estimate is also discussed. The algorithm allows an amount of "steering" of the controlled system through an event priority mechanism.**

I. Introduction

Decentralized control is commonly the most efficient or only method to control a system too large or complex to be controlled by a single centralized controller. Decentralized control systems are implemented as a number of local controllers that have authority over a subset of a system's controllable events. As system operation evolves, the local controllers make observations of the system's behavior and maintain local estimates of the system's state. The state information is used to during the calculation of local control actions that are combined globally to determine allowed system behavior.

Several decentralized control laws for discrete event systems have been introduced in the literature. See for example [1, 2, 5–8, 11, 13, 14]. The solvability problem for *safe* and *non-blocking* decentralized controllers is undecidable ([5, 12]). We restrict our attention to the safety problem and attempt to synthesize safe decentralized controllers that allow as much behavior as possible.

This paper builds upon Yoo and Lafortune's [14] work on the control structures for the so-called "general decentralized architecture" where the set of controllable events is partitioned into a set of disjunctive events that follow a "fusion by union" rule and a set of conjunctive events that follow a "fusion by intersection" rule. Given the necessary brevity of this technical note we assume the reader has an understanding of the material presented in [14]. A more thorough introduction is also presented in this paper's companion technical report [9] which is available for download.

Previously, many control schemes have used an inverse projection operation on the locally observed strings to estimate the current system state. However, this state estimation method does not make full use of a controller's knowledge; controllers may have memory of past local control actions. We investigate the properties of a memory-based state estimator that has knowledge of past control actions. We use the estimator to construct control laws that locally allow more behavior than the *gdec* control law in [14] for a given partition of controllable events when the legal behavior may not be controllable and coobservable. The local state estimator we discuss makes no assumptions on the specific control actions of other controllers except to assume that other controllers always enable all events.

This research was supported in part by NSF grant CCR-0082784 and, in the case of the first author, a GAANN fellowship for the 2000-2001 academic year.

Department of Electrical Engineering and Computer Science, The University of Michigan, 1301 Beal Ave., Ann Arbor, MI 48109-2122, USA {krohloff,stephane}@eecs.umich.edu; www.eecs.umich.edu/umdes

Our first control law is called *gmdec*. For sufficient safety conditions, *gmdec* always allows at least as much behavior as *gdec* if the same partition of the controllable events is used for both control laws. The authors in [5] also propose the use of information on applied control patterns in processing partial observations but their approach is different from ours.

We also develop a local greedy control policy that enables as many events as possible locally for a given state estimate. Ben Hadj-Alouane et al.[3] have done work with maximal control policies for centralized control systems and we apply their methods to general decentralized controllers. Our new control policy (VLP-GM2) enforces locally maximal control actions for a given state estimate.

In the second section of this paper, the general decentralized control work in [14] is briefly reviewed. In Section 3 the new state estimator is introduced. In the fourth section, we develop the new *gmdec* control policy. In Section 5 the VLP-GM2 algorithm is presented and discussed. Some concluding remarks are made in Section 6. We use the notation employed in [14] that we briefly review in the next section. The proofs of all results in this paper can be read in the companion technical report [9] along with more in-depth discussions of the topics presented here.

II. Problem Formulation and Previous Work

We identify some of the terminology used in this paper. The "state" of the system is the string of events that has been generated so far by the system. We consider a state estimate to be a set of possible states the system may be in at a given instance. A "state estimator" is a function that calculates a state estimate. Traditionally, many observers and controllers have used the inverse projection operation ($P^{-1}(P(\cdot))$) as a state estimator.

We restrict the possible and legal languages of the systems in this paper to be prefix-closed because we are only concerned with safety issues. We also specify that the possible and legal languages be regular so that when the discussed state estimators are implemented, the local state estimations can be determined by the states of some finite observer automata.

Let G represent the uncontrolled system modeled as an automaton and let K represent the prefix-closed legal behavior. It is assumed without loss of generality that K is controllable. If K is not controllable, the supremal controllable sublanguage of K can be found in polynomial time.

As mentioned earlier, for general decentralized control, the controllable events are partitioned into two disjoint sets, Σ_{ce} and Σ_{cd}. Events in Σ_{ce} are enabled globally if all local controllers enable them and so are called conjunctive events. Events in Σ_{cd} are enabled globally if any local controller enables them and so are called disjunctive events.

Because we discuss exclusively the general architecture in this paper, we use the definition of coobservability introduced in [14] which is an extension the original definition of coobservability discussed in Rudie and Wonham [11]. The authors of [14] relabel the coobservability of [11] as C&P coobservability and introduce D&A coobservability. The extended notion of coobservability in [14] incorporates both C&P and D&A coobservability. A controllability and coobservability theorem similar to that shown in [11] also holds for general decentralized control. As may be expected, not all specifications are controllable and coobservable, so we may be required to synthesize a safe control policy that allows as much behavior as possible in such cases. Presenting safe controller synthesis methods is one of the main objectives of this paper.

Let $\Sigma_{ter} : 2^{\Sigma^*} \to \Sigma$ where for $B \subseteq \Sigma^*$, $\Sigma_{ter}(B)$ identifies the last events of the traces in B. A notable result from the literature of decentralized control relevant later in this paper is the $\mathcal{M}$-machine constructed by Rudie and Willems [10]. Please consult the referenced paper for details on this machine, but important to our discussion is that after constructing the $\mathcal{M}$-machine for a given system, the set $\Sigma_{ter}(\mathcal{L}_m(\mathcal{M}))$ identifies all events that violate C&P coobservability when $\Sigma_{ce} = \Sigma_c$.

We use the notation $\gamma_i^{gdec}(s)$ to denote the control action of controller i using the *gdec* policy of [14] after string s has occurred. $\gamma_i^{gdec}(s)$ is formally defined as follows:

$$\begin{aligned}\gamma_i^{gdec}(s) = &\left\{\sigma \in \Sigma_{cei} : P_i^{-1}(P_i(s))\sigma \cap K \neq \emptyset\right\} \\ &\cup \left\{\sigma \in \Sigma_{cdi} : \left(P_i^{-1}(P_i(s)) \cap K\right)\sigma \cap \mathcal{L}(G) \subseteq K\right\} \\ &\cup \Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei}\end{aligned} \tag{1}$$

Note that *gdec* uses an inverse projection operation as a state estimator.

The set $\left\{\sigma \in \Sigma_{cei} : P_i^{-1}(P_i(s))\sigma \cap \overline{K} \neq \emptyset\right\}$ is the set of conjunctive events that i controls and enables locally. This equation captures the notion that controller i enables events in Σ_{cei} if the control action is ambiguous.

The set $\left\{\sigma \in \Sigma_{cdi} : \left(P_i^{-1}(P_i(s)) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq K\right\}$ identifies the set of all disjunctive events that i controls and locally enables. This equation captures the notion that controller i disables events in Σ_{cdi} if the correct control action is uncertain.

A controller cannot disable uncontrollable events so controllers must always enable the globally uncontrollable events in Σ_{uc}. For notational simplicity, a controller i also lists locally uncontrollable conjunctive events $(\Sigma_{ce} \setminus \Sigma_{cei})$ as locally enabled.

To achieve K exactly using *gdec*, K needs to be coobservable. Under some sufficient conditions when *gdec* is used to control a system G, the resulting generated behavior will be safe even if K is not coobservable. We attempt to build better control laws than *gdec* that avoid illegal behavior when the specification is not coobservable. One of our key improvements over previous work is in a local controller's state estimator. The $P_i^{-1}(P_i(s))$ state estimator used in [14] for *gdec* may include strings that i knows are not possible. Therefore, $P_i^{-1}(P_i(s))$ could be a generous overestimate of the current system state. Intuitively, with a more accurate state estimate, a supervisor could enable more events and achieve a larger legal sublanguage of K.

III. Improved State Estimator

To improve upon the $P_i^{-1}(P_i(\cdot))$ state estimator, an observer for a supervisor could use its memory of past control actions when calculating a state estimate. If a controller disables an event that cannot be otherwise enabled globally, the disabling controller could appropriately disregard all possible behavior after that disabled event when calculating future control actions. When calculating a state estimate in this manner we need to be sure that all events that could be enabled by other controllers are considered possible so that the true current system state is guaranteed to be in the local controller's state estimate.

We now analyze what unobservable events a local controller should assume to be enabled globally at the system state s. We suppose that the local controllers have no knowledge of the state estimates or control actions of the other controllers. Let $\gamma_i(s)$ represent the local control action at state s; $\gamma_i(\cdot)$ is the local control law at site i. Our explicit representation of the control action should not imply that we are restricting our state estimator to using a particular control law. The estimator uses its knowledge of local control actions to better estimate the current system state, so we need a representation of the local control action.

We assume that any event in Σ_{cd} can be enabled by a local controller other than i. Namely, any event in the set $\bigcup_{j\in I\setminus\{i\}} \Sigma_{cdj}$ should be considered possible to be enabled for $I = \{1, ..., n\}$. For the rest of this paper, we use the notation that $\Sigma_{cd}^{-i} \equiv \bigcup_{j\in I\setminus\{i\}} \Sigma_{cdj}$. Also, all unobservable events that controller i enables at state s are $(\gamma_i(s) \cap \Sigma_{uoi})$. For the sake of simplicity, we do not explicitly include $(\Sigma_{ce} \setminus \Sigma_{cei}) \cup \Sigma_{uc}$ in our set of possible globally enabled events because they should be included in $\gamma_i(s)$. Therefore, the set of all events that i should consider possible to occur can be expressed as $\left(\gamma_i(\cdot) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}$. We use the shorthand notation that $\Gamma_i(s) = \left(\gamma_i(s) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}$.

We can use $\Gamma_i(\cdot)$ to develop a state estimator $PS_i(\cdot)$ of controller i that takes a string of system events as input. $PS_i(\cdot)$ is necessarily recursive because the control action is continually updated as system behavior progresses and i observes events. We assume that control actions are updated instantaneously upon the observation of events. We wish to reiterate that this state estimator uses knowledge of local control actions to estimate the system state although it is not dependent on any particular control law.

$\Gamma_i(\epsilon)$ represents all events that controller i should believe to be enabled globally at initialization. The system is at state ϵ at initialization, so before any events are observed $PS_i(\epsilon)$ must be equal to $\Gamma_i(\epsilon)^* \cap \mathcal{L}(G)$.

Suppose the system has been operating for a while and string s' has occurred in the system so controller i has $PS_i(s')$ as a state estimate. Now suppose $\sigma \in \Sigma_{oi}$ occurs and let $s = s'\sigma$. After σ occurs but before any unobservable events occur i would believe the system would be at a state in the set $PS_i(s')P_i(\sigma)$. After the control action takes affect at state s, controller i knows that only strings in $\Gamma_i(s)^*$ can occur after σ and before another observable event in Σ_{oi}. So, controller i knows the system is in a state in the set $(PS_i(s')P_i(\sigma)\Gamma_i(s)^*) \cap \mathcal{L}(G)$. Therefore,

$$PS_i(s) = \left(PS_i(s')P_i(\sigma)\Gamma_i(s)^*\right) \cap \mathcal{L}(G)$$

Suppose now that σ is not observable to i. This implies $P_i(\sigma) = \varepsilon$ and $\gamma_i(s'\sigma) = \gamma_i(s')$. With these facts, it should be evident that $(PS_i(s')P_i(\sigma)\Gamma_i(s)^*) \cap \mathcal{L}(G) = PS_i(s') = PS_i(s)$ because a controller should not change its state estimate on the occurrence of unobservable events. We can now define $PS_i(\cdot) : \Sigma^* \to 2^{\Sigma^*}$:

Definition 1:

$$PS_i(s'\sigma) = \left\{\begin{array}{ll} \Gamma_i(\varepsilon)^* \cap \mathcal{L}(G) & \text{if } s'\sigma = \varepsilon \\ (PS_i(s')P_i(\sigma)\Gamma_i(s)^*) \cap \mathcal{L}(G) & \text{otherwise} \end{array}\right\} \tag{2}$$

It is straightforward to prove that $\{s\} \subseteq PS_i(s) \subseteq P_i^{-1}(P_i(s))$ which demonstrates that $PS_i(s)$ always contains the correct state estimate and is always at least as accurate as $P_i^{-1}(P_i(s))$. Notice that because $\mathcal{L}(G)$ is a regular language, the automaton states that correspond to $PS_i(s)$ can be updated in polynomial time with respect to the size of the encoding of G on the occurrence of a locally observable event σ in order to calculate $PS_i(s\sigma)$. This operation is equivalent to the "unobservable reach" operation.

IV. A New General Decentralized Control Law

Now that we have an improved state estimator for general decentralized control systems, it would be advantageous to use this knowledge to devise an improved control law. Suppose that string s' has been observed in the system so that a local controller estimates the current state to be $PS_i(s')$. As with *gdec*, strings in Σ_{uoi}^* could then occur and not be observed by i if the system were not under control. Therefore, $(PS_i(s')P_i(\alpha)\Sigma_{uoi}^*)$ should be used as an estimate of the possible future uncontrolled behavior to calculate the next control action. Remember that we make no assumptions on the behavior of the other controllers. This prompts us to define $PS_i^+(\cdot) : \Sigma^* \to 2^{\Sigma^*}$, seen below, as a local estimate of uncontrolled behavior after the last observed event.

Definition 2:

$$PS_i^+(s'\sigma) = \left\{ \begin{array}{cc} \Sigma_{uoi}^* \cap \mathcal{L}(G) & \text{if } s'\sigma = \varepsilon \\ (PS_i(s')P_i(\sigma)\Sigma_{uoi}^*) \cap \mathcal{L}(G) & \text{otherwise} \end{array} \right\} \quad (3)$$

The *gdec* control law of [14] can now be improved by replacing its state estimator with one based on $PS_i(\cdot)$. This control scheme is called *gmdec* and stands for "General Memory-based Decentralized Control law" and can be seen below. $S_{gmdec}(s)$ is the global control law generated by combining all of the memory-based local control laws in the same manner that *gdec* combines all of its local control laws.

Definition 3: gmdec Control Law

$$\begin{aligned} \gamma_i^{gmdec}(s) = &\left\{\sigma \in \Sigma_{cei} : PS_i^+(s)\sigma \cap K \neq \emptyset\right\} \quad (4) \\ &\cup \left\{\sigma \in \Sigma_{cdi} : \left(PS_i^+(s) \cap K\right)\sigma \cap \mathcal{L}(G) \subseteq K\right\} \\ &\cup \Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei} \end{aligned}$$

When implemented in an on-line manner to control a system modeled as a finite automaton, the above control method can be updated online in polynomial time as the state estimate is updated. This can be done if we represent the state estimates as automata states instead of possibly infinite strings which is why we assume that all systems are regular. State estimates can be updated using an operation similar to the unobservable reach operation in [3]. Control actions are calculated using similar operations on directed graphs. See the technical report [9] for further discussion of the computation time required for both online and offline implementations of this control method.

Now that we have defined a new control law, we investigate several properties of the languages generated by the control law. If K is controllable and coobservable, K can be achieved exactly by the *gmdec* control policy.

Proposition 1: (K is controllable and coobservable) $\Rightarrow$ $(\mathcal{L}(S_{gmdec}/G) = K)$

As may be intuitive, K might not always be controllable and coobservable, but we may still be required to devise a safe control policy for G that allows as much behavior in K as possible. There are no known algorithms for finding maximal coobservable sublanguages (if they exist), so we need to develop heuristic control methods to attempt to allow as much behavior as possible in a computationally feasible manner. An approach for control synthesis using *gdec* has been investigated in [14] where a safe partition on controllable events is calculated for *gdec*. In a similar manner, if K is not coobservable we can use *gmdec* to attempt to achieve a larger subset of the behavior in K than would be achieved using the safe partition method described in [14]. We briefly review some related results for *gmdec*.

We provide a sufficient condition for the safety of the language $\mathcal{L}(S_{gmdec}/G)$ when *gmdec* is used to achieve a sublanguage of a specification language K that is controllable but not coobservable. The condition provided here is identical to that shown in [14] and is based on the fact that if we fix the partition of controllable events such that $\Sigma_{ter}(\mathcal{L}_m(\mathcal{M})) \subseteq \Sigma_{cd}$ holds, no conjunctive events will lead to illegal behavior where $\mathcal{M}$ is the $\mathcal{M}$-machine discussed in [10] and [14].

Theorem 1: $\Sigma_{ter}(\mathcal{L}_m(\mathcal{M})) \subseteq \Sigma_{cd} \Rightarrow \mathcal{L}(S_{gmdec}/G) \subseteq K$

Now that we have introduced some of the basic properties of the *gmdec* control scheme, it would be interesting to see how the languages generated by *gmdec* compare to the languages generated by *gdec* for a given partition. Disjunctive events that normally would remain disabled by *gdec* could be enabled by a memory-based decentralized control law such as *gmdec* due to its improved state estimation technique. Given that a sufficient safety condition has been met for *gdec*, the *gmdec* control law generates a language at least as large as the language generated by *gdec*.

Theorem 2: $\mathcal{L}(S_{gdec}/G) \subseteq K \Rightarrow \mathcal{L}(S_{gdec}/G) \subseteq \mathcal{L}(S_{gmdec}/G)$

An intuitive corollary to the theorem just demonstrated is that if $\Sigma_{ter}(\mathcal{L}_m(\mathcal{M})) \subseteq \Sigma_{cd}$, *gmdec* will produce a safe language no smaller than that produced by *gdec*.

There are also sufficient conditions for *gmdec* and *gdec* to generate the same language. As mentioned earlier, one such condition is when K is controllable and coobservable. The state estimate $PS_i(s)$ used for the *gmdec* control law might gain an advantage over $P^{-1}(P_i(s))$ only when there could be disjunctive events that controller i knows are not enabled globally. This logic is formalized in the following theorem.

Theorem 3: $\left[(\forall i \in I)\left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right)\right] \Rightarrow \mathcal{L}(S_{gdec}/G) = \mathcal{L}(S_{gmdec}/G)$

V. VLP-GM2 Algorithm

Even though *gmdec* allows more behavior than *gdec*, the local control actions of *gmdec* are in general not maximal for the $PS_i(\cdot)$ state estimator. There may be some events that *gmdec* could enable locally but remain disabled. Please see [9] for several examples of this phenomenon. In this section we discuss a locally maximal control action for a given state estimate.

For a given state estimate and controllable event partition, a local controller could initially enable all events that would be enabled by *gmdec* and then filter out all of the events that are known to not add behavior to the controlled system. A local controller could then attempt to enable more locally disabled events sequentially. The controllers could use an ordered list of controllable events to determine the order in which the controllers attempt to enable events. The ordering of events in the list could reflect the relative desirability of enabling those events. A similar approach to developing a maximal control policy for *centralized* control under partial observation was developed by Ben Hadj-Alouane et al. [3] and called VLP-PO for "Variable Lookahead Policies under Partial Observation".

We describe the locally maximal control algorithm outlined above which we call VLP-GM2, meaning "Second Variable Lookahead Policies for General decentralized Memory-based control". In [9] we present an algorithm, termed VLP-GM, that is a precursor of VLP-GM2, but VLP-GM is not discussed here. VLP-GM2 calculates a supervisor's local control action based on a state estimate PS_i similar to $PS_i(\cdot)$ introduced in Section 3. For a given state estimate PS_i, VLP-GM2 initially enables all events that *gmdec* would and then attempts to enable more events in an iterative greedy manner.

When controller i uses VLP-GM2 to calculate local control actions, it considers the possibility that other local controllers may enable disjunctive events that the i^{th} controller would disable. Furthermore, when a controller attempts to enable an event, the controller needs to be sure that the unobservable behavior of all events previously enabled would still be valid after that new event is enabled. To do this, previously enabled events are verified for validity as new events are enabled.

As with the previously discussed *gmdec* control policy, for VLP-GM2 introduced below, PS_i is an iteratively updated state estimate for each local controller and represents all states that the local controller believes the system could be in after the control action is calculated. PS_i is a global variable whose contents are retained from one run of the VLP-GM2 algorithm to the next. On initialization and before any control action is taken, PS_i should be set to $\{\varepsilon\}$. The set NS_i represents all strings that could have occurred if there were no unobservable events after the last observable event. γ_{gmdec} represents the initial control action of i, i.e., all events that would be enabled by *gmdec* for state estimate NS_i. After these events are enabled, we need to test that all events in γ_{gmdec} actually add to the behavior of the system and we filter out all of the "don't care" events in *controlFilter*. *EventOrdering* is an ordered list of the controllable events and represents the relative importance of enabling the controllable events. *ControlAction* iteratively and greedily enables more locally controllable events from *EventOrdering*. The symbol γ_i represents the local control action taken by i.

```
Algorithm 1: $VLP-GM2\,(\sigma \in \Sigma_{oi} \cup \{\varepsilon\})$
$NS_i = PS_i\sigma \cap \mathcal{L}(G)$;
$\gamma_{gmdec} = \gamma_i^{gmdec}(NS_i)$;
$\gamma_{filt} = controlFilter\,(NS_i, \gamma_{gmdec})$;
$\gamma_i = ControlAction(\gamma_{filt}, NS_i)$;
$PS_i = NS_i\left[\left(\gamma_i \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$;
$RETURN\ \gamma_i$;
```

The following algorithm for $\gamma_i^{gmdec}(\cdot)$ calculates the control action of *gmdec* given the current state estimate before unobservable events occur, denoted by NS_i

```
Algorithm 2: $\gamma_i^{gmdec}(NS_i)$
$\gamma_i^{uc} = \Sigma_{uc} \cup \Sigma_{ce} \backslash \Sigma_{cei}$;
$PS_i^{gmdec} = NS_i\Sigma_{uoi}^* \cap \mathcal{L}(G)$;
$\gamma_i^d = \left\{\sigma \in \Sigma_{cdi} : \emptyset \subset \left(PS_i^{gmdec} \cap K\right)\sigma \cap \mathcal{L}(G) \subseteq K\right\}$;
$\gamma_i^e = \left\{\sigma \in \Sigma_{cei} : PS_i^{gmdec}\sigma \cap K \neq \emptyset\right\}$;
$RETURN\ \left(\gamma_i^{uc} \cup \gamma_i^d \cup \gamma_i^e\right)$;
```

The *controlFilter* algorithm makes sure that from the current state estimate and the current set of enabled or possibly enabled events, all events locally controlled and enabled can actually lead to another possible state of the system.

```
Algorithm 3: $controlFilter\,(NS_i, \gamma)$
$\gamma_{filt} = \left\{\sigma \in \Sigma_{ci} | NS_i\left[\left(\gamma \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*\sigma \cap \mathcal{L}(G) \neq \emptyset\right\}$
        $\cup\, \Sigma_{uc} \cup \Sigma_{ce} \backslash \Sigma_{cei}$;
$RETURN\ \gamma_{filt}$;
```

The algorithm *ControlAction* (presented below) is used to enable more events than would be enabled by *gmdec* alone. It attempts to greedily enable the events ranked highest in *EventOrdering* first. Note that because of the ordering of the events in *EventOrdering*, a controller in a manner "steers" the system by allowing events with a higher priority more chances to become enabled. The *EventOrdering* list does not necessarily need to have the same ordering for all controllers and differences between local lists might reflect some local control priorities. We always consider *EventOrdering* to be constant for all local controllers, but all of our results hold if the local event lists are not identical.

Inside *ControlAction*, the set ACT_i represents all events that *ControlAction* has already enabled. ACT_i is initialized to γ_{init} which is passed to the algorithm by the calling function. After the *ControlAction* algorithm has completed, the final value for ACT_i is used to update γ_i. *ControlAction* is greedy in that once an event is enabled and added to ACT_i, that event is never disabled until the next control action is calculated.

ControlAction operates by cycling through all events in the event list (*EList*) from high priority to low. *EList* is a local copy of *EventOrdering* that is modified as the algorithm progresses. If an event is enabled, it is added to ACT_i and removed from *EList* so it will not be considered again. While *ControlAction* is operating, if a high priority event is not enabled during a cycle, that event is left on the list for later consideration and the next lowest priority event is tested. If a lower priority event is enabled, it is added to ACT_i and removed from *EList*, but then *ControlAction* cycles back to the top of the *EList* to see if any high priority events can be enabled due to the enabling of the lower priority event. If an event is a "don't care" event and does not add extra behavior if enabled at a given instance, then it is not enabled by VLP-GM2.

```
Algorithm 4: $ControlAction(\gamma_{init}, NS_i)$
$ACT_i = \gamma_{init}$;
$EList = (EventOrdering - \gamma_{init}) \cap \Sigma_{ci}$;
$Pt = 1$;
$While(Pt \leq |EList|)\ do:$
    $If\ Admissible(NS_i, EList.Pt, ACT_i),\ then$
        $ACT_i = ACT_i \cup \{EList.Pt\}$;
        $EList = EList - EList.Pt$;
        $Pt = 1$;
    $Else\ Pt++$;
$RETURN\ ACT_i$;
```

The function *Admissible* determines if an individual event σ should be enabled given the set of states that the control operation will be starting from, NS_i, and the set of events currently enabled by i, ACT_i. Once an event is enabled by VLP-GM2, it is never disabled, but events in ACT_i need to still be tested when enabling new events because enabling the new events may cause the already enabled events to become illegal. If an event being tested makes a previously enabled event illegal, the tested event is assigned to be inadmissible. The algorithm *Admissible* tests to see that all events in $(ACT_i \cup \{\sigma\}) \cap \Sigma_{ci}$ are still validly enabled if σ were to be enabled. To do this, *Admissible* looks at the state estimate RS_i^+ which represents what behavior could occur in the system if σ were enabled with the events in ACT_i.

After calculating RS_i^+ the system verifies that all events in $(ACT_i \cup \{\sigma\}) \cap \Sigma_{cei}$ lead at least once to legal behavior and that all events in $(ACT_i \cup \{\sigma\}) \cap \Sigma_{cdi}$ always lead to legal behavior. If this condition holds, then σ can be legally added to ACT_i.

```
Algorithm 5: $Admissible(NS_i, \sigma, ACT_i)$
$RS_i^+ = NS_i\left[\left(ACT_i \cup \{\sigma\} \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$;
$Output = True$;
$For\ all\ \alpha \in (ACT_i \cup \{\sigma\}) \cap \Sigma_{ci}$
    $If\ \alpha\ is\ disjunctive,\ then$
        $If\ \neg\left[\emptyset \subset \left(RS_i^+ \cap K\right)\alpha \cap \mathcal{L}(G) \subseteq K\right],\ then$
            $Output = False$;
    $Else\ (\alpha\ is\ conjunctive)$
        $If\ \left[\left(RS_i^+\alpha \cap K\right) = \emptyset\right],\ then$
            $Output = False$;
$RETURN\ Output$;
```

VLP-GM2 generates a local maximal control policy for a given state estimate. This property is shown in [9].

As with *gmdec*, the control actions from VLP-GM2 can be updated in an on-line manner in polynomial time due to the assumption of regularity. These properties are further discussed in [9].

We now explore some properties of the languages generated by using the VLP-GM2 algorithm to control a system G with specification language K. As before, it is assumed without loss of generality that K is controllable but not necessarily coobservable. We can now show a sufficient safety condition for VLP-GM2.

Theorem 4: $\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd} \Rightarrow \mathcal{L}(S_{VLPGM2}/G) \subseteq K$

We can also show that VLP-GM2 always allows at least as much behavior as *gdec* if *gdec* generates a safe controlled language.

Theorem 5: $\mathcal{L}(S_{gdec}/G) \subseteq K \Rightarrow$
$\mathcal{L}(S_{gdec}/G) \subseteq \mathcal{L}(S_{VLPGM2}/G)$

This result can be used to show that if K is controllable and coobservable then K will be achieved exactly by VLP-GM2.

Corollary 1: (K is controllable and coobservable) $\Rightarrow$
$(\mathcal{L}(S_{VLPGM2}/G) = K)$

It should be apparent to the reader that many language properties are shared by systems controlled using *gmdec* and VLP-GM2, but the languages generated by these two control policies may not always be equal. For one, although we showed that $\left[(\forall i \in I)\left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right)\right] \Rightarrow \mathcal{L}(S_{gdec}/G) = \mathcal{L}(S_{gmdec}/G)$, an identical result does not hold for VLP-GM2. In [9], we show an example where a system controlled by VLP-GM2 generates more behavior than if the same system were controlled by *gdec* or *gmdec* when $\left[(\forall i \in I)\left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right)\right]$ holds. Also, *gmdec* is not afforded a degree of steering as with VLP-GM2 and *gmdec* is not a guaranteed locally maximal control policy for a given state estimate like VLP-GM2 is. A natural question to ask when one considers the "local maximality" properties of VLP-GM2 is whether VLP-GM2 generates a language always at least as large as the language generated by *gmdec*. Sadly, this is not always the case. We present in [9] three examples where $\mathcal{L}(S_{VLPGM2}/G)$ may be strictly larger than $\mathcal{L}(S_{gmdec}/G)$, $\mathcal{L}(S_{VLPGM2}/G)$ and $\mathcal{L}(S_{gmdec}/G)$ may be incomparable and $\mathcal{L}(S_{VLPGM2}/G)$ may be strictly smaller than $\mathcal{L}(S_{gmdec}/G)$ respectively. This shows that locally maximal control actions may not lead to globally maximal behavior.

We learn from these results that generally, enhanced state estimation techniques allow for more efficient control of a system, but state estimation techniques become in some sense "less efficient" as more behavior is allowed by a controller. By disabling less behavior early in the operation of a system, more potentially illegal behavior would have to be accounted for in the future. This is in fact a manifestation of the "dual aspect of control" in problems of control under partial observation [4].

VI. Conclusion

This paper explores some of the interdependence of control methods and state estimation for decentralized discrete event systems. An improved state estimator for general decentralized controllers is introduced. The state estimator improves upon previous work because it takes past control actions into account when calculating the set of possible current system states. The *gmdec* control policy is the result of combining this new state estimator with the *gdec* control scheme of [14]. The *gmdec* control method generates legal languages at least as large as the *gdec* method when *gdec* is safe, but under some conditions *gmdec* produces languages equal to those generated by the *gdec* controller.

The VLP-GM2 control scheme is also introduced. VLP-GM2 is an iterative, greedy algorithm that produces locally maximal control policies. VLP-GM2 also has a sufficient safety condition similar to one for *gmdec* and *gdec*. The languages generated by VLP-GM2 are in general incomparable with the languages generated by *gmdec*, but VLP-GM2 allows at least as much behavior as *gdec* given sufficient safety conditions. We have learned that locally maximal control policies do not imply globally maximal behavior. Developing sufficient conditions for the achievability of maximal controllable and coobservable sublanguage by online general decentralized control remains an open problem. Finding necessary or sufficient conditions for locally maximal control actions to achieve globally maximal safe behavior is also an open problem.

Acknowledgment

We would like to thank the reviewers for their helpful suggestions.

References

[1] A. Bergeron. Sharing out control in distributed processes. *Theoretical Computer Science*, 139:163–186, 1995.

[2] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya. Supervisory control of discrete-event processes with partial observations. *IEEE Trans. Auto. Contr.*, 33(3):249–260, March 1988.

[3] N. Ben Hadj-Alouane, S. Lafortune, and F. Lin. Centralized and distributed algorithms for on-line synthesis of maximal control policies under partial observation. *Journal of Discrete Event Dynamical Systems: Theory and Applications*, 6:379–427, 1996.

[4] P.V. Kumar and P. Varaiya. *Stochastic Systems: Estimation, Identification, and Adpative Control.* Prentice Hall, Englewood Cliffs, NJ, 1986.

[5] H. Lamouchi and J.G. Thistle. Effective control synthesis for DES under partial observations. In *Proc. 39th IEEE Conf. on Decision and Control*, pages 22–28, 2000.

[6] F. Lin and W. M. Wonham. Decentralized supervisory control of discrete-event systems. *Information Sciences*, 44:199–224, 1988.

[7] A. Overkamp and J. van Schuppen. Maximal solutions in decentralized supervisory control. *SIAM J. Control Optimization*, 39(2):492–511, 2000.

[8] J.H. Prosser, M. Kam, and H.G. Kwatny. Decision fusion and supervisor synthesis in decentralized discrete-event systems. In *Proc. of 1997 American Control Conference*, pages 2251–2255, 1997.

[9] K. Rohloff and S. Lafortune. Advances in state estimation and controller synthesis for general decentralized control. Technical Report CGR01-11, Department of Electrical Engineering and Computer Science, University of Michigan, 2001.

[10] K. Rudie and J.C. Willems. The computational complexity of decentralized discrete-event control problems. *IEEE Trans. Auto. Contr.*, 40(7):1313–1318, 1995.

[11] K. Rudie and W.M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Trans. Auto. Contr.*, 37(11):1692–1708, November 1992.

[12] S. Tripakis. Undecidable problems of decentralized observation and control. In *Proc. 40th IEEE Conf. on Decision and Control*, pages 4104–4109, 2001.

[13] Y. Willner and M. Heyman. Supervisory control of concurrent discrete event systems. *International Journal of Control*, 54(5):1143–1169, 1991.

[14] T.-S. Yoo and S. Lafortune. A general architecture for decentralized supervisory control of discrete-event systems. *Journal of Discrete Event Dynamical Systems: Theory and Applications*, 13(3):335–377, 2002.