Modular Supervisory Control with Equivalence-Based Conflict Resolution

R. C. Hill, D. M. Tilbury and S. Lafortune

***Abstract*— This paper proposes a set of requirements on coordinating filters that will resolve conflict among modular supervisors. Our specific approach is unique in that it employs a conflict-equivalent abstraction, which offers the potential for greater reduction than those abstractions employed in existing works on conflict resolution. The resulting control implemented by the modular supervisors in conjunction with coordinating filters meeting the proposed requirements is shown to be safe and nonblocking. Approaches for constructing these filters are discussed and it is proposed that a static state-feedback approach to control be employed that implements deterministic coordinating filter control laws by nondeterministic automata.**

I. Introduction

A framework for the control of discrete-event systems (DES) referred to as supervisory control [1] has been developed and remains an active area of research. This control paradigm is well-suited to manufacturing and computer systems, as well as to high-level coordination of complex systems. The application of this theory to complex industrial systems has been limited by the computational difficulties that arise. One methodology for addressing the complexity problem is a modular approach to control [2] [3]. In modular supervisory control a series of smaller supervisors are designed to meet various specifications individually, rather than constructing a single monolithic supervisor to meet all specifications simultaneously. This approach offers significant savings in computation, but may result in individual supervisors interfering with one another. It is possible to check a priori that the modular supervisors will not conflict, but this verification is often computationally expensive.

Abstraction has been employed to reduce the complexity associated with verifying nonconflict [4] [5], or combined with modular approaches to control that achieve nonconflict by construction [6] [7] [8], or used to add coordinators on top of the modular supervisors to resolve conflict [9] [10]. Most of these works employ for their abstraction a language projection with the observer property of [11]. The work of [9] does not presume language projection is used, but does require of their mappings the observer property. The observer property guarantees that the abstraction maintains a type of observation equivalence. The works of [4] and [8] are different in that they use the notion of conflict equivalence introduced in [12]. As demonstrated in [13], a conflict-equivalent abstraction has the potential for greater reduction in model size than can be achieved by language projection with the observer property. In this paper we leverage this potential to achieve a greater reduction in model size than is achieved by existing techniques for conflict resolution. The work of [8] also employs conflict-equivalent abstraction to incrementally build modular supervisors in a manner similar to [7], though [8] does not explicitly specify how to construct supervisors based on the abstracted models.

We build on the approach of [4] where conflict-equivalent abstractions were employed in conflict detection. This paper advances the current state of the art by providing a novel set of requirements for coordinating filters that resolve conflict among modular supervisors. A difficulty that arises, however, is that a conflict-equivalent abstraction can introduce nondeterminism. To address this difficulty, we propose that a static state-feedback approach be employed for the filters.

The outline of this paper is as follows. Section II introduces notation. Section III provides a procedure for resolving conflict assuming that filters exist, while Section IV provides sufficient conditions on the filters and discusses some approaches for their construction. Section V then applies these results to a manufacturing example, and Section VI summarizes the contributions of this paper.

II. Notation and Preliminaries

A DES is modeled by a possibly nondeterministic automaton $G = (Q, \Sigma_\tau, \delta, q_0, Q_m)$, where Q is the set of states, $\Sigma_\tau = \Sigma \cup \{\tau\}$ is the set of events including the silent event τ, $\delta : Q \times \Sigma_\tau \rightarrow 2^Q$ is the state transition function, $q_0 \in Q$ is the initial state, and $Q_m \subseteq Q$ is the set of marked states representing completion of a task. The silent event τ can be thought of as a generic unobservable event. Let Σ_τ^* be the set of all finite strings of elements of Σ_τ, including the empty string ε. Let the function δ be extended to $\delta : Q \times \Sigma_\tau^* \rightarrow 2^Q$. The notation $\delta(q, s)!$ for any $q \in Q$ and any $s \in \Sigma_\tau^*$ represents that $\delta(q, s)$ is nonempty.

Let $P_\tau : \Sigma_\tau^* \rightarrow \Sigma^*$ be the natural projection that erases the silent event τ from strings $s \in \Sigma_\tau^*$. Also, let $\mathcal{L}(G)$ and $\mathcal{L}_m(G)$ be the *generated* and *marked languages* of G respectively, defined $\mathcal{L}(G) = \{P_\tau(s) \in \Sigma^* \mid \delta(q_0, s)!\}$ and $\mathcal{L}_m(G) = \{P_\tau(s) \in \Sigma^* \mid \delta(q_0, s) \cap Q_m \neq \emptyset\}$. The *prefix-closure* of a language K is the set of all prefixes of strings in K and is denoted by $\overline{K}$. An automaton is *nonblocking* when all of its reachable states can reach a marked state. For languages, this is defined as $\overline{\mathcal{L}_m(G)} = \mathcal{L}(G)$.

A. Supervisory control

Traditionally, the theory of supervisory control [1] has been developed for application to deterministic automata models that do not include the silent event τ and are characterized by the fact that any string can take the automaton to

This work was supported in part by NSF grants CMS-05-28287 and EECS-0624821. All authors are with the University of Michigan, Ann Arbor, MI 48109-2125, USA (rchill@umich.edu; tilbury@umich.edu; stephane@umich.edu).

only a single state. Nondeterministic automata can arise due to abstraction where events are hidden by replacing them by the silent (uncontrollable) event τ. Let $\Sigma_h \subseteq \Sigma$ represent the set of events that have been hidden. We assume all automata have the same event set Σ_τ. When two automata operate concurrently they will synchronize on all events except τ, as defined below by the *synchronous composition* operator, $\|$.

Definition 1: The *synchronous composition* of two automata G_1 and G_2, where $G_1 = (Q_1, \Sigma_\tau, \delta_1, q_{01}, Q_{m1})$ and $G_2 = (Q_2, \Sigma_\tau, \delta_2, q_{02}, Q_{m2})$ is the automaton

$$G_1 \| G_2 = (Q_1 \times Q_2, \Sigma_\tau, \delta, (q_{01}, q_{02}), Q_{m1} \times Q_{m2})$$

where the transition function $\delta : (Q_1 \times Q_2) \times \Sigma_\tau \to 2^{(Q_1 \times Q_2)}$ is defined for $q_1 \in Q_1, q_2 \in Q_2$, and $\sigma \in \Sigma_\tau$ as:

for $\sigma = \tau$, $\delta((q_1, q_2), \sigma) =$

$$\begin{cases} \delta_1(q_1, \tau) \times \{q_2\}, & \text{if } \delta_1(q_1, \tau)! \text{ and } \neg\delta_2(q_2, \tau)! \\ \{q_1\} \times \delta_2(q_2, \tau), & \text{if } \neg\delta_1(q_1, \tau)! \text{ and } \delta_2(q_2, \tau)! \\ (\delta_1(q_1, \tau) \times \{q_2\}) \cup (\{q_1\} \times \delta_2(q_2, \tau)), & \text{if } \delta_1(q_1, \tau)! \text{ and } \delta_2(q_2, \tau)! \end{cases}$$

for $\sigma \in \Sigma$, $\delta((q_1, q_2), \sigma) = \delta_1(q_1, \sigma) \times \delta_2(q_2, \sigma)$ if $\delta_1(q_1, \sigma)!$ and $\delta_2(q_2, \sigma)!$

else $\delta((q_1, q_2), \sigma)$ is empty. $\diamond$

In terms of their generated languages, $\mathcal{L}(G_1) \| \mathcal{L}(G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$.

The plant G and specification E are modeled by deterministic finite state automata given in a component-wise manner:

$$G = G_1 \| \cdots \| G_n \text{ and } E = E_1 \| \cdots \| E_p$$

Modular supervisors will be built in the sense of [3]. Let H_i be an automaton realization of the closed-loop subsystem $\mathcal{S}_i / G_i'$ consisting of a plant G_i' under the control of the supervisor $\mathcal{S}_i$. The notation $\sup \mathcal{C}(K, L)$ represents the supremal controllable sublanguage of K with respect to the prefix-closed language L. The notation $\Sigma_{rel}(G)$ will represent the set of relevant events in G. *Relevant* events are defined here to be those events that are not τ and are not self-looped at every state in the automaton.

$$\begin{aligned} G_i' &= \|_{j \in J_i} G_j, \text{ where:} \\ J_i &= \{j \in \{1, \ldots, n\} \mid \Sigma_{rel}(G_j) \cap \Sigma_{rel}(E_i) \neq \emptyset\} \\ \mathcal{L}_m(H_i) &= \sup \mathcal{C}(\mathcal{L}(E_i) \cap \mathcal{L}_m(G_i'), \mathcal{L}(G_i')) \end{aligned} \quad (1)$$

The conjunction of modular supervisors succeeds in satisfying all of the component specifications, but does not guarantee nonblocking. In order to accomplish this goal, we need that the closed-loop subsystems H_i be nonconflicting. A set of automata $H_1, H_2, \ldots, H_p$ is *nonconflicting* if the synchronous composition $H_1 \| H_2 \| \ldots \| H_p$ is nonblocking. A set of automata being nonconflicting implies its corresponding set of marked languages $K_1, K_2, \ldots, K_p$ is also nonconflicting. Nonconflict of a set of languages is defined as $\overline{K_1 \cap K_2 \cap \ldots \cap K_p} = \overline{K_1 \cap K_2 \cap \ldots \cap K_p}$.

Unfortunately, a priori verification of nonconflict is generally quite expensive computationally. With this in mind, we will incrementally apply abstraction to our modular supervisors as was done in [4].

B. Equivalence reductions

Many types of equivalence relations can be employed for reducing the complexity of a model. We are specifically interested in generating reduced models that preserve conflict properties, a notion introduced in [12].

Definition 2: [12] Two automata H_1 and H_2 are said to be *conflict equivalent* if for any third automaton T, H_1 and T are nonconflicting if and only if H_2 and T are nonconflicting. If the automata H_1 and H_2 are conflict equivalent we write, $H_1 \simeq_{\text{conf}} H_2$. $\diamond$

Note that conflict equivalence respects the property of blocking. Also, two languages can be defined as conflict equivalent in a similar manner to Definition 2. Using the fact that nonconflict of two automata implies their marked languages are nonconflicting means that $H_1 \simeq_{\text{conf}} H_2$ implies $\mathcal{L}_m(H_1) \simeq_{\text{conf}} \mathcal{L}_m(H_2)$. The converse however does not hold since the automaton representation of a given language is not unique. Specifically, two automata can generate the same language and not be conflict equivalent [12], as demonstrated by the example presented in Fig. 1. In the figure, automata G_1 and G_2 generate the same language, but G_1 conflicts with G_3 while G_2 does not. Initial states are denoted by an arrow and marked states by double circles.

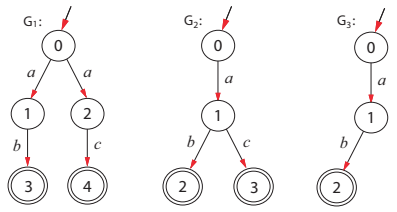

Fig. 1. Illustrative example of nondeterminism

More generally, when nondeterministic automata models are considered, language equivalence is insufficient for capturing certain system properties. A very strong equivalence relation is *bisimulation* equivalence [14]. In the following, we will use the notation $q \xrightarrow{s} q'$ to represent that state q' is reached from state q by the string $s \in \Sigma_\tau^*$.

Definition 3: Let there be two automata $G_1 = (Q_1, \Sigma_\tau, \delta_1, q_{01}, Q_{m1})$ and $G_2 = (Q_2, \Sigma_\tau, \delta_2, q_{02}, Q_{m2})$. An equivalence relation $\sim$ on the states of these automata is said to be a *bisimulation* equivalence if for any $q_1 \in Q_1$ and $q_2 \in Q_2$, $q_1 \sim q_2$ implies that for any $s \in \Sigma_\tau^*$:

if $q_1 \xrightarrow{s} q_1'$ then $\exists q_2'$ such that $q_2 \xrightarrow{s} q_2'$ and $q_1' \sim q_2'$;

if $q_2 \xrightarrow{s} q_2'$ then $\exists q_1'$ such that $q_1 \xrightarrow{s} q_1'$ and $q_1' \sim q_2'$;

$q_1 \in Q_{m1}$ if and only if $q_2 \in Q_{m2}$. $\diamond$

Two automata are said to be bisimulation equivalent if their initial states are bisimulation equivalent: $q_{01} \sim q_{02}$. If two automata are bisimulation equivalent, most properties of interest are consistent between the two, including blocking.

Another equivalence relation called *weak bisimulation* or *observation equivalence* [14] has also been defined where states are considered equivalent if they have the same "observed" futures. That is, their futures must be the same when the silent event τ is projected away. This concept

of observation equivalence is similar to the notion of the *observer property* employed in [5] [7] [9] [10].

Conflict-equivalent abstraction in general provides a greater reduction in the state size of a model than either an observation-equivalent abstraction or a projection with the observer property [13]. A conflict-equivalent abstraction, however, is not as straightforward to implement; it is implemented via heuristics and a select set of rules [4]. Also, a unique minimal reduction does not exist in general.

Example 1 demonstrates a conflict-equivalent abstraction. The notation G_a represents an abstraction of the automaton G generated by replacing the events in Σ_h by the τ event to generate an intermediate automaton G', then applying the conflict equivalence preserving rules of [4] to arrive at G_a.

Remark 1: By construction, the intermediate automaton G' and the abstraction G_a are conflict equivalent per Definition 2. However, the original automaton G and G_a are only guaranteed to be conflict equivalent with respect to automata that do not have any relevant events that were hidden in the process of generating G'. In a slight abuse of notation, we will still write that $G \simeq_{\text{conf}} G_a$. Note also that G and G_a are consistent with respect to the property of blocking.

Example 1: Consider automaton G in Fig. 2 where event f is not relevant to any other automata. Since f is "local" to G, we can hide it by replacing all occurrences of f by the silent event τ. In the resulting G', states 1 and 2 are not observation equivalent because state 1 has the observed continuation bc while state 2 does not. States 1 and 2 of G', however, can be merged by the *active events rule* of [4] leading to the conflict equivalent automaton G_a. We will consider the abstraction G_a to have the same alphabet as G, namely Σ_τ, but will not picture an event (except τ) if it is not relevant to the automaton. Therefore, one can imagine that G_a has the event f self-looped at every state. A consequence of this abstraction is that G_a is nondeterministic. $\diamond$

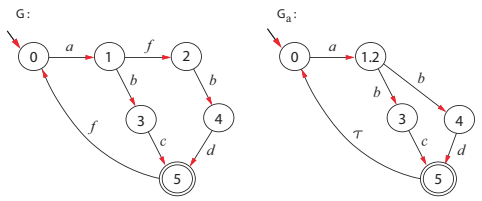

Fig. 2. Illustrative example of a conflict-equivalent abstraction

In order to make the conflict-equivalent abstraction useful, we need to show that it is preserved under the synchronous composition operation $\|$. This result follows from Proposition 1 that is a reformulation of a result from [12].

Proposition 1: Let G, G_a, and H be automata. Also assume that any events hidden in the process of generating G_a are not relevant to H, that is, $\Sigma_{rel}(H) \cap \Sigma_h = \emptyset$. If $G \simeq_{\text{conf}} G_a$ then $G\|H \simeq_{\text{conf}} G_a\|H$. See Remark 1.

The above proposition can be used to show that if no relevant events shared between G_1 and G_2 are hidden, then $G_1\|G_2 \simeq_{\text{conf}} G_{1,a}\|G_{2,a}$.

III. Conflict Resolution Procedure

In this section we describe a procedure by which conflict is detected and resolved among modular supervisors without specifying how the conflict-resolving filters are constructed. This will be discussed in Section IV.

Step 1: Build modular supervisors according to (1) which results in a deterministic closed-loop automaton H_i for each specification E_i. Note that the supervisors may be constructed in other ways.

Step 2: For each supervised subsystem H_i, generate a conflict-equivalent abstraction $H_{i,a}$ employing the rules of [4]. The set of hidden events in this step corresponds to those events relevant to only a single H_i, that is, $\Sigma_h = \Sigma - \bigcup_{i \neq j}(\Sigma_{rel}(H_i) \cap \Sigma_{rel}(H_j))$.

Step 3: Choose an abstracted subsystem $H_{1,a}$ with which to begin the procedure. Let $H'_{i,a} = H_{1,a}$ where $i = 1$ is the index for the individual closed-loop modules. Also initialize the index for filters, $j = 1$.

Step 4: Choose one of the remaining abstracted subsystems, $H_{i+1,a}$, to compose with $H'_{i,a}$. This operation is performed via synchronous composition, $H'_{i,a}\|H_{i+1,a}$.

Step 5: If the composition $H'_{i,a}\|H_{i+1,a}$ is nonblocking, skip to Step 7, otherwise proceed to Step 6.

Step 6: At this point a coordinating filter law $\mathcal{H}_{filt,j} : \mathcal{L}(H'_{i,a}\|H_{i+1,a}) \longrightarrow 2^{\Sigma}$ must be generated to resolve the detected conflict in the preceding blocking composition. Otherwise stated, $\mathcal{H}_{filt,j}$ is built so that the controlled system $\mathcal{H}_{filt,j}/(H'_{i,a}\|H_{i+1,a})$ is nonblocking. Specific requirements for this filter and an approach for its construction will be discussed in Section IV. After the filter is constructed, increment the index j.

Step 7: If all controlled subsystems have been addressed, then the procedure is finished. Otherwise, more abstraction is performed and this overall procedure is repeated beginning at Step 4. The abstraction is performed in order to take advantage of the fact that some events are no longer relevant to any remaining abstracted subsystems and hence can now be hidden. More precisely, the set of hidden events becomes

$$\Sigma_h \leftarrow \Sigma_h \cup (\Sigma - \bigcup_{k>i+1} \Sigma_{rel}(H_{k,a})) \tag{2}$$

and $H'_{i+1} = H'_{i,a}\|H_{i+1,a}$ is abstracted to generate $H'_{i+1,a}$. The index i is incremented before returning to Step 4. $\diamond$

The process in Step 4 to Step 7 of abstracting and composing subsystems and adding filters as necessary to prevent blocking is repeated until there are no more subsystems remaining. The work of [4] offers a sizable survey of heuristics for determining the ordering with which subsystems are addressed. The end result of this procedure is a set of filters that act in conjunction with the set of modular supervisors. If the controlled subsystems are nonconflicting on their own, no filters are needed. If a filter is generated that is the empty automaton, it is possible that a nonempty filter can be found by abstracting away fewer details of the controlled subsystems, that is, by making Σ_h smaller. A nonempty solution could also be found by addressing the supervisors in a different order.

IV. Coordinating Filters

In the preceding section a Conflict Resolution Procedure (CRP) was presented for incrementally resolving conflict among a set of modular supervisors. This approach to conflict resolution is the first to employ conflict-equivalent abstraction in the construction of the coordinating filter laws. In this section, we will provide a set of conditions on these coordinating filter laws and will discuss some approaches for their construction.

A. Language-based requirements

In the CRP, each filter law $\mathcal{H}_{filt,j}$ is built with respect to a blocking composition of abstracted automata that have preceded it. Here we will denote the associated blocking composition $B_{j,a}$. In the proofs that follow, we will assume the control required by each filter law $\mathcal{H}_{filt,j}$ is deterministic, realized by a (possibly nondeterministic) nonblocking automaton $H_{filt,j}$, and applied via synchronous composition, that is, $\mathcal{H}_{filt,j}/B_{j,a} = H_{filt,j}\|B_{j,a}$. Therefore,

$$B_{j,a} = (H_{filt,j-1}\| \ldots (H_{filt,1}\|H_{1,a}\|H_{2,a})_a \ldots)_a\|H_{i_j,a}$$

We will also prove that filter automata meeting the following requirements will provide safe, nonblocking control when acting in conjunction with the modular supervisors:

R1) $H_{filt,j}\|B_{j,a}$ is nonblocking

R2) $\mathcal{L}(H_{filt,j})$ is language controllable w.r.t $\mathcal{L}(B_{j,a})$

R3) $\Sigma_{rel}(H_{filt,j}) \cap \Sigma_h = \emptyset$

In the above, the property of *language controllability* is defined as in [1]. Also, requirement $R3$ is meant to prevent a given filter law from trying to affect the occurrence of events that have been hidden. Since the set Σ_h changes over time, it is implicit in $R3$ that Σ_h be the set taken at the time the filter automaton $H_{filt,j}$ is constructed. We will now demonstrate global nonblocking. In the following we will assume sequential ordering of the automata without loss of generality.

Theorem 2: Let H_i be the automaton representing the behavior of the i^{th} controlled subsystem where $i \in \{1, \ldots, p\}$. Also let there be filter automata $H_{filt,j}$, $j \in \{1, \ldots, k\}$, constructed according to the CRP and satisfying requirements *R1* and *R3*. Then the conjunction of supervised subplants and filters $H_{filt,1}\| \ldots \|H_{filt,k}\|H_1\| \ldots \|H_p$ is nonblocking.

Proof:

• By the procedure of Section III, automata are incrementally composed and abstracted. Assume the first two abstracted automata do not conflict. Therefore, $H_{1,a}\|H_{2,a}$ is nonblocking. Since $\Sigma_{rel}(H_1) \cap \Sigma_{rel}(H_2) \subseteq (\Sigma - \Sigma_h)$, $H_{1,a}\|H_{2,a} \simeq_{\text{conf}} H_1\|H_2$ by Proposition 1. Therefore, $H_1\|H_2$ is also nonblocking since conflict equivalence preserves blocking properties.

• Assume the addition of a third automaton also does not cause conflict, then $(H_{1,a}\|H_{2,a})_a\|H_{3,a}$ is nonblocking. Noting again that conflict equivalence holds across synchronous composition, $(H_{1,a}\|H_{2,a})_a\|H_{3,a}$ is conflict equivalent to $H_{1,a}\|H_{2,a}\|H_3$. Since those events made silent in the generation of $H_{1,a}$ and $H_{2,a}$ are not relevant to any of the remaining subsystems, Proposition 1 provides that $H_{1,a}\|H_{2,a}\|H_3 \simeq_{\text{conf}} H_1\|H_2\|H_3$. Furthermore, since equivalence relations are transitive, $(H_{1,a}\|H_{2,a})_a\|H_{3,a} \simeq_{\text{conf}} H_1\|H_2\|H_3$. Therefore, $H_1\|H_2\|H_3$ is also nonblocking.

• Assume for the first i_1 automata addressed, where $1 \leq i_1 \leq p$, no conflict is detected. Therefore the resulting nested composition given below is nonblocking.

$$((\ldots((H_{1,a}\|H_{2,a})_a\|H_{3,a})_a\|\ldots)_a\|H_{i_1-1,a})_a\|H_{i_1,a} \quad (3)$$

Following the logic above, the expression in (3) is conflict equivalent to $H'_{i_1} = H_1\|H_2\|\ldots\|H_{i_1}$. Therefore, H'_{i_1} is nonblocking since the expression in (3) is.

• If $i_1 = p$, then there are no filters and we are done. Otherwise, the filter $H_{filt,1}$ is needed to resolve the conflict in $H'_{i_1,a}\|H_{i_1+1,a}$, where $H'_{i_1,a}$ is the further abstraction of the expression in (3). By $R1$, $H_{filt,1}$ is nonconflicting with $H'_{i_1,a}\|H_{i_1+1,a}$. Therefore, $H_{filt,1}\|H'_{i_1,a}\|H_{i_1+1,a}$ is nonblocking. This result means that $H_{filt,1}\|H'_{i_1}\|H_{i_1+1}$ will also be nonblocking by Proposition 1 since $H'_{i_1,a}\|H_{i_1+1,a}$ and $H'_{i_1}\|H_{i_1+1}$ are conflict equivalent and since no events in Σ_h are relevant to $H_{filt,1}$ by $R3$.

• Repeating this process, supervisors and filters are added to the composition until they have all been addressed. The resulting composition $H_{filt,1}\| \ldots \|H_{filt,k}\|H_1\| \ldots \|H_p$ is, therefore, shown to be nonblocking. ■

We now need to demonstrate that the control required of these filters is realizable. For a deterministic control law, this corresponds to demonstrating language controllability. In this process we need that the original and reduced automata generate the same projected languages $P_h(\mathcal{L}(H)) = P_h(\mathcal{L}(H_a))$, where $P_h : \Sigma^* \rightarrow (\Sigma - \Sigma_h)^*$ is the natural projection which erases those events that have been hidden. Specifically, this property can be demonstrated for each of the rules of [4] that are applicable to the generation of a reduction from a nonblocking automaton. This is the situation that we have in this paper. The property $P_h(\mathcal{L}(H)) = P_h(\mathcal{L}(H_a))$ is also claimed by [15] for observation-equivalent abstractions in particular.

Now recall that a reduced automaton H_a has replaced all hidden events with the τ event then added self-loops at every state for each hidden event, therefore, none of the events that have been made silent are relevant to H_a. This in turn means that $P_h^{-1}(P_h(\mathcal{L}(H_a))) = \mathcal{L}(H_a)$. Here P_h^{-1} is an inverse projection that expands the alphabet from $(\Sigma - \Sigma_h)$ to Σ. In terms of automata, P_h^{-1} adds self-loops at every state for all events in Σ_h. Therefore:

$$P_h^{-1}(P_h(\mathcal{L}(H))) = P_h^{-1}(P_h(\mathcal{L}(H_a))) = \mathcal{L}(H_a) \supseteq \mathcal{L}(H)$$

Defining the languages marked by these automata as $K = \mathcal{L}_m(H)$ and $K_a = \mathcal{L}_m(H_a)$ and assuming the automata are nonblocking, we then have that:

$$\overline{K_a} \supseteq \overline{K} \quad (4)$$

Repeated application of (4) can be used to show the following expression where each K_i is either the marked language of a modular supervisor or a coordinating filter.

$$((\ldots(\overline{K_{1,a}} \cap \overline{K_{2,a}})_a \cap \ldots)_a \cap \overline{K_{k-1,a}})_a \cap \overline{K_{k,a}} \supseteq \overline{K_1} \cap \overline{K_2} \cap \ldots \cap \overline{K_{k-1}} \cap \overline{K_k} \quad (5)$$

Equation (5) and the following well-known propositions will help to demonstrate that our filters acting in conjunction with the modular supervisors will be language controllable with respect to the global uncontrolled plant language L.

Proposition 2: Let $K, L \subseteq L' \subseteq \Sigma^*$ be languages. If K is language controllable with respect to L', then K is language controllable with respect to L.

Proposition 3: Let K_1, K_2, and $L \subseteq \Sigma^*$ be languages and let $K = K_1 \cap K_2$. If K_1 and K_2 are nonconflicting and K_1 and K_2 are language controllable with respect to L, then K is language controllable with respect to L.

The following lemma will be instrumental in showing the ultimate desired result. We will denote the languages marked and generated by the filter automata $H_{filt,j}$ as $K_{filt,j} = \mathcal{L}_m(H_{filt,j})$ and $\overline{K_{filt,j}} = \mathcal{L}(H_{filt,j})$.

Lemma 1: Let K_{filt}, K_1, K_2, ..., K_k, and $L \subseteq \Sigma^*$ be languages and L be prefix-closed. Let the subscript a represent an abstraction satisfying $\overline{K_a} \supseteq \overline{K}$. Also let K_{filt}, K_1, K_2, ..., K_k be a nonconflicting set. Let $\Sigma_u \subseteq \Sigma$ be the set of uncontrollable events. If K_{filt} is language controllable with respect to $L'_a = (\ldots(\overline{K_{1,a}} \cap \overline{K_{2,a}})_a \cap \ldots)_a \cap \overline{K_{k,a}}$ and K_1, K_2, ..., K_k are each language controllable with respect to L, then $K_{filt} \cap K_1 \cap \ldots \cap K_k$ is language controllable with respect to L.

Proof:

• It is given that K_{filt} is language controllable w.r.t. L'_a:

$$\overline{K_{filt}}\Sigma_u \cap L'_a \subseteq \overline{K_{filt}}$$

• Noting (5), intersection of both sides of the above with $L' = \overline{K_1} \cap \overline{K_2} \cap \ldots \cap \overline{K_k}$ gives us that:

$$\overline{K_{filt}}\Sigma_u \cap L' \subseteq \overline{K_{filt}} \cap L' \tag{6}$$

• It is further given that K_1, K_2, ..., K_k are each language controllable w.r.t. L. Hence, $L'\Sigma_u \cap L \subseteq L'$. This fact combined with (6) gives us that:

$$\overline{K_{filt}}\Sigma_u \cap (L'\Sigma_u \cap L) \subseteq \overline{K_{filt}}\Sigma_u \cap L' \subseteq \overline{K_{filt}} \cap L'$$

and substituting the expression for L' we get

$$(\overline{K_{filt}} \cap \overline{K_1} \cap \ldots \cap \overline{K_k})\Sigma_u \cap L \subseteq \overline{K_{filt}} \cap \overline{K_1} \cap \ldots \cap \overline{K_k}$$

• Also recalling that it is given that the set K_{filt}, K_1, K_2, ..., K_k is nonconflicting, we have our desired result:

$$(\overline{K_{filt} \cap K_1 \cap \ldots \cap K_k})\Sigma_u \cap L \subseteq \overline{K_{filt} \cap K_1 \cap \ldots \cap K_k}$$

■

The following theorem provides the ultimate language controllability result we require.

Theorem 3: Let $K_i = \mathcal{L}_m(H_i)$ be the language representing the behavior of the i^{th} subplant $L'_i = \mathcal{L}(G'_i)$ under the supervision of the i^{th} modular supervisor where $i \in \{1, \ldots, p\}$. Let there also be filter languages $K_{filt,j}$, $j \in \{1, \ldots, k\}$, constructed according to the CRP and satisfying requirements $R1$ and $R2$. The conjunction of supervised languages and filters $K_{filt,1} \cap \ldots \cap K_{filt,k} \cap K_1 \cap \ldots \cap K_p$ is then language controllable with respect to the global uncontrolled plant $L = \mathcal{L}(G) = L'_1 \cap \ldots \cap L'_p$.

Proof:

• Each supervised language K_i is language controllable with respect to its associated subplant L'_i by construction. Since $L \subseteq L'_i$ for each local subplant, each supervised language is also language controllable with respect to the global plant L by Proposition 2.

• Let the set $K_1, \ldots, K_{i_1}$ be nonconflicting where $1 \leq i_1 \leq p$. Since each K_i is language controllable with respect to L, $K'_{i_1} = K_1 \cap \ldots \cap K_{i_1}$ is also language controllable with respect to L by Proposition 3.

• If $i_1 = p$, then there are no filters and we are done. Otherwise, the filter $K_{filt,1}$ is needed to resolve the conflict in the composition $K'_{i_1} \cap K_{i_1+1}$. By $R1$, $K_{filt,1}$, $K'_{i_1,a}$, and $K_{i_1+1,a}$ are nonconflicting. Also by $R2$, $K_{filt,1}$ is language controllable with respect to $\overline{K'_{i_1,a}} \cap \overline{K_{i_1+1,a}}$, where $\overline{K'_{i_1,a}} = (\ldots(\overline{K_{1,a}} \cap \overline{K_{2,a}})_a \cap \ldots)_a \cap \overline{K_{i_1,a}}$. Therefore, $K_{filt,1} \cap K'_{i_1} \cap K_{i_1+1}$ is language controllable with respect to L by Lemma 1.

• Repeating this logic, supervisors and filters are added to the composition until they have all been addressed. The resulting composition $K_{filt,1} \cap \ldots \cap K_{filt,k} \cap K_1 \cap \ldots \cap K_p$ is therefore shown to be language controllable with respect to L. ■

Theorem 2 and Theorem 3 therefore provide the desired result that deterministic filter laws built to satisfy requirements $R1$, $R2$, and $R3$ provide safe, nonblocking control when acting in conjunction with the modular supervisors.

B. Filter construction discussion

In this section we will first discuss some existing strategies for the construction of the filters required by the CRP. We are specifically interested in constructing deterministic filter laws that satisfy the conditions $R1$, $R2$, and $R3$ given in Section IV-A. After demonstrating that existing work does not provide a construction algorithm with less than exponential complexity that applies to our specific situation, we will propose a state-based approach to control.

A difficulty that arises in the filter construction process is that each blocking composition $B_{j,a}$ is possibly nondeterministic because of the abstraction employed. Determinization is not feasible because it can make a blocking automaton nonblocking and can lead the state space of the model to grow exponentially. A way to think about our problem is that each blocking composition $B_{j,a}$ is like our uncontrolled plant and we are trying to build a supervisor (the filter law $\mathcal{H}_{filt,j}$) to achieve a specification in a nonblocking manner.

If we consider our specification to be the language Σ^*, then we have a situation where the "plant" is nondeterministic and the "specification" is deterministic. Of the existing research on supervisory control in the presence of nondeterminism, some address the situation where either only the plant is nondeterministic [16] [17] or only the specification is nondeterministic [18]. Still other research allows the supervisors to be nondeterministic but only in application to partially observed deterministic plants and deterministic specifications [19] [20]. The works applicable to our situation [16] [17], demonstrate conditions for super-

visor existence, but do not provide a supervisor construction algorithm.

Another, perhaps more intuitive way to think about our situation is to consider our specification to be the trim of $B_{j,a}$. Therefore, we have a situation where our "plant" and "specification" are both nondeterministic. Research that addresses this situation is presented in [21] [22] and [23]. The work of [21] only addresses deadlock avoidance and its construction algorithm for building supervisors has exponential complexity. In the work of [22], conditions are presented under which a supervisor exists that can achieve behavior that is bisimilar to the given specification. A limitation of this work is that supervisor synthesis is not addressed other than to mention that a search can be performed over the cartesian product of the plant and specification state spaces. The work of [23] handles the situation of a nondeterministic plant and specification by converting the models to partially observed deterministic ones. At this point, traditional techniques for control under partial observation can be applied. This approach could be applied to our situation, but we hope to avoid the conversion process and the exponential complexity of the traditional techniques.

C. State-based approach

As existing works do not provide a supervisor construction algorithm for our particular situation with less than exponential complexity, we will propose a state-based approach for constructing deterministic filter laws that meet the requirements $R1$, $R2$, and $R3$. We will represent these deterministic control laws by possibly nondeterministic automata in order to keep the representation compact and in order to avoid determinizing the model. One problem that arises is that language controllability is insufficient to assess the realizability of a control law in regards to nondeterministic automata, as demonstrated by the following example.

Example 2: Consider the automata in Fig. 3 where G is the plant, H is the specification, and event b is uncontrollable. Since the string ab is in $\mathcal{L}(G)$ as well as in $\mathcal{L}(H)$, $\mathcal{L}(H)$ is language controllable with respect to $\mathcal{L}(G)$. However, the automaton H still requires that the uncontrollable event b be disabled at state 2. $\diamond$

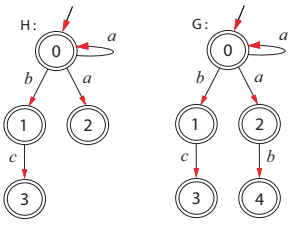

Fig. 3. State controllability example

One solution is to require a *state controllability* property similar to what was done in [18] and [22]. Language controllability requires that following an observed string s, if there is an uncontrollable continuation σ allowed in the plant automaton, then at least one instance of σ must be allowed following a string with the same observation s. With the state controllability property of [18] and [22], it is rather required that the continuation σ be allowed following every string with the observation s. In the case of subautomata, we can apply a slightly weaker notion of state controllability. Specifically, following a string with an observation s, we will require that an instance of an uncontrollable event σ must be allowed only if the event σ is possible in that particular state of the plant automaton. That is, if there is a string with an observation s that leads to a state in the plant automaton where σ is not possible, then σ does not have to be enabled at that state. Both state controllability properties imply language controllability. We will now formally define our state controllability property for a subautomaton.

Definition 4: $G_1 = (Q_1, \Sigma_\tau, \delta_1, q_{01}, Q_{m1})$ is a *subautomaton* of $G_2 = (Q_2, \Sigma_\tau, \delta_2, q_{02}, Q_{m2})$ denoted $G_1 \sqsubseteq G_2$ if and only if

$$Q_1 \subseteq Q_2,\ q_{01} = q_{02},\ Q_{m1} = Q_{m2} \cap Q_1, \text{ and}$$
$$\delta_1 = \delta_2 \text{ when restricted to } Q_1. \quad \diamond$$

Now the state controllability definition.

Definition 5: Let $\Sigma_u \subseteq \Sigma_\tau$ with $\tau \in \Sigma_u$. Subautomaton H of G is *state controllable in* G if

$$\forall s \text{ for which } \delta_H(q_o, s)! \text{ and } \forall q \in \delta_H(q_o, s) \text{ and } \forall \sigma \in \Sigma_u,\ q' \in \delta_G(q, \sigma) \Rightarrow q' \in \delta_H(q, \sigma) \quad \diamond$$

State controllability as a property, however, is not sufficient to provide that the subautomaton H represents a deterministic control law with respect to G. If the same observed string leads to two different states, those two states could require conflicting control actions. As such, we need a new observability type requirement.

Definition 6: Let $\Sigma_c \subseteq \Sigma$. Subautomaton H of G is *state observable in* G with respect to the event set Σ_c if

$$\forall s \text{ for which } \delta_H(q_o, s)! \text{ and } \forall q \in \delta_H(q_o, s) \text{ and } \forall \sigma \in \Sigma_c,\ P_\tau(s)\sigma \in \mathcal{L}(H) \text{ and } q' \in \delta_G(q, \sigma) \Rightarrow q' \in \delta_H(q, \sigma) \quad \diamond$$

Taken together, state controllability and state observability provide that H represents a deterministic control law with respect to G. This is demonstrated formally by the following theorem that shows that a deterministic automaton H_{obs} that generates and marks the same languages as H will produce a result that is bisimulation equivalent to H when it is composed with G. In essence, H_{obs} can be considered a supervisor that achieves the specification represented by the nondeterministic automaton H for the nondeterministic plant model G. Similar results for generating control for bisimulation equivalence can be found in [24] and [25], but none demonstrate the following specific result. Here we implicitly assume the states of the automata are reachable.

Theorem 4: Let H and G be nonempty (possibly nondeterministic) automata such that $H \sqsubseteq G$ and H is state controllable and state observable in G. If H_{obs} is a deterministic automaton for which $\mathcal{L}(H_{obs}) = \mathcal{L}(H)$ and $\mathcal{L}_m(H_{obs}) = \mathcal{L}_m(H)$, then the synchronous composition $H_{obs} \| G$ is bisimulation equivalent to H.

Proof: See proof in [26]. ■

The above theorem also provides a connection to the results and language-based requirements of Section IV-A. Therefore, a new set of requirements can be expressed in terms of a nondeterministic automaton model of our filter:

$R1'$) $H_{filt,j}$ is a nonblocking subautomaton of $B_{j,a}$

$R2'$) $H_{filt,j}$ is state controllable and state observable in $B_{j,a}$

$R3'$) $\Sigma_{rel}(H_{filt,j}) \cap \Sigma_h = \emptyset$

These results imply that a determinized version of the automaton $H_{filt,j}$, which we will denote $H_{filt,j,obs}$, will meet the previously established requirements $R1$, $R2$, and $R3$. Specifically, conditions $R1'$ and $R2'$ together with Theorem 4 mean that $H_{filt,j,obs} \| B_{j,a}$ is nonblocking since it is bisimulation equivalent to $H_{filt,j}$. Therefore, requirement $R1$ is satisfied. Furthermore, since $H_{filt,j}$ is state controllable, the generated language $\mathcal{L}(H_{filt,j})$ is language controllable. This then implies that $\mathcal{L}(H_{filt,j,obs})$ is also language controllable, thereby satisfying requirement $R2$. Also, $R3'$ are $R3$ are equivalent.

Thus far we have demonstrated that determinized versions of the filter automata $H_{filt,j}$ meeting requirements $R1'$, $R2'$, and $R3'$ will provide safe, nonblocking control when acting in conjunction with traditionally built modular supervisors. However, we would like to avoid the determinization process. Since the nondeterministic filter automata $H_{filt,j}$ possesses all the information that $H_{filt,j,obs}$ does, it turns out that $H_{filt,j,obs}$ never actually has to be constructed. However, the control required by the automaton $H_{filt,j}$ cannot be implemented via the synchronous composition operation. Rather, following the observation of a string s, all continuations active at all states reached by strings with the same observation must be allowed. In essence, we are using $H_{filt,j}$ to generate an online implementation of $H_{filt,j,obs}$. In order to make this more clear, consider the automaton G_a in Fig. 2. If we consider G_a to be a nondeterministic representation of a deterministic control law, then following an observation of the string ab we do not know if we are in state 3 or state 4, therefore, we have to allow both event c and event d to occur.

The bisimulation equivalence result also allows the composition $\mathcal{H}_{filt,j,obs}/B_{j,a}$ to be replaced by the subautomaton $H_{filt,j}$ in the course of the overall CRP.

Having established that we can employ a filter law represented by a nondeterministic automaton, the final question that remains is how to construct $H_{filt,j}$ so that requirements $R1'$, $R2'$, and $R3'$ are satisfied. Since we are ultimately trying to find a subautomaton, we are in essence trying to find a static state-feedback law. In other words, the control we apply depends only on the state we are in, not on the path we took to get there. It is well-established that a static control law is more restrictive than a dynamic control law in the case of partial observation [27], however, we are willing to make this sacrifice in order to avoid exponential complexity. Specifically, we can adapt to the nonblocking case existing state-feedback approaches, for example [28], that partition the state space into sets of equivalent classes that have the same observation when viewed through a "mask." The details of an adapted approach to state-feedback control are presented in [26] where it is shown that the approach has polynomial complexity and is more permissive than existing static state-feedback methodologies.

V. Comprehensive Example

In this section we will demonstrate our approach for generating nonblocking modular supervisory control through a Flexible Manufacturing System (FMS) example modified from [29] and shown in Fig. 4. The machines *Con2*, *Robot*, *Lathe*, *Con3*, *PM*, and *AM* can be thought of as components of the open-loop plant. The buffers *B2*, *B4*, *B6*, *B7*, and *B8* can be thought of as the component specifications for the system where it is desired that the buffers do not underflow or overflow. The automata models for these machines and buffers are given in Fig. 5 and Fig. 6 respectively.

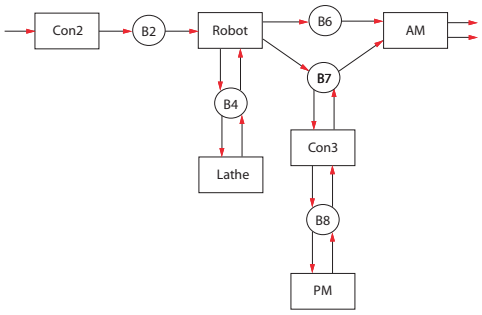

Fig. 4. Flexible manufacturing system (FMS)

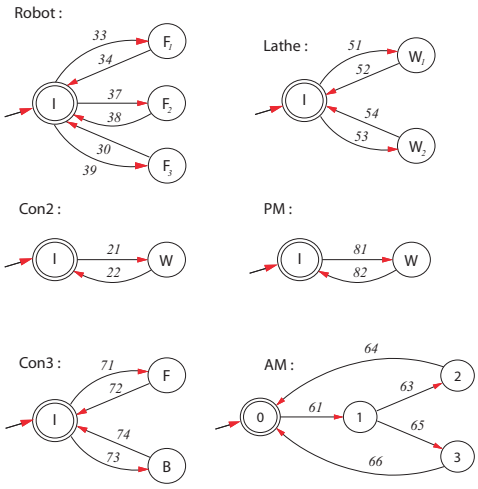

Fig. 5. Automata modeling the components of the open-loop plant

In these automata, odd labels represent controllable events and even labels represent uncontrollable events. Additionally, all automata have the same event set Σ_τ, though only relevant events are pictured.

Following the Conflict Resolution Procedure, the modular supervisors for this system are H_2, H_4, H_6, H_7, and H_8, where the subscript refers to the corresponding buffer specification. Addressing the supervisors in the order $6 \rightarrow 7 \rightarrow 4 \rightarrow 8 \rightarrow 2$, blocking is not detected until $H_{8,a}$ is added to the composition. Therefore, the blocking composition $B_{1,a} = ((H_{6,a} \| H_{7,a})_a \| H_{4,a})_a \| H_{8,a}$ becomes the "plant" for the filter construction procedure. The automaton $B_{1,a}$ is nondeterministic and a nonblocking, state controllable, and

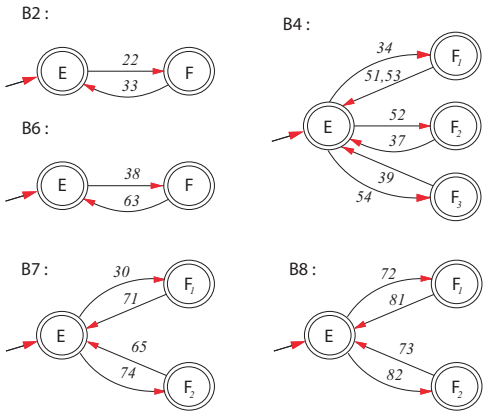

Fig. 6. Automata modeling the component buffer specifications

state observable filter automaton can be constructed by the procedure of [26].

The resulting filter $H_{filt,1}$ is a subautomaton of $((H_{6,a}\|H_{7,a})_a\|H_{4,a})_a\|H_{8,a}$, and the resulting supervised behavior $H_{filt,1,obs}\|((H_{6,a}\|H_{7,a})_a\|H_{4,a})_a\|H_{8,a}$ is bisimulation equivalent to $H_{filt,1}$. Therefore, the composition can be replaced by $H_{filt,1}$ and $H_{filt,1,obs}$ never has to be constructed. Finally, it turns out that $(H_{filt,1})_a\|H_{2,a}$ is nonblocking so no further filters are needed.

The resulting modular control achieved by the five original modular supervisors along with the conflict-resolving law $\mathcal{H}_{filt,1}$ satisfies the given specifications in a nonblocking manner, though the resulting control is more restrictive than the monolithic solution. The modular solution allows five pieces to be processed by the FMS at a given time, while the monolithic solution allows six. This loss of optimality, however, is often worth the reduction in complexity the modular approach provides.

An indication of the complexity of the modular solution in the above example is that the largest automaton that had to be built was $H_{6,a}\|H_{7,a}$ which had 136 states and 405 transitions. In the monolithic approach, the composition of all machines and buffers leads to an automaton with 13,248 states and 46,424 transitions. State size of course does not completely define the complexity of this approach and is something that must be investigated further.

VI. Conclusions

This paper has proposed a new approach for resolving conflict among traditionally-built modular supervisors. Requirements are presented for conflict-resolving filter laws that guarantee safe, nonblocking control. It is also proposed that a state-based approach to control be employed for the filter laws to avoid exponential complexity. The coordinating filters are constructed based on conflict-equivalent abstractions, which offer the potential for a greater reduction in state-size than existing work on conflict resolution. A manufacturing example is also presented showing the overall potential of this approach.

References

[1] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proc. of IEEE*, 1989.

[2] ——, "Modular supervisory control of discrete event systems," *Mathematics of Control, Signal and Systems*, 1988.

[3] M. H. de Queiroz and J. E. R. Cury, "Modular supervisory control of composed systems," in *Proc. ACC*, 2000.

[4] H. Flordal and R. Malik, "Modular nonblocking verification using conflict equivalence," in *Proc. WODES*, 2006.

[5] P. Pena, J. Cury, and S. Lafortune, "Testing modularity of local supervisors: An approach based on abstractions," in *Proc. WODES*, 2006.

[6] K. Schmidt, T. Moor, and S. Perk, "A hierarchical architecture for nonblocking control of discrete event systems," in *Mediterranean Conf. Control and Automation*, 2005.

[7] R. Hill and D. Tilbury, "Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction," in *Proc. WODES*, 2006.

[8] P. Malik, R. Malik, D. Streader, and S. Reeves, "Modular synthesis of discrete controllers," in *Proc. ICECCS*, 2007.

[9] K. Wong and W. Wonham, "Modular control and coordination of discrete-event systems," *Discrete Event Dynamic Systems: Theory and Applications*, 1998.

[10] L. Feng and W. Wonham, "Computationally efficient supervisor design: Abstraction and modularity," in *Proc. WODES*, 2006.

[11] K. Wong and W. Wonham, "Hierarchical control of discrete-event systems," *Discrete Event Dynamic Systems: Theory and Applications*, 1996.

[12] R. Malik, D. Streader, and S. Reeves, "Conflicts and fair testing," *International Journal of Foundations of Computer Science*, 2006.

[13] R. Malik, H. Flordal, and P. Pena, "Conflicts and projections," in *Proc. DCDS*, 2007.

[14] R. Milner, *Communication and Concurrency*. London: Prentice-Hall, Inc, 1989.

[15] R. Su and J. Thistle, "A distributed supervisor synthesis approach based on weak bisimulation," in *Proc. WODES*, 2006.

[16] R. Kumar and M. Shayman, "Non-blocking supervisory control of nondeterministic discrete-event systems via prioritized synchronoziation," *IEEE Trans. Automat. Contr.*, 1996.

[17] S. Park and J. Lim, "Nonblocking supervisory control of nondeterministic systems based on multiple deterministic model approach," *IEICE Trans. Inf. & Syst.*, 2000.

[18] M. Fabian and B. Lennartson, "On non-deterministic supervisory control," in *Proc. 35th IEEE Conf. Decision & Control*, 1996.

[19] K. Inan, "Supervisory control: Theory and application to the gateway synthesis problem," in *Belgian-French-Netherlands Summer School on Discrete Event Systems*, Spa, Belgium, 1993.

[20] R. Kumar, S. Jiang, C. Zhou, and W. Qiu, "Polynomial synthesis of supervisor for partially observed discrete-event systems by allowing nondeterminism in control," *IEEE Trans. Automat. Contr.*, 2005.

[21] A. Overkamp, "Supervisory control using failure semantics and partial specification," *IEEE Trans. Automat. Contr.*, 1997.

[22] C. Zhou, R. Kumar, and S. Jiang, "Control of nondeterministic discrete-event systems for bisimulation equivalence," *IEEE Trans. Automat. Contr.*, 2006.

[23] M. Heymann and F. Lin, "Nonblocking supervisory control of nondeterministic systems," Technion, Israel Institute of Technology, Haifa, Israel, Tech. Rep. CIS-9620, Oct. 1996.

[24] P. Madhusudan and P. Thiagarajan, "Branching time controlers for discrete event systems," *Theoretical Computer Science*, 2002.

[25] P. Tabuada, "Open maps, alternating simulations and control synthesis," in *International Conference on Concurrency Theory*, 2004.

[26] R. Hill, "Modular verification and supervisory controller design for discrete-event systems using abstraction and incremental construction," Ph.D. dissertation, University of Michigan, Ann Arbor, USA, 2008.

[27] R. Kumar, V. Garg, and S. Marcus, "Predicates and predicate transformers for supervisory control of discrete event dynamical systems," *IEEE Trans. Automat. Contr.*, 1993.

[28] S. Takai and S. Kodama, "Characterization of all M-controllable subpredicates of a given predicate," *Int. J. of Control*, 1998.

[29] M. H. de Queiroz, J. E. R. Cury, and W. Wonham, "Multitasking supervisory control of discrete-event systems," *Discrete Event Dynamic Systems: Theory and Applications*, 2005.