Modular Supervisory Control of Discrete-Event Systems with Abstraction and Incremental Hierarchical Construction

R.C. Hill and D.M. Tilbury

***Abstract*— This paper addresses the problem of state explosion by outlining a procedure for incrementally building modular supervisors that are nonconflicting by construction. Abstractions are employed to make the procedure more computationally feasible. Proof is given showing the set of modular supervisors generated in this manner meet given specifications without blocking. Furthermore, an example is provided that demonstrates the reduction in complexity that this approach provides.**

I. INTRODUCTION

In recent years, a well formed theory for the control of discrete-event systems (DES) has developed following the supervisory control framework introduced by Ramadge and Wonham [1]. A significant hurdle to the adoption of these methods is the state explosion that occurs in modelling systems of the size most commonly found in industry.

Consider the transfer line example shown in Fig. 1 consisting of six machines and four buffers. The traditional approach to this control problem is to build a single monolithic supervisor to control the entire combined system. Assuming that each machine model has five states and each buffer model has two states, the maximum possible size of the supervisor for this simple system grows quickly to 250,000 states ($5^6 \times 2^4$).

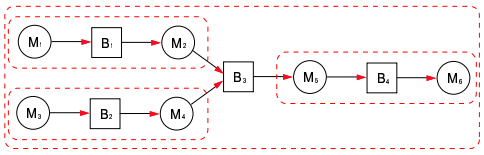

Fig. 1. transfer line example

One possible solution is to apply hierarchical methods where the supervisor is designed for an abstracted version of the system [2] [3]. The idea is that by abstracting away details of the system, it becomes more understandable and can decrease the amount of necessary computation.

The primary limitation here is that maintaining "consistency" between the high and low levels of the hierarchy is often difficult and limits either the amount of abstraction or the quality of results that can be achieved. Another problem is that generally the full monolithic system must be built before the abstraction is performed.

Another approach is to build a series of smaller component supervisors that act in conjunction, rather than a single monolithic supervisor. This approach, which is referred to as modular supervisory control, offers significant gains in computational complexity and understandability [4] [5] [6] [7].

While this modular approach to supervision has been shown to guarantee the satisfaction of specifications, there is no guarantee that the system will be able to complete its desired tasks. In other words, the system may block or reach a deadlock. To avoid this, it is necessary to verify that the individual supervisors have the property that they are nonconflicting. Unfortunately, the process of verifying nonconflict is approximately as computationally intensive as building the monolithic supervisor in the first place [8].

The approach we propose in this paper addresses the problem of complexity in supervisory control by adopting some of the elements of existing hierarchical and modular approaches, while at the same time avoiding some of their weaknesses. The overall goal is to generate a set of modular supervisors that, when acting in conjunction, result in a controlled system that is guaranteed to meet given specifications and to be nonblocking, without having to build the full unabstracted system and without having to verify that the component supervisors are nonconflicting.

For the previously given transfer line example, a possible partition generated by our proposed approach is shown in Fig. 1. The systems in each of the three inner dashed blocks are supervised by their own supervisor and the outer dashed block represents the system supervised by the fourth supervisor. Nonconflict of each of the supervisors is guaranteed since the three supervisors on the first level of the hierarchy are "disjoint", and the supervisor on the second level is a subset of all the supervisors on the first level. The problem one might recognize is that the fourth supervisor acts on the full system, which is counter to the goal of a modular approach. The solution is to perform an abstraction on the supervised systems in the inner blocks, before moving up a level of the hierarchy to design the remaining supervisor. In this way, abstractions are performed incrementally, rather than on the entire system at once.

The outline of the rest of the paper is as follows: Section II introduces notation and some definitions, section III uses an example to demonstrate the procedure for generating the modular supervisors, section IV demonstrates the conjunction of these supervisors will be controllable and nonblocking without consideration of an abstraction, section V repeats the proof with the addition of abstraction, and section VI summarizes the work of the paper and outlines some areas of further investigation.

This work was supported in part by NSF CMS 05-28287.

R.C. Hill and D.M. Tilbury are with the Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109-2125, USA (rchill@umich.edu; tilbury@umich.edu).

II. NOTATION AND PRELIMINARIES

In this work we will consider DES modelled by automata in the context of the supervisory control framework put forth by Ramadge and Wonham [1] [8]. The automaton for generating the language representing the set of possible behaviors of our DES is represented by the five-tuple $G = (X, \Sigma, \delta, x_0, X_m)$, where X is the set of states, Σ is the set of events, $\delta : X \times \Sigma \rightarrow X$ is the partial state transition function, $x_0 \in X$ is the initial state, and $X_m \subseteq X$ is the set of marked states representing successful termination of a process. Let Σ^* be the set of all finite strings of elements of Σ, including the empty string ε. The partial function δ can be extended to $\delta : X \times \Sigma^* \rightarrow X$ in the natural way. The notation $\delta(s, x)!$ for any $s \in \Sigma^*$ and any $x \in X$ denotes that $\delta(s, x)$ is defined. The *generated* and *marked languages* of G, denoted by $\mathcal{L}(G)$ and $\mathcal{L}_m(G)$ respectively, are defined by $\mathcal{L}(G) = \{s \in \Sigma^* | \delta(s, x_0)!\}$ and $\mathcal{L}_m(G) = \{s \in \Sigma^* | \delta(s, x_0) \in X_m\}$. The notation $\overline{L}$ represents the set of all prefixes of strings in the language L, and is referred to as the *prefix closure* of L. An automaton is said to be *nonblocking* when $\overline{\mathcal{L}_m(G)} = \mathcal{L}(G)$.

For the purposes of control, the event set of an automaton is partitioned into *controllable* and *uncontrollable* events, $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$, where controllable events can be disabled and uncontrollable events cannot. Therefore a supervisor, denoted S, is a mapping that, upon observation of a string generated by a plant G, outputs a list of events to be disabled. Keeping in mind that uncontrollable events are not allowed to be disabled, $S : \mathcal{L}(G) \rightarrow 2^{\Sigma - \Sigma_u}$. Given a set of allowed behaviors $K \subseteq \mathcal{L}(G)$ and partitioning of the event set $\Sigma_u \subseteq \Sigma$, the existence of a supervisor that can successfully restrict the operation of the plant within the behavior allowed by the specification is guaranteed by satisfaction of the following *controllability condition* [8]:

$$\overline{K}\Sigma_u \cap \mathcal{L}(G) \subseteq \overline{K} \tag{1}$$

The operation of two automata together is captured via the *synchronous composition* (parallel composition) operator, $\|$. By representing the supervisor mapping S as an automaton H, and the open loop plant as a separate automaton G, the closed loop or supervised behavior of the system can be modelled using the synchronous composition operator, $H\|G$. When composed, events not shared by both automata are allowed to occur without participation of the other automaton, while those events that are shared must occur with the two automata synchronized. In order to precisely define the synchronous composition operator, the *projection*, P, is defined as follows:

$$\begin{aligned} P_i(\varepsilon) &:= \varepsilon \\ P_i(e) &:= \begin{cases} e, & e \in \Sigma_i \subseteq \Sigma \\ \varepsilon, & e \notin \Sigma_i \subseteq \Sigma \end{cases} \\ P_i(se) &:= P_i(s)P_i(e), s \in \Sigma^*, e \in \Sigma \end{aligned} \tag{2}$$

Given a string $s \in \Sigma^*$, the projection P_i erases those events in the string that are in global alphabet $\Sigma = \cup \Sigma_i$, but not in the local alphabet Σ_i of the language L_i. We can also define the inverse projection as follows:

$$P_i^{-1}(t) := \{s \in \Sigma^* : P_i(s) = t\} \tag{3}$$

These definitions can be naturally extended to languages and then applied to give a formal definition of the synchronous composition.

$$L_1 \| L_2 \| \cdots \| L_n := \bigcap_{i=1}^{n} P_i^{-1}(L_i) \tag{4}$$

A couple of restrictions on the projection operation that will be needed later will now be defined. The first requirement is that the projection have the L_m-*observer property* as defined in [9].

Definition - P_i has the L_m-observer property $\iff$ $[(\forall s \in L)(\forall t \in \Sigma_i^*) P_i(s)t \in P_i(L_m) \implies (\exists u \in \Sigma^*) su \in L_m,\ and\ P_i(su) = P_i(s)t]$

Intuitively, the observer property means that any branching of the unabstracted model can be observed in its projected version. In the above and following definitions, $\Sigma_i^* = P_i(\Sigma^*)$. A useful byproduct of the observer property is that it guarantees that the projected system is simpler than the original system, and that the computation of the projected model is at worst polynomial in time [10].

The second requirement is that the projection maintain a *consistency of marking*. We will define this to mean that $L_m = P_i^{-1}(P_i(L_m)) \cap L$, the idea here being that the inverse projection doesn't add in strings from L that don't run to completion.

In addition to determining the existence of a supervisor S that can achieve a given specification, it is also desirable that the controlled system be nonblocking. A nonblocking supervisor S for G exists if and only if the controllability condition of (1) is satisfied and an $\mathcal{L}_m(G)-$closure condition defined as $K = \overline{K} \cap \mathcal{L}_m(G)$ holds. If and only if both of these conditions are satisfied, then a supervisor S exists such that the supervised behavior exactly equals the admissible language, $\overline{K}$, and the set of marked behaviors exactly equals K [8]. In this sense, $\overline{K}$ can be interpreted as the set of strings allowed by the supervisor S, that is, the closed-loop behavior of the system. In general, the $\mathcal{L}_m(G)-$closure condition holds by construction of K.

In the case that the controllability condition of (1) does not hold, and a supervisor cannot be constructed to meet K, it is desirable to find the largest sublanguage of K for which a supervisor does exist. This supremal controllable sublanguage is written, $K^{\uparrow C}$.

In the work of this paper it will be assumed that the plant and specification are given in the following component-wise manner:

$$G = G_1 \| \cdots \| G_n, \text{ and } H_{spec} = H_{spec1} \| \cdots \| H_{specm}$$

In terms of languages, the uncontrolled behavior and language specification are given as:

$$L = \mathcal{L}(G) = P_i^{-1}(L_1) \cap \cdots \cap P_n^{-1}(L_n)$$
$$\overline{K_{spec}} = \mathcal{L}(H_{spec}) = P_{spec1}^{-1}(\overline{K_{spec1}}) \cap \cdots \cap P_{specm}^{-1}(\overline{K_{specm}})$$

where $L_i = \mathcal{L}(G_i)$ and $\overline{K_{specj}} = \mathcal{L}(H_{specj})$.

As stated earlier, modular supervisors won't block one other if they are *nonconflicting*. Two languages are said to be nonconflicting if they satisfy the relation $\overline{K_1} \cap \overline{K_2} = \overline{K_1 \cap K_2}$. In other words, to be nonconflicting means that if K_1 and K_2 share a prefix, they must share a string containing that prefix. In general, determining whether two languages are nonconflicting is a computationally expensive procedure. Two special cases under which languages are guaranteed to be nonconflicting are when languages have disjoint event sets ($\Sigma_1 \cap \Sigma_2 = \emptyset$), and when one language is a subset of the other ($K_1 \subseteq K_2$). In the case that two languages have shared alphabets, but the sharing arose from the addition of events by way of the inverse projection operation, then the two languages are still nonconflicting. This is deduced from the following property given in [7]:

Proposition 1: If $\Sigma_1 \cap \Sigma_2 = \emptyset$ then $P_1^{-1}(K_1) \cap P_2^{-1}(K_2) = P_1^{-1}(K_1) \cap P_2^{-1}(K_2)$

III. PROCEDURE WITH EXAMPLE

In this section we will outline the process by which the set of modular supervisors is generated through application to a manufacturing example taken from [11] and shown here in Fig. 2. The "plant" consists of a *Robot*, a Conveyor (C), a Painting Machine (PM), and an Assembly Machine (AM). The two buffers connecting the various machines, B_7 and B_8, serve as the specifications for the system where it is desired that that buffers do not underflow or overflow. It is interesting to note that if two modular supervisors are built in the traditional manner, their conjunction will result in blocking even though they are individually nonblocking.

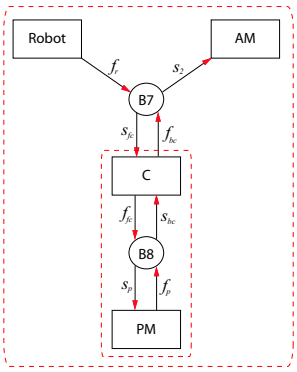

Fig. 2. simple flexible manufacturing system example

The automata models for the different machines are shown in Fig. 3. Note that each of the starting events, s_i, are controllable, and each of the finishing events, f_i, are uncontrollable.

The finite state automaton models for the buffers are shown in Fig. 4.

In the following procedure, each series of automata will be indexed sequentially. Furthermore, $P_i^{-1}(\mathcal{L}(G_i'))$ will be written as L_i', $P_i^{-1}(\mathcal{L}_m(G_i'))$ as $L_{m,i}'$, $P_i^{-1}(\mathcal{L}(G_i''))$ as L_i'' and so forth. The inverse projection P_i^{-1} lifts the local event sets, Σ_i, up to the global level, $\Sigma = \cup\Sigma_i$.

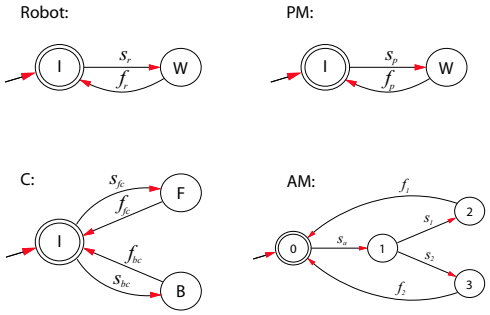

Fig. 3. finite state automaton model of each machine

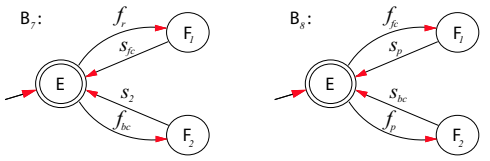

Fig. 4. finite state automaton model of each buffer

1) **abstract away strictly private events**
Before the "real" procedure begins, go through each plant submodule and attempt to abstract away any events not shared with any other plant submodule or specification, $G_i \rightarrow G_{i,a}$. Of all the events in the alphabet of our simple example, s_r, s_a, s_1, f_1, and f_2 are all considered private. However, it turns out none of these events can be abstracted away since that would cause a violation of the L_m-observer property or the consistency of marking property.
Specifically, abstraction of events s_r, s_a, f_1 and f_2 cause a loss of the consistency of marking property because they represent transitions between states with different marking properties. Furthermore, abstraction of event s_1 by itself causes a loss of the observer property because event s_2 leaves state 1 but not state 2. Otherwise stated, if event s_1 is abstracted away, then strings s_a and $s_a s_1$ would have the same projection, but different observed futures since s_2 is an observed continuation of s_a, but not of the string $s_a s_1$.
2) **group modules within the current level**
– Pick a specification H_{spec1}, and group it with all plant submodules with which it shares an event. We will assume that all plant submodules have already been organized such that they don't share events with each other. All interaction will take place through the specifications. For our example, we will choose buffer B_8 to be the first specification and its corresponding subplants are C, and PM.
– If another specification H_{spec2} can be found that doesn't share any events with any members of the H_{spec1} group, group it with all of its affected plant submodules. In our case, the only other specification,

B_7, shares events with C, which is a member of the first group. Therefore, we will have to wait to address that specification until we move up to the next level of the hierarchy.

– In general, this process of partitioning is continued until there are no more independent groups.

3) **compose subplant members of each group**
Perform a synchronous composition of all plant submodules in each grouping. Now each group has a single plant and a single specification.

$$\begin{aligned} G'_1 &= G_{1,a} \| \cdots \| G_{p,a} \\ G'_2 &= G_{p+1,a} \| \cdots \| G_{q,a} \\ \vdots &= \vdots \\ G'_k &= G_{r+1,a} \| \cdots \| G_{s,a} \end{aligned}$$

In our factory example, we currently have only one grouping. The resulting composition, $C \| PM$, gives us the plant for our first module, G'_1.

4) **perform the next phase of abstraction**
Perform an abstraction on all modules, $G'_i \rightarrow G'_{i,a}$, again maintaining the L_m-observer and consistency of marking properties. Note that the observer property only needs to be maintained one level at a time. This step is not necessary on the first level of the hierarchy, since all strictly private events were already addressed in step 1. Because we are still on the first level of our example, no further abstraction will take place at this point, $G'_{1,a} = G'_1$.

5) **build a supervisor for each abstracted module**
Following the traditional techniques for supervisor construction, build a nonblocking supervisor for each abstracted plant/specification pair.

$$\begin{aligned} K_{1,a} &= (\overline{K_{spec1,a}} \cap P_1(L'_{m,1}))^{\uparrow C} \\ K_{2,a} &= (\overline{K_{spec2,a}} \cap P_2(L'_{m,2}))^{\uparrow C} \\ \vdots &= \vdots \\ K_{k,a} &= (\overline{K_{speck,a}} \cap P_k(L'_{m,k}))^{\uparrow C} \end{aligned}$$

It should be noted that each $\overline{K_{speci,a}}$ isn't an abstraction, but rather each is a version of $\mathcal{L}(H_{spec,i})$ with an inverse projection that only adds events up to the alphabet of the current level of the hierarchy. Furthermore, $P_i(L'_{m,i})$ represents the marked language generated by the abstracted plant automaton $\mathcal{L}_m(G'_{i,a})$. The local event set, Σ_i, is defined by the alphabet of $G'_{i,a}$ and contains the alphabet of $H_{spec,i}$. If a nonblocking supervisor does not exist for this abstracted system, it is possible that a supervisor can be found by including fewer events in the abstraction.

For our example system, the abstracted allowable language for the first specification is generated by the automaton given by $B_8 \| G'_{1,a}$. It turns out this language is not controllable and hence the supremal controllable sublanguage must be taken. In terms of languages, $K_{1,a} = (\overline{K_{spec1,a}} \cap P_1(L'_{m,1}))^{\uparrow C}$. The automaton that generates this language, $H_{1,a}$, represents the first modular supervisor and consists of 6 states and 6 transitions.

6) **lift abstracted supervisors to the global level**
In order to apply the supervisors generated in step 5 to the global plant, a default control is implemented that enables all events that had been previously projected away. This is accomplished by the inverse projection operation and is represented by $\widetilde{\overline{K_i}} = P_i^{-1}(\overline{K_{i,a}})$. For our example no events have been abstracted away yet. In the proofs to follow, the events abstracted away in step 1 are added in separate of the other events, however, in terms of actual implementation all events can be added in at the same time.

7) **move up to the next level of the hierarchy**
In order to address those specifications for which a modular supervisor has not yet been built, we now move up to the next level of the hierarchy and repeat steps 2-6. The individual automata representing the closed loop behavior of each subgrouping ($H_{1,a}$, $H_{2,a}$, $\ldots$, $H_{k,a}$) from the previous level become subplants on this level. For example, the allowable language $K_{k+1,a}$ for specification $\overline{K_{spec,k+1}}$ will be designed with respect to a "plant" G'_{k+1} that is the synchronous composition of some $H_{i,a}$ from the first level as well as some subplants $G_{j,a}$ that were not employed in the first level.

$$\begin{aligned} G'_{k+1} &= H_{i_1,a} \| \cdots \| H_{i_u,a} \| G''_{k+1} \\ &\quad \{i_1, \ldots, i_u\} \subseteq \{1, \ldots, k\} \end{aligned}$$

where

$$\begin{aligned} G''_{k+1} &= G_{j_1,a} \| \cdots \| G_{j_v,a} \\ &\quad \{j_1, \ldots, j_v\} \subseteq \{\text{s+1}, \ldots, n\} \end{aligned}$$

For our manufacturing example, we move up a level of hierarchy to build a supervisor for the second specification, B_7. Our new plant consists of the remaining machines, $Robot$ and AM, as well as the allowable language from the first level generated by $H_{1,a}$. Hence the plant for the second level, G'_2, is generated by $H_{1,a} \| Robot \| AM$. At this point the events s_p, f_p, f_{fc}, and s_{bc} are no longer relevant since they are not addressed by the second specification. Since their erasure does not cause a loss of the L_m-observer or consistency of marking properties, they can now be projected away. As a result, $G'_{2,a}$, is reduced to an automaton consisting of 16 states and 52 transitions. Following step 5, the closed-loop automaton is built as $B_7 \| G'_{2,a}$ and again fails to be controllable. Therefore, taking the supremal controllable sublanguage, the new allowable language for the second specification is $K_{2,a} = (\overline{K_{spec2,a}} \cap P_2(L'_{m,2}))^{\uparrow C}$. The automaton generator for this allowable language, $H_{2,a}$, is made up of 20 states and 37 transitions. The final step again is to lift this abstraction up to the global level.

8) **repeat until finished**
Continue to repeat this process until there are no more specifications left. $\diamond$

For the purposes of comparison, the monolithic solution of our manufacturing example generates a supervisor whose automaton representation consists of 36 states and 69 transitions, while the two modular supervisors are represented by $6+20=26$ states and $6+37=43$ transitions. Even though the highest level supervisor is built in essence with respect to the full plant, it will almost always be significantly smaller than the monolithic supervisor since it is built to address only those strings relevant to the last specification, rather than the conjunction of all the specifications.

It is interesting to note that if the modular supervisor for B_7 had been built first and the supervisor for B_8 been built second, the modular supervisors would have had $32+24=56$ states and $73+45=118$ transitions. Some heuristic guidelines for how to choose which specifications to address first include, try to keep the lower level supervisors small while limiting the overall number of levels in the hierarchy, and try and maximize the amount of abstraction that can take place with the lower level supervisors.

In this case, it turns out that the behavior allowed by the two modular supervisors is identical to the behavior allowed by the single monolithic supervisor and hence is optimal. This will not be the case in general since each of the modules were chosen to be optimal with respect to a subset of the full plant and hence may not be optimal in the global sense.

IV. INCREMENTAL HIERARCHICAL ANALYSIS

At this point, we will consider the modular supervisors constructed by the algorithm of the previous section without the use of any abstraction. Specifically, it will be shown that the conjunction of the supervisors will be controllable and nonblocking based on the fact that the supervisors are nonconflicting by construction. In the next section these arguments will be reconsidered with the addition of abstraction.

The following propositions will be employed in the proceeding proof. The logic here follows closely the work of [12], with the primary contribution being the additional discussion of non-prefix closed languages and nonblocking (propositions 5 and 6, in particular).

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

The above proposition is useful in showing that if an allowable language is controllable with respect to a subset of plant subsystems, it is controllable with respect to the full plant.

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

In other words, the intersection of two nonconflicting controllable languages is itself controllable.

Proposition 4: Let K_1, K_2, $L \subseteq \Sigma^*$ be languages and let $K = K_1 \cap K_2$. Also let $\Sigma' \subseteq \Sigma$ and K_1 and K_2 be nonconflicting. If K_2 is $\Sigma'-$controllable with respect to $K_1 \cap L$, and K_1 is $\Sigma'-$controllable with respect to L, then K is $\Sigma'-$controllable with respect to L.

The above proposition is useful in our application since we are building each successive level of allowable languages, K_i, with respect to an uncontrolled plant intersected with a set of allowable languages from the previous level.

Proposition 5: Let K_1, K_2, L_{m1}, L_{m2} be languages and let $K = K_1 \cap K_2$ and $L_m = L_{m1} \cap L_{m2}$. If K_1 and K_2 are nonconflicting, and K_1 is $L_{m1}-$closed and K_2 is $L_{m2}-$closed, then K is L_m-closed.

The above proposition is useful in that if each modular supervisor is nonblocking with respect to its portion of the plant, then the conjunction of the supervisors will be nonblocking with respect to the conjunction of the associated subplants.

Proposition 6: Let K_1, K_2, L_{m1}, L_{m2} be languages and let $K_2 \subseteq K_1$ and $L_m = L_{m1} \cap L_{m2}$. If K_2 is closed with respect to $K_1 \cap L_{m2}$ and K_1 is $L_{m1}-$closed, then K_2 is L_m-closed.

This proposition demonstrates that as we move up levels of the hierarchy, each successive supervisor will be nonblocking with respect to a larger and larger portion of the plant. These given propositions will help us show our first main result.

Main Result I: *The set of modular supervisors constructed by the procedure of Section III, with steps 1 and 4 omitted, will meet the given specifications in a nonblocking manner when acting in conjunction if such a set of nonblocking modular supervisors exists.*

Proof:

- Beginning on the first level of the hierarchy, each allowable language is built with respect to a subset of the full uncontrolled plant:

$$K_i = (\overline{K_{spec,i}} \cap L'_{m,i})^{\uparrow C} \tag{5}$$

  Since we are on the first level, $L'_i = L''_i$, that is, the allowable language is built with respect to a portion of the plant without any control included. By construction, the allowable language is Σ_u-controllable with respect to L''_i and is $L''_{m,i}-$closed, therefore a nonblocking supervisor is guaranteed to exist for that portion of the plant. Since the language is Σ_u-controllable with respect to $L''_i \supseteq L$, it is also Σ_u-controllable with respect to the full L by proposition 2.
- Furthermore, since each K_i on a given level is "disjoint" and Σ_u-controllable with respect to L, their conjunction, $\bigcap K_i$, is also Σ_u-controllable with respect to L by proposition 3. Also, the conjunction of supervisors

on the first level, $\bigcap K_i$, will be nonblocking when supervising the portion of the uncontrolled plant given by $\bigcap L''_{m,i}$ as demonstrated by proposition 5.

- Moving up a level, each new allowable language is constructed in a similar manner:

$$K_{k+1} = (\overline{K_{spec,k+1}} \cap L'_{m,k+1})^{\uparrow C} \tag{6}$$

except now, $L'_{m,k+1}$ consists of a plant subsystem as well as a subset of the controlled plant from the first level.

$$L'_{m,k+1} = \bigcap_{i=i_1}^{i_u} K_i \cap L''_{m,k+1} \tag{7}$$

Building the allowable language with respect to an $L'_{m,k+1}$ constructed in this manner will prove useful since it results in each new allowable language, K_{k+1}, being a subset of the allowable languages from the first level, and hence languages on different levels won't conflict either. This fact is demonstrated below.

$$\begin{aligned} K_{k+1} &= (\overline{K_{spec,k+1}} \cap L'_{m,k+1})^{\uparrow C} \\ &\subseteq \overline{K_{spec,k+1}} \cap L'_{m,k+1} \\ &= \overline{K_{spec,k+1}} \bigcap_{i=i_1}^{i_u} K_i \cap L''_{m,k+1} \\ &\subseteq K_i \end{aligned} \tag{8}$$

Also note that $K_{k+1} \subseteq K_i$ implies $\overline{K_{k+1}} \subseteq \overline{K_i}$.
Since K_{k+1} is Σ_u-controllable with respect to $\bigcap K_i \cap L''_{k+1}$, it is also Σ_u-controllable with respect to $\bigcap K_i \cap L$ by proposition 2. Furthermore, since $\bigcap K_i$ is Σ_u-controllable with respect to L, $\bigcap K_i \cap K_{k+1}$ is Σ_u-controllable with respect to L by proposition 4. This implies that the conjunction of these supervisors will provide controllability.
Also by construction, the new K_{k+1} is $L'_{m,k+1}-$closed. It was already shown that within any given level the conjunction of supervisors will be nonblocking when supervising that portion of the uncontrolled plant. Nonblocking of the conjunction of supervisors from different levels follows from the fact that $K_{k+1} \subseteq K_i$ and from proposition 6. Specifically, K_{k+1} is closed with respect to $\bigcap L''_{m,i} \cap L''_{m,k+1}$. For the ($k$+1)th allowable language, we will denote the conjunction of the subset of languages from the previous level, $\bigcap L''_{m,i}$, as $L''_{m,K}$. Since the allowable language is Σ_u-controllable and closed with respect to that portion of the plant, a nonblocking supervisor is again guaranteed to exist for that subsystem.

- This logic continues until a supervisor has been constructed for each specification. Each allowable language, K_i, is Σ_u-controllable with respect to the full uncontrolled plant, L, and closed with respect to a portion of the uncontrolled plant, $L''_{m,I-1} \cap L''_{m,i}$. Furthermore, each pair of allowable languages are either "disjoint" or subsets of one another and hence are nonconflicting. Therefore the conjunction of all allowable languages, $\mathbf{K} = \cap K_i$, is Σ_u-controllable with respect to L by proposition 3, and L_m-closed as shown below.

$$\begin{aligned} \mathbf{K} &= \bigcap_{i=1}^{m} K_i \\ &= \bigcap_{I=2,i=1}^{m} (\overline{K_i} \cap L''_{m,I-1} \cap L''_{m,i}) \\ &= \overline{K_1 \cap \cdots \cap K_m} \bigcap_{I=2,i=1}^{m} L''_{m,I-1} \cap L''_{m,i} \\ &= \overline{\mathbf{K}} \cap L_m \end{aligned} \tag{9}$$

- Therefore the conjunction of modular supervisors meets the overall specification and is nonblocking. $\diamond$

While the above proof doesn't guarantee the modular supervisors will be optimal in the sense of being least restrictive, it does guarantee that they will satisfy the specifications in a nonblocking manner, which is our primary goal.

The fact that, in essence, we had to build the full system defeats one of the primary advantages sought from a modular approach to supervisory control. In order to make this approach feasible, it is proposed that abstractions of the system take place as we move up each level of the hierarchy. The details of this concept are examined in the next section.

V. SYSTEM ABSTRACTION

We will now additionally consider abstraction. In the context of our approach, we will abstract away those elements of our system that are not relevant to the specifications being addressed at the current or higher levels of our hierarchy. For our abstraction, we will simply apply the natural projection operation defined earlier by (2), where it is further required that the projections maintain the L_m-observer and consistency of marking properties.

In order for the supervisor corresponding to this abstracted language to be applied to the global plant, it is necessary to incorporate a default control that permanently enables all events that had been previously hidden. From an implementation standpoint, this can be achieved by the inverse projection, $\widetilde{K}_i = P_i^{-1}(K_{i,a})$. In the proofs to follow, the inverse projection adds in all previously erased events except those abstracted away in step 1 of the procedure of section III. These strictly private events will be addressed separately at the very end of this section. Therefore, when speaking of the "global" plant L_a, what will actually be considered is the language with these strictly private events projected away. Mathematically, this lifted modular supervisor will likely allow strings that are outside the uncontrolled system behavior L, and are outside the behavior allowed by the conjunction of the other modular supervisors. This, however, is of no consequence since these strings won't occur anyway. In the definitions given below, the restriction to $\overline{\widetilde{K}_{I-1}} \cap L''_i$ is added to assist with the proofs that are to follow, but in practice won't have to be implemented as such. Note that $\overline{\widetilde{K}_{I-1}}$ corresponds to the conjunction of supervisors that directly precede the ith supervisor in the hierarchy. In

general, all languages with the subscript $(I-1)$ consist of a conjunction of corresponding sets of lower-level languages.

$$\begin{aligned}\overline{\widetilde{K}_i} &= P_i^{-1}(\overline{K_{i,a}}) \cap \overline{\widetilde{K}_{I-1}} \cap L_i'' \\ \widetilde{K}_i &= P_i^{-1}(K_{i,a}) \cap \widetilde{K}_{I-1} \cap L_i'' \end{aligned} \quad (10)$$

It is the goal of this section to show that the results from the previous section will still be valid when applied to abstracted systems. Specifically, it is necessary to show that if an abstracted admissible language, $K_{i,a}$, is $\Sigma_{u,i}-$controllable with respect to $P_i(L_i')$, then the lifted version, $\widetilde{K}_i$, is Σ_u-controllable with respect to the unabstracted language L_i'. Furthermore, if the abstracted admissible language is nonblocking, we want that to imply that the lifted version is also nonblocking.

From examination of the definition of the lifted languages, one can see the modular supervisors will be either "disjoint" or contained in one another, $\widetilde{K}_i \subseteq \widetilde{K}_{I-1}$, thereby maintaining nonconflict by construction.

Expanding the definitions of $\widetilde{K}_i$ and $\overline{\widetilde{K}_i}$ given in (10) will help us with the proofs to follow.

$$\begin{aligned}\overline{\widetilde{K}_i} &= P_i^{-1}(\overline{K_{i,a}}) \cap P_{I-1}^{-1}(\overline{K_{I-1,a}}) \cap L_{I-1}'' \cap L_i'' \\ \widetilde{K}_i &= P_i^{-1}(K_{i,a}) \cap P_{I-1}^{-1}(K_{I-1,a}) \cap L_{I-1}'' \cap L_i'' \end{aligned} \quad (11)$$

If we further note that each additional plant submodule, L_i'', has no private events that can be abstracted away due to step 1 of the procedure of section III, then the following result holds,

$$\begin{aligned} K_{i,a} &\subseteq P_i(L_{m,i}') = P_i(P_{I-1}^{-1}(K_{I-1,a}) \cap L_{m,i}'') \\ &\subseteq P_i(L_{m,i}'') \\ P_i^{-1}(K_{i,a}) &\subseteq P_i^{-1}(P_i(L_{m,i}'')) = L_{m,i}'' \subseteq L_i'' \end{aligned} \quad (12)$$

which when combined with (11) gives us that

$$\begin{aligned}\overline{\widetilde{K}_i} &= P_i^{-1}(\overline{K_{i,a}}) \cap P_{I-1}^{-1}(\overline{K_{I-1,a}}) \cap L_i'' \\ &= P_i^{-1}(\overline{K_{i,a}}) \cap L_i' \end{aligned} \quad (13)$$

The above definition and following proposition, which follows closely from results in [5], will be used in showing the maintenance of controllability properties between abstracted and lifted languages.

Proposition 7: If K_a is $\Sigma_a'-$controllable with respect to $P(L)$, then $\widetilde{K}$ is $\Sigma'-$controllable with respect to L, where $P(\sigma) \in \Sigma_a',\ \sigma \in \Sigma'$, and $\overline{\widetilde{K}} = P^{-1}(\overline{K_a}) \cap L$.

The above proposition demonstrates that if $K_{i,a}$ is $\Sigma_{u,i}-$controllable with respect to $P_i(L_i')$, then $\widetilde{K}_i$ is Σ_u-controllable with respect to $L_i' = P_{I-1}^{-1}(\overline{K_{I-1,a}}) \cap L_i''$. On the first level of the hierarchy $L_i' = L_i''$ and proposition 2 provides that each $\widetilde{K}_i$ is controllable with respect to the full L_a. Proposition 3 further provides that the conjunction of lifted languages is also controllable with respect to L_a. Moving up to the next level of the hierarchy, $\widetilde{K}_i$ is Σ_u-controllable with respect to $\widetilde{K}_{I-1} \cap L_a$ again by application of propositions 7 and 2. Since it is already known that $\widetilde{K}_{I-1}$ is Σ_u-controllable with respect to L_a, the conjunction $\widetilde{K}_i \cap \widetilde{K}_{I-1}$ is Σ_u-controllable with respect to L_a by proposition 4.

In considering nonblocking, we will see the necessity of the observer and consistency of marking properties introduced earlier. The first desired result of the observer property is that it makes the definitions put forth in (10) consistent, meaning that it provides that the prefix closure of $\widetilde{K}_i$ is equal to $\overline{\widetilde{K}_i}$ as we have defined them. The following proposition goes a long way in helping us to show this result.

Proposition 8: If the projection P possesses the L_m-observer property, then $\overline{P^{-1}(K_a) \cap L_m} = P^{-1}(\overline{K_a}) \cap L$.

Beginning with the result of proposition 8,

$$\overline{P_i^{-1}(K_{i,a}) \cap L_{m,i}'} = P_i^{-1}(\overline{K_{i,a}}) \cap L_i'$$

expanding,

$$\begin{aligned}&\overline{P_i^{-1}(K_{i,a}) \cap P_{I-1}^{-1}(K_{I-1,a}) \cap L_{m,i}''} = \\ &\quad P_i^{-1}(\overline{K_{i,a}}) \cap P_{I-1}^{-1}(\overline{K_{I-1,a}}) \cap L_i''\end{aligned}$$

recalling (12), $P_i^{-1}(K_{i,a}) \subseteq L_{m,i}'' \subseteq L_i''$ and $P_{I-1}^{-1}(K_{I-1,a}) \subseteq P_{I-1}^{-1}(\overline{K_{I-1,a}}) \subseteq L_{I-1}''$,

$$\begin{aligned}&\overline{P_i^{-1}(K_{i,a}) \cap P_{I-1}^{-1}(K_{I-1,a}) \cap L_{I-1}'' \cap L_i''} = \\ &\quad P_i^{-1}(\overline{K_{i,a}}) \cap P_{I-1}^{-1}(\overline{K_{I-1,a}}) \cap L_{I-1}'' \cap L_i''\end{aligned}$$

Comparing the above with (11) completes the proof that the prefix closure of $\widetilde{K}_i$ equals $\overline{\widetilde{K}_i}$.

If we additionally consider the consistency of marking requirement, we can further show that if $K_{i,a}$ is $P(L_{m,i}')-$closed, then the lifted language is closed with respect to the portion of the uncontrolled plant addressed by the current and lower levels of supervisors, $L_{m,I-1}'' \cap L_{m,i}''$. The following proposition will be used with an induction argument to show this result.

Proposition 9: If the projection P possesses the L_m-observer property and consistency of marking, and K_a is $P(L_m)-$closed, then $\widetilde{K} = P^{-1}(K_a) \cap L$ is L_m-closed.

In the first level of our hierarchy, each of the supervised languages correspond to a language of the form, $P_i^{-1}(K_{i,a}) \cap L_{m,i}''$, since $L_{m,i}' = L_{m,i}''$. Therefore, all the first level supervised languages are closed with respect to their portion of the uncontrolled plant by proposition 9 directly. Assuming that each of the languages making up $\widetilde{K}_{I-1}$ are closed with respect to their portion of the uncontrolled plant, $L_{m,I-1}''$, we will show that $\widetilde{K}_i$ is closed with respect to $L_{m,i}'' \cap L_{m,I-1}''$. Beginning with the results of proposition 9:

$$(P_i^{-1}(\overline{K_{i,a}}) \cap L_i') \cap L_{m,i}' = P_i^{-1}(K_{i,a}) \cap L_i'$$

expanding,

$$\begin{aligned}&(P_i^{-1}(\overline{K_{i,a}}) \cap P_{I-1}^{-1}(\overline{K_{I-1,a}}) \cap L_i'') \cap L_{m,i}' \\ &\quad = P_i^{-1}(K_{i,a}) \cap P_{I-1}^{-1}(\overline{K_{I-1,a}}) \cap L_i''\end{aligned}$$

intersecting both sides of the equation with L''_{I-1} and $P^{-1}_{I-1}(K_{I-1,a})$ and noting that $L'_{m,i} = P^{-1}_{I-1}(K_{I-1,a}) \cap L''_{m,i} \subseteq P^{-1}_{I-1}(K_{I-1,a}) \subseteq P^{-1}_{I-1}(\overline{K_{I-1,a}})$,

$$\begin{aligned}(P_i^{-1}(\overline{K_{i,a}}) \cap P^{-1}_{I-1}(\overline{K_{I-1,a}}) \cap L''_{I-1} \cap L''_i) \cap L'_{m,i} \\ = P_i^{-1}(K_{i,a}) \cap P^{-1}_{I-1}(K_{I-1,a}) \cap L''_{I-1} \cap L''_i\end{aligned}$$

now comparing to (11),

$$\begin{aligned}\overline{\widetilde{K}_i} \cap L'_{m,i} &= \widetilde{K}_i \\ \overline{\widetilde{K}_i} \cap \widetilde{K}_{I-1} \cap L''_{m,i} &= \widetilde{K}_i\end{aligned}$$

applying the induction hypothesis,

$$\begin{aligned}\overline{\widetilde{K}_i} \cap (\overline{\widetilde{K}_{I-1}} \cap L'_{m,I-1}) \cap L''_{m,i} &= \widetilde{K}_i \\ \overline{\widetilde{K}_i} \cap L'_{m,I-1} \cap L''_{m,i} &= \widetilde{K}_i\end{aligned}$$

continuing to expand the terms making up $L'_{m,I-1}$,

$$\overline{\widetilde{K}_i} \cap L''_{m,I-1} \cap L''_{m,i} = \widetilde{K}_i$$

Hence, each lifted language, $\widetilde{K}_i$, is closed with respect to $\cap L''_{m,I-1} \cap L''_{m,i}$. Following the logic of (9), one can see the conjunction of lifted languages will be closed with respect to the full plant $L_{m,a}$.

So far we have shown that the conjunction of our supervisors is controllable and closed with respect to the language L_a, which is the projection of the global uncontrolled behavior of our system. The only step left is to address those strictly private events abstracted away by the first step of our procedure. Since the projection distributes over the synchronous composition when no shared events are erased [11], the composition of the individual abstracted subplants, $G_{i,a}$, is equivalent to L_a. Results of [11] also show the projection used to achieve this L_a possesses the L_m-observer property. Lifting the modular supervisors generated so far to include the strictly private events results in supervisors whose conjunction is controllable and closed with respect to the global unabstracted plant as demonstrated by a final application of proposition 7, proposition 8 and proposition 9. Therefore we are left with our final and most important result.

Main Result II: *The set of modular supervisors constructed by the procedure of Section III will meet the given specifications in a nonblocking manner when acting in conjunction if such a set of nonblocking modular supervisors exists.*

One problem is that in general obtaining abstractions with the observer property can be difficult, especially if the state space of the system of interest is large. The complexity added by testing and/or achieving the observer property is something that needs to be considered. The work of [13] and [14] address some of these issues with respect to general abstractions in the former and with respect to natural projection in the latter.

VI. CONCLUSIONS AND FUTURE WORK

This paper puts forth an incremental procedure for generating a set of modular supervisors that are nonconflicting by construction. The conjunction of the modular supervisors was shown to satisfy given specifications without blocking, though the resulting behavior is not guaranteed to be optimal.

An example was presented for which two supervisors were generated consisting of a total of 26 states and 43 transitions. This showed a significant improvement over the monolithic solution, which required 36 states and 69 transitions. This approach to supervisor design is especially useful for systems where the coupling between elements is well-distributed. If every specification of a system addresses every plant module, then this approach will provide little if any improvement.

Some directions for future work include determining some further heuristics for choosing the ordering in which specifications are addressed. Work also can be done to determine the exact conditions under which this modular approach is optimal. It is also desirable to better understand how to verify and/or achieve the observer property in a computationally efficient manner.

REFERENCES

[1] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proc. of IEEE, Special Issue on Discrete Event Dynamic Systems*, vol. 77, no. 1, pp. 81–98, January 1989.

[2] H. Zhong and W. Wonham, "On the consistency of hierarchical supervision in discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 35, no. 10, pp. 1125–1134, October 1990.

[3] K. Wong and W. Wonham, "Hierarchical control of discrete-event systems," *Discrete Event Dynamic Systems: Theory and Application*, vol. 6, pp. 241–273, 1996.

[4] P. Ramadge and W. Wonham, "Modular supervisory control of discrete event systems," *Mathematics of Control, Signal and Systems*, vol. 1, no. 1, pp. 13–30, 1988.

[5] F. Lin and W. Wonham, "Decentralized supervisory control of discrete-event systems," *Information Sciences*, no. 44, pp. 199–224, 1988.

[6] K. Wong and W. Wonham, "Modular control and coordination of discrete-event systems," *Discrete Event Dynamic Systems: Theory and Application*, vol. 8, pp. 247–297, 1998.

[7] M. H. de Queiroz and J. E. R. Cury, "Modular supervisory control of composed systems," in *Proceedings of the American Control Conference*, Chicago, June 2000, pp. 4051–4055.

[8] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Boston, MA: Kluwer Academic Publishers, 1999.

[9] K. Wong, J. Thistle, H.-H. Hoang, and R. Malhamé, "Conflict resolution in modular control with applications to feature interaction," in *Proceedings of the IEEE Conference on Decision & Control*, New Orleans, LA, Dec. 1995, pp. 416–421.

[10] K. Wong, "On the complexity of projections of discrete event systems," in *Proceedings of the IFAC Workshop on Discrete Event Systems (WODES)*, Cagliari, Italy, Aug. 1998, pp. 201–206.

[11] P. Pena, J. Cury, and S. Lafortune, "Testing modularity of local supervisors: An approach based on abstractions," in *Proceedings of the IFAC Workshop on Discrete Event Systems (WODES)*, Ann Arbor, Michigan USA, 2006.

[12] B. A. Brandin, R. Malik, and P. Malik, "Incremental verification and synthesis of discrete-event systems guided by counter examples," *IEEE Transactions on Control Systems Technology*, vol. 12, no. 3, pp. 387–401, May 2004.

[13] K. Wong and W. Wonham, "On the computation of observers in discrete-event systems," *Discrete Event Dynamic Systems: Theory and Application*, vol. 14, pp. 55–107, 2004.

[14] L. Feng, "On the computation of natural observers in discrete-event systems," University of Toronto, Systems and Control Group, Toronto, Canada, Tech. Rep., Jan. 2006.