Multi-Level Hierarchical Interface-Based Supervisory Control ⋆

R.C. Hill [a], J.E.R. Cury [c], M.H. de Queiroz [c], D.M. Tilbury [b], S. Lafortune [b],

[a] *University of Detroit Mercy, Detroit, MI 48221-3038, USA*

[b] *University of Michigan, Ann Arbor, MI 48109-2125, USA*

[c] *Federal University of Santa Catarina, Florianópolis, SC 88040-900, Brazil*

Abstract

Hierarchical Interface-Based Supervisory Control employs interfaces that allow properties of a monolithic system to be verified through local analysis. By avoiding the need to verify properties globally, significant computational savings can be achieved. In this paper we provide local requirements for a multi-level architecture employing command-pair type interfaces. This multi-level architecture allows for a greater reduction in complexity and improved reconfigurability over the two-level case that has been previously studied since it allows the global system to be partitioned into smaller modules. This paper also provides results for synthesizing supervisors in the multi-level architecture that are locally maximally permissive with respect to a given specification and set of interfaces.

Key words: Discrete-event systems; supervisory control; hierarchical control.

1 Introduction

In recent years, a well-formed body of theory has been developed with regard to the control of discrete-event systems (DES). Application of this theory has been hindered by the well-known *state-space explosion* problem. One approach to address this problem is to introduce interfaces between various components of a larger system [8] [17] [19] [20]. The purpose of these interfaces is to limit the interaction of various components in such a way that global properties can be verified locally. By avoiding analysis of the global system, state-space explosion can often be avoided. This architecture also provides for improved reconfigurability since a system component can be modified without having to re-analyze the entire global system. The increased restrictiveness of the interfaces can result in suboptimal control. In many cases, this reduction in computational complexity and improved reconfigurability in exchange for optimality may be desirable.

Considering existing research on interface-based control, the results provided by [8] do not address nonblocking. In this paper we refer to nonblocking in the standard supervisory control sense where deadlock and livelock may be considered through the marking of the system. The work of [17] [19] [20] addresses the properties of controllability and nonblocking, but this treatment is limited to two levels of modules. Other works that exist for reducing the complexity associated with synthesizing global nonblocking control rely on incremental construction and abstraction [9] [11] [14] [23]. Still other research employs similar approaches, but for verification [1] [10] [21]. These works are quite useful, but they do not strictly rely on local analysis and design; rather, their techniques are in essence applied to incrementally constructed abstractions of the global system. As such, these techniques are not very reconfigurable and can still suffer from state-space explosion. The work of [5] [6] employs interface automata for software design, but does not explicitly address the problem of supervisory control.

This paper introduces a new set of local conditions that guarantee global controllability and nonblocking of a

⋆ This work was supported in part by NSF grant CMS-05-28287 and EECS-06-24821. This paper was not presented at any IFAC meeting. Corresponding author R. C. Hill. Tel. +1 313 5780428.

Email addresses: hillrc@udmercy.edu (R.C. Hill), cury@das.ufsc.br (J.E.R. Cury), max@das.ufsc.br (M.H. de Queiroz), tilbury@umich.edu (D.M. Tilbury), stephane@umich.edu (S. Lafortune).

multi-level system with command-pair interfaces. An example multi-level interface system is pictured in Fig. 1 where each closed-loop component $\mathbf{H}_k^i$ interacts with its neighboring modules through interfaces $\mathbf{I}_k^i$. This generalized architecture offers significant advantage over the two-level architecture of [17] [19] [20] since in many instances it allows the system to be partitioned into smaller modules, further limiting the complexity of analysis and design. This multi-level architecture, therefore, can greatly increase the size of systems that can be addressed by an interface-based approach to control.

Another contribution of this paper is the relaxation of the interface consistency requirements of [17] [19] [20] which allows a system to be modeled more compactly in some instances, though at the possible expense of additional complications. Finally, the paper builds on [18] to develop methods for synthesizing modular supervisors in the multi-level architecture that are locally maximally permissive with respect to the interface-based requirements and a given specification and set of interfaces, though the overall resulting control may not be globally maximally permissive.

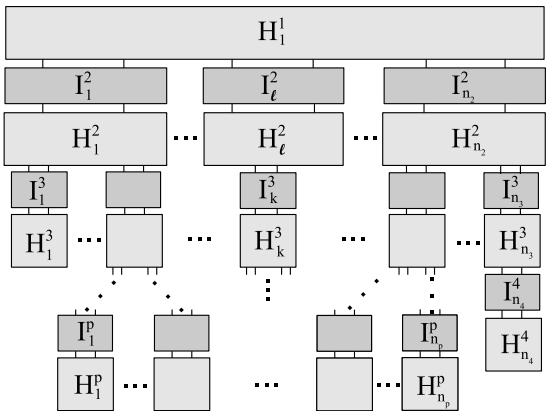

Fig. 1. Illustration of the multi-level architecture

The organization of the remainder of this paper is as follows. Section 2 introduces some preliminary notation and definitions, Section 3 presents the local requirements for the multi-level hierarchical interface-based approach to supervisory control, and Section 4 demonstrates that the global properties of nonblocking and controllability can be verified based on these local requirements. Section 5 outlines an approach to supervisor synthesis in the multi-level architecture. Section 6 demonstrates the application of this architecture to a manufacturing example, while Section 7 concludes the paper with a summary of its contributions.

2 Preliminaries

We will consider DES modeled by automata that are represented by the five-tuple $\mathbf{G} = (Q, \Sigma_G, \delta, q_0, Q_m)$, where Q is the set of states, $\Sigma_G \subseteq \Sigma$ is the set of events over which $\mathbf{G}$ is defined and Σ is the global alphabet, $\delta : Q \times \Sigma_G \to Q$ is the partial state transition function, $q_0 \in Q$ is the initial state, and $Q_m \subseteq Q$ 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 $\delta : Q \times \Sigma_G^* \to Q$ in the natural way. The notation $\delta(q, s)!$ for any $q \in Q$ and any $s \in \Sigma_G^*$ denotes that $\delta(q, s)$ is defined. The notation $\Sigma(\mathbf{G}) \subseteq \Sigma_G$ will be employed to denote the *relevant* event set of the automaton $\mathbf{G}$. By relevant, it is meant all events over which $\mathbf{G}$ is defined that are not self-looped at every state.

The *generated* and *marked languages* of $\mathbf{G}$, denoted by $\mathcal{L}(\mathbf{G})$ and $\mathcal{L}_m(\mathbf{G})$ respectively, are defined by $\mathcal{L}(\mathbf{G}) = \{s \in \Sigma_G^* \mid \delta(q_0, s)!\}$ and $\mathcal{L}_m(\mathbf{G}) = \{s \in \Sigma_G^* \mid \delta(q_0, s) \in Q_m\}$. 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. The following eligibility operator will be employed to denote which events in the set Σ are enabled in the language L following the occurrence of a string $s \in \Sigma^*$, $Elig_L(s) := \{\sigma \in \Sigma \mid s\sigma \in L\}$. An event $\sigma \in \Sigma$ is defined to be irrelevant to the language $L \subseteq \Sigma^*$, if for all $s, t \in \Sigma^*$ $st \in L$ if and only if $s\sigma t \in L$, otherwise σ is relevant to L [1].

An automaton is said to be *nonblocking* when from all of its reachable states, a marked state can be reached. From a language point of view, this is defined as $\overline{\mathcal{L}_m(\mathbf{G})} = \mathcal{L}(\mathbf{G})$. If an automaton enters a state from which it cannot reach a marked state, the automaton is said to have *blocked*.

The operation of two automata $\mathbf{G}_1$ and $\mathbf{G}_2$ together is captured via the *synchronous composition* (parallel composition) operator, $\|$. 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 [3].

We will employ the *natural projection* $P_i : \Sigma^* \to \Sigma_i^*$. Given a string $s \in \Sigma^*$, the natural projection P_i erases those events in the string that are in the global alphabet Σ, but not in the local alphabet Σ_i, where $\Sigma_i \subseteq \Sigma$.

$$P_i(\varepsilon) := \varepsilon \qquad P_i(e) := \begin{cases} e, & e \in \Sigma_i \subseteq \Sigma \\ \varepsilon, & e \notin \Sigma_i \subseteq \Sigma \end{cases}$$

$$P_i(se) := P_i(s)P_i(e), s \in \Sigma^*, e \in \Sigma$$

We can also define the inverse natural projection for a string $t \in \Sigma_i^*$ as follows, $P_i^{-1}(t) := \{s \in \Sigma^* : P_i(s) = t\}$. These definitions can naturally be extended to languages.

In supervisory control [22], 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 by a supervisory controller, while uncontrollable events cannot. The notation $\dot{\cup}$ represents the union of disjoint sets. A supervisor, denoted by $\mathcal{S}$, is a mapping that outputs a list of events to be disabled based on the observation of events generated by a plant $\mathbf{G}$. Keeping in mind that uncontrollable events are not allowed to be disabled, a supervisor $\mathcal{S} : \mathcal{L}(\mathbf{G}) \rightarrow 2^{\Sigma}$ can be represented by an automaton $\mathbf{S}$ such that the closed-loop system behavior $\mathcal{S}/\mathbf{G} = \mathbf{S} \| \mathbf{G}$.

To ensure that a given automaton $\mathbf{S}$ with alphabet Σ represents a supervisor $\mathcal{S}$ that restricts the plant $\mathbf{G}$ to a subset of the behavior of $\mathbf{S}$, it is necessary that the following Σ_u-*controllability* condition be satisfied, where $\Sigma_u \subseteq \Sigma$. The following expression can be interpreted as providing that the language $\mathcal{L}(\mathbf{S})$ is Σ_u-controllable with respect to $\mathcal{L}(\mathbf{G})$, where the two languages are defined over the same set of events.

$$(\forall s \in \mathcal{L}(\mathbf{S}) \cap \mathcal{L}(\mathbf{G})),\ \mathit{Elig}_{\mathcal{L}(\mathbf{G})}(s) \cap \Sigma_u \subseteq \mathit{Elig}_{\mathcal{L}(\mathbf{S})}(s)$$

Some interface-based requirements in this paper also rely on a controllability-type condition with respect to event sets other than Σ_u.

3 Modular Requirements

We will now define the notation and requirements necessary for proving results with regard to a multi-level application of hierarchical interface-based supervisory control. We specifically assume a restricted type of hierarchy consistent with a connected tree architecture with a single root node. Figure 1 illustrates this situation. Our component-wise specified system is split up into modules, each consisting of a plant $\mathbf{G}_k^i$ and a supervisor $\mathbf{S}_k^i$ constructed with respect to a local specification $\mathbf{E}_k^i$ resulting in the closed-loop subsystem $\mathbf{H}_k^i = \mathbf{G}_k^i \| \mathbf{S}_k^i$. The superscript i reflects the level of the hierarchy and takes values $\{1, \ldots, p\}$. The subscript k indicates the index within a given level and takes the values $\{1, \ldots, n_i\}$, where this set represents all modules and interfaces on a given level i, including modules and interfaces that have different corresponding higher-level neighbors. For each closed-loop subsystem $\mathbf{H}_k^i$, the set of indices of the interfaces and corresponding modules directly below it in the connected tree architecture will be identified by the notation J_k^i. The sets J_k^i for modules on the i^{th} level partition the set of indices $\{1, \ldots, n_{i+1}\}$ into disjoint subsets. While the open-loop components are not necessarily nonblocking, the component supervisors will be constructed such that the closed-loop modules are nonblocking.

All interaction between modules takes place through corresponding interfaces $\mathbf{I}_k^i$. These interfaces restrict the behavior of the overall system in such a way that global properties can be guaranteed by local analysis. In a sense, these interfaces may apply additional control. Figure 2 shows a detail of the multi-level architecture. We will refer to the global system defined in terms of these modules and interfaces as the system Φ.

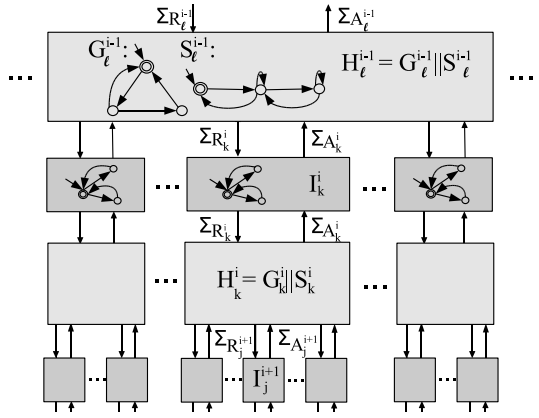

Fig. 2. Detail of the multi-level architecture

In this architecture, all events shared between a given module $\mathbf{H}_k^i$ and its higher-level neighbor are classified as either request events $\rho \in \Sigma_{R_k^i}$ or answer events $\alpha \in \Sigma_{A_k^i}$. The occurrence of each of these events must then be accepted by the corresponding interface $\mathbf{I}_k^i$. Conceptually, request events are thought of as being under the control of the higher-level module and answer events as being under the control of the lower-level module. For the purposes of this paper, we will assume the interfaces take the form of a *command-pair interface* defined below in the manner of [17].

Definition 1 *A DES* $\mathbf{I} = (X, \Sigma_A \dot{\cup} \Sigma_R, \xi, x_0, X_m)$ *is a* command-pair interface *if the following are true:*
A) $\mathcal{L}(\mathbf{I}) \subseteq \overline{(\Sigma_R.\Sigma_A)^*}$
B) $\mathcal{L}_m(\mathbf{I}) = (\Sigma_R.\Sigma_A)^* \cap \mathcal{L}(\mathbf{I})$

From the above definition it can be deduced that the event set for the interface $\mathbf{I}_k^i$ is given by $\Sigma_{I_k^i} := \Sigma_{A_k^i} \dot{\cup} \Sigma_{R_k^i}$. The event set of a given module $\mathbf{H}_k^i$ is first defined to be equal to the union of the event sets of the associated plant component and specification and the event sets of the neighboring interfaces, $\Sigma_{H_k^i} := \Sigma_{G_k^i} \cup \Sigma_{E_k^i} \cup \Sigma_{I_k^i} \cup \bigcup_{j \in J_k^i} \Sigma_{I_j^{i+1}}$. We then extend the event sets of $\mathbf{G}_k^i$, $\mathbf{E}_k^i$, and $\mathbf{S}_k^i$ so that they are equal to to the event set of $\mathbf{H}_k^i$, therefore, from this point on it is assumed that $\Sigma_{G_k^i} = \Sigma_{E_k^i} = \Sigma_{S_k^i} = \Sigma_{H_k^i}$. We also assume that the global alphabet is partitioned as shown in (1), where the set Σ_k^i represents those events that are in the event set of $\mathbf{H}_k^i$, but not in the event set of any other

modules, that is, $\Sigma^i_k \cap \Sigma_{H^{i'}_{k'}} = \emptyset$, $\forall((i \neq i') \vee (k \neq k'))$. The following also assumes there is only a single module on level 1, the top level.

$$\Sigma := \Sigma^1_1 \dot\cup \left(\dot\bigcup_{i=2,\ldots,p} \left(\dot\bigcup_{k=1,\ldots,n_i} \left(\Sigma^i_k \dot\cup \Sigma_{A^i_k} \dot\cup \Sigma_{R^i_k} \right) \right) \right) \tag{1}$$

A consequence of (1) is that each interface is completely disjoint from all other interfaces, that is, $\Sigma_{I^i_k} \cap \Sigma_{I^{i'}_{k'}} = \emptyset$, $\forall((i \neq i') \vee (k \neq k'))$. Furthermore, the sets of request $\Sigma_{R^i_k}$ and answer events $\Sigma_{A^i_k}$ are also disjoint from one another. We further assume that the event set of each module $\mathbf{H}^i_k$ is constrained to have the partitioning given in (2).

$$\Sigma_{H^i_k} = \Sigma^i_k \dot\cup \Sigma_{I^i_k} \dot\cup \left(\dot\bigcup_{j \in J^i_k} \Sigma_{I^{i+1}_j} \right) \tag{2}$$

The above equation is consistent with the connected tree architecture of our approach, that is, each module $\mathbf{H}^i_k$ may share events only with modules from the $(i+1)^{th}$ level (through $\mathbf{I}^{i+1}_j$ where $j \in J^i_k$) and a single module from the $(i-1)^{th}$ level (through $\mathbf{I}^i_k$). We will employ script letters to represent the languages generated by the corresponding automata lifted to the global alphabet Σ. This convention is employed in the following definitions.

$$\begin{array}{ll} P_{H^i_k} : \Sigma^* \to \Sigma^*_{H^i_k}, & P_{I^i_k} : \Sigma^* \to \Sigma^*_{I^i_k} \\ \mathcal{H}^i_k := P^{-1}_{H^i_k}(\mathcal{L}(\mathbf{H}^i_k)), & \mathcal{H}^i_{m_k} := P^{-1}_{H^i_k}(\mathcal{L}_m(\mathbf{H}^i_k)) \\ \mathcal{G}^i_k := P^{-1}_{H^i_k}(\mathcal{L}(\mathbf{G}^i_k)), & \mathcal{G}^i_{m_k} := P^{-1}_{H^i_k}(\mathcal{L}_m(\mathbf{G}^i_k)) \\ \mathcal{E}^i_k := P^{-1}_{H^i_k}(\mathcal{L}(\mathbf{E}^i_k)), & \mathcal{E}^i_{m_k} := P^{-1}_{H^i_k}(\mathcal{L}_m(\mathbf{E}^i_k)) \\ \mathcal{S}^i_k := P^{-1}_{H^i_k}(\mathcal{L}(\mathbf{S}^i_k)), & \mathcal{S}^i_{m_k} := P^{-1}_{H^i_k}(\mathcal{L}_m(\mathbf{S}^i_k)) \\ \mathcal{I}^i_k := P^{-1}_{I^i_k}(\mathcal{L}(\mathbf{I}^i_k)), & \mathcal{I}^i_{m_k} := P^{-1}_{I^i_k}(\mathcal{L}_m(\mathbf{I}^i_k)) \end{array}$$

The following requirements modified from [19] will be employed to guarantee global properties through local analysis for a given set of DES. Specifically, the properties will be checked with respect to each $(i,k)^{th}$ module and those interfaces with which it shares events. Here the $(i,k)^{th}$ module is defined to be the plant and supervisor that make up $\mathbf{H}^i_k$. Refer again to Fig. 2 to help visualize the structure of a single module. In the following, the event set $\Sigma_{L^i_k} = \Sigma_{H^i_k} - \Sigma_{I^i_k}$ consists of those events that are in the event set of the module $\mathbf{H}^i_k$ that are not in the event set of the neighboring module on the next higher level of the hierarchy. In the following definitions and in (1) and (2) given above, we will let $\Sigma_{I^1_1} = \emptyset$ and $\mathcal{I}^1_1 = \mathcal{I}^1_{m_1} = \Sigma^*$ since there is no interface above the root module $\mathbf{H}^1_1$. This provides that Points 4 and 5 of Definition 4 do not need to be verified for level 1 since $\Sigma_{A^1_1} = \Sigma_{R^1_1} = \emptyset$. Point 6 also does not need to be verified for level 1 since it does not apply to $i = 1$. When there are no interfaces below a module, $J^i_k = \emptyset$; therefore, Point 3 of Definition 4 does not need to be verified for these modules. Furthermore, for the intersection of a set of indexed languages over Σ^*, we use the convention that when the index set is empty, the result of the intersection is Σ^*.

Definition 2 *The multi-level interface system Φ is said to be* multi-level nonblocking *if for all $i \in \{1,\ldots,p\}$ and for all $k \in \{1,\ldots,n_i\}$ corresponding to each i, the following condition is satisfied:*

$$\overline{\mathcal{H}^i_{m_k} \cap \mathcal{I}^i_{m_k} \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_{m_j}} = \mathcal{H}^i_k \cap \mathcal{I}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j$$

Definition 3 *The multi-level interface system Φ is said to be* multi-level controllable *with respect to the alphabet partitions given by (1) and (2), if for all $i \in \{1,\ldots,p\}$ and for all $k \in \{1,\ldots,n_i\}$ corresponding to each i, the following conditions are satisfied:*

i) The event set of $\mathbf{G}^i_k$ and $\mathbf{S}^i_k$ is $\Sigma_{H^i_k}$ and the event set of $\mathbf{I}^i_k$ is $\Sigma_{I^i_k}$.

ii) $(\forall s \in \mathcal{G}^i_k \cap \mathcal{I}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j \cap \mathcal{S}^i_k)\ \ Elig_{\mathcal{G}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j}(s) \cap \Sigma_u \subseteq Elig_{\mathcal{S}^i_k \cap \mathcal{I}^i_k}(s)$

Definition 4 *The multi-level interface system Φ is said to be* multi-level weak interface consistent *with respect to the alphabet partitions given by (1) and (2), if for all $s \in \mathcal{H}^i_k \cap \mathcal{I}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j$, for all $i \in \{1,\ldots,p\}$ and for all $k \in \{1,\ldots,n_i\}$ corresponding to each i, the following conditions are satisfied:*

1) The event set of $\mathbf{H}^i_k$ is $\Sigma_{H^i_k}$.

2) $\mathbf{I}^i_k$ is a command-pair interface, $i > 1$.

3) $Elig_{\mathcal{I}^{i+1}_{j'}}(s) \cap \Sigma_{A^{i+1}_{j'}} \subseteq Elig_{\mathcal{H}^i_k}(s),\ \forall j' \in J^i_k$

4) $(\forall \rho \in \Sigma_{R^i_k})\ s\rho \in \mathcal{I}^i_k \Rightarrow$
$(\exists l \in \Sigma^*_{L^i_k})\ sl\rho \in \mathcal{H}^i_k \cap \mathcal{I}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j$

5) $(\forall \alpha \in \Sigma_{A^i_k})(s\rho \in \mathcal{H}^i_k \cap \mathcal{I}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j) \wedge$
$(s\rho\alpha \in \mathcal{I}^i_k) \Rightarrow (\exists l \in \Sigma^*_{L^i_k})\ s\rho l\alpha \in \mathcal{H}^i_k \cap \mathcal{I}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j$

6) $s \in \mathcal{I}^i_{m_k} \Rightarrow$
$(\exists l \in \Sigma^*_{L^i_k})\ sl \in \mathcal{H}^i_{m_k} \cap \mathcal{I}^i_{m_k} \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_{m_j},\ i > 1$

In words, Point 3 of Definition 4 requires that each modular language $\mathcal{H}^i_k$ be $\Sigma_{A^{i+1}_j}$-controllable with respect to

each of its lower-level interface languages $\mathcal{I}_j^{i+1}$. Points 4 and 5 require that request and answer events, respectively, be reachable in a module by events not shared with the corresponding higher-level module. Point 6 requires that if a string is marked and accepted by an interface, then it can be extended to a marked string in the corresponding lower-level module by events that again are not shared with the higher-level module. Point 4 of this definition differs from the corresponding two-level definition of [19] in that it has been relaxed from what was originally a controllability requirement to the reachability requirement prescribed in this paper. The Point 4 of [19] specifically required that each low-level module $\mathcal{H}_k^2$ of a two-level interface system be $\Sigma_{R_k^2}$-controllable with respect to its interface $\mathcal{I}_k^2$ as shown in (3):

$$(\forall s \in \mathcal{H}_k^2 \cap \mathcal{I}_k^2), Elig_{\mathcal{I}_k^2}(s) \cap \Sigma_{R_k^2} \subseteq Elig_{\mathcal{H}_k^2}(s) \tag{3}$$

The spirit of (3) is that the low-level modules have control only over those events shared with the high-level module that are answer events. Therefore, the high-level module knows that if it issues a request that is allowed by an interface, the low-level module will not disable it. This requirement is mirrored by Point 3 that specifies that the high-level module $\mathcal{H}^1$ be $\Sigma_{A_k^2}$-controllable with respect to each of its interfaces $\mathcal{I}_k^2$. These requirements are at the core of what enables conclusions to be drawn about the global system with only local analysis. Definition 4 with a Point 4 that is a multi-level version of (3) will be referred to as *multi-level interface consistency*. All systems that are *multi-level interface consistent* are also *multi-level weak interface consistent*. Therefore, all of the results of this paper that are demonstrated for systems that are multi-level weak interface consistent will also be satisfied by systems that are multi-level interface consistent.

The relaxed Point 4 of multi-level weak interface consistency still captures the intent of (3) by requiring instead that any continuation of a string s possible in the low-level must have a path of low-level events l that lead to a request event ρ if that request event is allowed by the associated interface. This requirement addresses all continuations of s by reapplying the requirement to a new string $s' = s\sigma$ after the occurrence of subsequent events. Therefore, even though the low-level module may not allow a request event immediately (as dictated by the controllability requirement of [19]), it will eventually enable the execution of the required request event following the occurrence of a string of low-level events. Since the request event is reached by local low-level events, we know that the low-level module cannot be prevented from reaching the request event by interaction with the interface or high-level module. The relaxed Point 4 is verified with complexity that is cubic in the number of states of the involved automata, whereas the original controllability-type requirement can be verified with quadratic complexity.

If a given system model is multi-level weak interface consistent, but not multi-level interface consistent, the model of a higher-level module can be modified by replacing a problematic request event by a newly added event to the system, making the original request event local to the lower-level. The newly added event represents a communication from the higher-level, rather than a low-level action. In this manner, the higher-level module could issue this new request and it would be enacted by the corresponding lower-level module immediately. The lower-level module then would go on to carry out some local events, including the event that had been replaced in the higher level, before enacting the associated answer event [15]. The drawback of adding this new request is that additional logic must be added to enforce the ordering between the original request and the new request. In our experience, this logic, when composed with the original models, may increase the size of the resulting composition. On the other hand, an advantage of adding these events is that if a modular supervisor and the associated interface enables an event, it is then guaranteed to be enabled for the global system immediately. With the relaxed Point 4, a request event is not able to occur in the global system until it is enabled by the interface and both the associated higher-level and lower-level modular supervisors. This breaks event localization and may require communication with the two associated levels, whereas employing additional events as described above allows all control to be implemented locally. Breaking event localization additionally may increase the complexity associated with simulating such a system.

4 Global Nonblocking and Controllability

In this section we will present the main results of this paper. Specifically, we will show that if a set of local conditions based on the definitions of Section 3 are satisfied, then the global multi-level system is nonblocking and the conjunction of modular supervisors and interfaces is controllable with respect to the global plant. We first, however, present results for the special instance of an interface system with only two levels.

4.1 Two-Level Case

Consider a two-level system consisting of a single high-level module $\mathbf{H}^1$, and a series of low-level modules $\mathbf{H}_1^2, \ldots, \mathbf{H}_n^2$, and interfaces $\mathbf{I}_1^2, \ldots, \mathbf{I}_n^2$. The notion of multi-level nonblocking introduced in Definition 2 reduces to the *level-wise nonblocking* definition presented in [19] when applied to a two-level interface system. Likewise, the property of multi-level controllability introduced in Definition 3 reduces to the *level-wise controllability* definition of [19] for a two-level interface

system. In the two-level case, Definition 4 reduces to *weak interface consistency* which is similar to the *interface consistency* definition presented in [19] with the only difference being the relaxation of Point 4. We can now use these definitions to present the following results that are special cases of the general multi-level theorems that will follow. Specifically, Theorem 1 is a result modified from [19] for the weak interface consistency definition, while Theorem 2 is taken directly from [19].

Theorem 1 *If the two-level interface system composed of DES* $\mathbf{H}^1, \mathbf{H}^2_1, \mathbf{I}^2_1, \ldots, \mathbf{H}^2_n, \mathbf{I}^2_n$*, is* level-wise nonblocking *and* weak interface consistent *with respect to the alphabet partition given by (1), then the global system is nonblocking:*

$$\overline{\mathcal{H}^1_m \cap \bigcap_{j=1,\ldots,n} (\mathcal{H}^2_{m_j} \cap \mathcal{I}^2_{m_j})} = \mathcal{H}^1 \cap \bigcap_{j=1,\ldots,n} (\mathcal{H}^2_j \cap \mathcal{I}^2_j)$$

Proof. Available in [13]. □

Theorem 2 *[19] If the two-level interface system composed of plant components* $\mathbf{G}^1, \mathbf{G}^2_1, \ldots, \mathbf{G}^2_n$*, supervisors* $\mathbf{S}^1, \mathbf{S}^2_1, \ldots, \mathbf{S}^2_n$*, and interfaces* $\mathbf{I}^2_1, \ldots, \mathbf{I}^2_n$*, is* level-wise controllable *with respect to the alphabet partition given by (1), then the supervisor language* $\mathcal{S} = \mathcal{S}^1 \cap \bigcap_{j=1,\ldots,n} (\mathcal{S}^2_j \cap \mathcal{I}^2_j)$ *is* Σ_u*-controllable with respect to the plant language* $\mathcal{G} = \mathcal{G}^1 \cap \bigcap_{j=1,\ldots,n} \mathcal{G}^2_j$*.*

4.2 General Multi-Level Case

We will now demonstrate results analogous to Theorem 1 and Theorem 2 for the general multi-level architecture. In order to better understand the process by which these theorems will be proved, the outline of the proofs will be discussed first for a system where each level of the hierarchy consists of only a single module. Controllability and nonblocking of the multi-level architecture will follow from the results presented for the two-level case. Specifically, the requirements of Theorem 1 and Theorem 2 must be met for a series of two-level systems consisting of a high-level module $\mathbf{H}^{i-1} = \mathbf{S}^{i-1} \| \mathbf{G}^{i-1}$, a low-level module $\mathbf{H}^i \| \mathbf{I}^{i+1} = \mathbf{S}^i \| \mathbf{G}^i \| \mathbf{I}^{i+1}$, and an interface $\mathbf{I}^i$, where $i = \{2, \ldots, p\}$. The proofs to follow rely on this modified formulation where the low-level plant includes the interface from the level below, that is, the low-level plant is considered to be $\mathbf{G}^i \| \mathbf{I}^{i+1}$. The disjointness of the alphabet partitions of (1) and (2) and the connected tree architecture will also be needed. For the interface $\mathbf{I}^p$, $\mathbf{H}^{p-1}$ is considered the high-level module and $\mathbf{H}^p$ is considered the low-level module since there is no interface preceding the bottom level of the hierarchy. In the two-level case, each interface $\mathbf{I}^2_k$ is considered part of the plant for level 1 and part of the supervisor for level 2.

Figure 3 illustrates the approach taken in the following proofs. The proofs begin with the two-level system at the top of the hierarchy, which is immediately nonblocking and controllable by Theorems 1 and 2. We then consider this system to be the "high-level" module and add another subsystem that is considered the "low-level" module. This process continues where the high-level module gets larger and larger and the low-level module is just the next subsystem considered. With this in mind, all low-level and multi-level requirements are immediately met. The high-level properties are shown by strong induction.

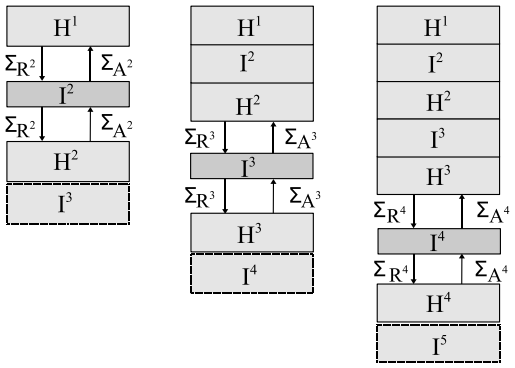

Fig. 3. Illustration of approach of proofs

The proposition given below will be employed in the proofs to follow to demonstrate controllability between languages that are separated in the hierarchy, that is, languages that do not share relevant events.

Proposition 1 *Let* $\mathcal{K}$*,* $\mathcal{L} \subseteq \Sigma^*$ *be prefix-closed languages. If* $\mathcal{K}$ *does not have any relevant events in the set* $\Sigma_u \subseteq \Sigma$*, then* $\mathcal{K}$ *is* Σ_u*-controllable with respect to* $\mathcal{L}$*.*

Proof. Available in [13]. □

The proofs of the main results of this paper, Theorem 3 and Theorem 4, can now be presented where the strong induction approach outlined above is employed. Recall Fig. 1 and Fig. 2 that illustrate the general multi-level architecture we are considering.

Theorem 3 *If the multi-level interface system* Φ *is* multi-level nonblocking *and* multi-level weak interface consistent *with respect to the alphabet partitions given by (1) and (2), then the complete system is nonblocking:*

$$\overline{\mathcal{H}^1_m \cap \mathcal{H}^2_m \cap \mathcal{I}^2_m \cap \ldots \cap \mathcal{H}^p_m \cap \mathcal{I}^p_m} = \mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}^2 \cap \ldots \cap \mathcal{H}^p \cap \mathcal{I}^p$$

where

$$\mathcal{H}^i_m = \mathcal{H}^i_{m_1} \cap \ldots \cap \mathcal{H}^i_{m_{n_i}}, \mathcal{I}^i_m = \mathcal{I}^i_{m_1} \cap \ldots \cap \mathcal{I}^i_{m_{n_i}},$$
$$\mathcal{H}^i = \mathcal{H}^i_1 \cap \ldots \cap \mathcal{H}^i_{n_i}, \mathcal{I}^i = \mathcal{I}^i_1 \cap \ldots \cap \mathcal{I}^i_{n_i}$$

Proof. Beginning at the top of the hierarchy, consider a two-level system consisting of a high-level module $\mathbf{H}_1^1$, a set of interfaces $\{\mathbf{I}_\ell^2\}$, and a corresponding set of low-level modules $\{\mathbf{H}_\ell^2||(||_{k\in J_\ell^2}\mathbf{I}_k^3)\}$ where $\ell = \{1,\ldots,n_2\}$. Because the overall system is multi-level nonblocking, we have for the first level that $\overline{\mathcal{H}_{m_1}^1 \cap \bigcap_{\ell=1}^{n_2}\mathcal{I}_{m_\ell}^2} = \mathcal{H}_1^1 \cap \bigcap_{\ell=1}^{n_2}\mathcal{I}_\ell^2$. Similarly for each module on the second level, we have that $\overline{\mathcal{H}_{m_\ell}^2 \cap \mathcal{I}_{m_\ell}^2 \cap \bigcap_{k\in J_\ell^2}\mathcal{I}_{m_k}^3} = \mathcal{H}_\ell^2 \cap \mathcal{I}_\ell^2 \cap \bigcap_{k\in J_\ell^2}\mathcal{I}_k^3$. These two results provide that this two-level component is level-wise nonblocking. Additionally, the fact that the overall system is multi-level weak interface consistent provides that this two-level component is weak interface consistent. Paying particular attention to Point 1 of the weak interface consistency definition in terms of (1), it is necessary that the high-level module and each low-level module only share events through their associated interface. Since $\Sigma_{H_1^1} \cap (\Sigma_{H_\ell^2} \cup \bigcup_{k\in J_\ell^2}\Sigma_{I_k^3}) = \Sigma_{I_\ell^2}$ for all $\ell \in \{1,\ldots,n_2\}$ by (1) and (2), this condition is met. Furthermore, it is also required that the alphabets of each low-level module $\mathbf{H}_\ell^2||(||_{k\in J_\ell^2}\mathbf{I}_k^3)$ be disjoint from one another. This condition is also met by (1) and (2). Therefore, Theorem 1 can be applied to show equation (4). Within a given level, all modules are included since it is assumed that the system has the form of a connected tree.

$$\overline{\mathcal{H}_m^1 \cap \mathcal{H}_m^2 \cap \mathcal{I}_m^2 \cap \mathcal{I}_m^3} = \mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}^2 \cap \mathcal{I}^3 \tag{4}$$

Now consider a system with a high-level module $\mathbf{H}_1^1\|\mathbf{H}_1^2\|\ldots\|\mathbf{H}_{n_2}^2\|\mathbf{I}_1^2\|\ldots\|\mathbf{I}_{n_2}^2$, a set of interfaces $\{\mathbf{I}_k^3\}$, and a corresponding set of low-level modules $\{\mathbf{H}_k^3||(||_{j\in J_k^3}\mathbf{I}_j^4)\}$ where $k = \{1,\ldots,n_3\}$. Based on the given assumptions, all low-level and multi-level requirements are known to be met. In particular, Point 1 of the weak interface consistency definition is satisfied by the fact that (1) and (2) imply that $(\Sigma_{H_1^1} \cup \Sigma_{H_1^2} \cup \ldots \cup \Sigma_{H_{n_2}^2} \cup \Sigma_{I_1^2} \cup \ldots \cup \Sigma_{I_{n_2}^2}) \cap (\Sigma_{H_k^3} \cup \bigcup_{j\in J_k^3}\Sigma_{I_j^4}) = \Sigma_{I_k^3}$ for all $k \in \{1,\ldots,n_3\}$ and each low-level module $\mathbf{H}_k^3||(||_{j\in J_k^3}\mathbf{I}_j^4)$ is disjoint from each of the other low-level modules. The level-wise nonblocking of the high-level has been shown to be met by (4). The only requirement left is Point 3 of the weak interface consistency definition, that is, it must be shown that the high-level language is $\Sigma_{A_k^3}$-controllable with respect to each $\mathcal{I}_k^3$, $\forall k = \{1,\ldots,n_3\}$.

Consider a single interface language $\mathcal{I}_k^3$ from this two-level system. On level 2, there is a single module H_ℓ^2 that shares events with this interface, that is, $(\Sigma_{H_\ell^2} \cap \Sigma_{I_k^3} \neq \emptyset)$. The language generated by this module $\mathcal{H}_\ell^2$ is $\Sigma_{A_k^3}$-controllable with respect to $\mathcal{I}_k^3$ due to the multi-level weak consistency requirement. For those modules $H_{\ell'}^2$ from level 2 that do not share events with I_k^3, they do not possess any events that are in the set $\Sigma_{A_k^3}$ by equation (2). Therefore by Proposition 1, each language $\mathcal{H}_{\ell'}^2$ for which $\ell' \neq \ell$ is also $\Sigma_{A_k^3}$-controllable with respect to $\mathcal{I}_k^3$.

Furthermore, each interface from level 2, $\mathbf{I}_\ell^2$, and the module from the level 1, $\mathbf{H}_1^1$, also do not share any events with the event set $\Sigma_{A_k^3}$ by (1) and (2). Applying Proposition 1 again demonstrates that each of the languages generated by these DES are $\Sigma_{A_k^3}$-controllable with respect to the interface language $\mathcal{I}_k^3$.

Since the module language $\mathcal{H}_1^1$ and the interface and module languages $\mathcal{I}_\ell^2$ and $\mathcal{H}_\ell^2$, where $\ell = \{1,\ldots,n_2\}$, are $\Sigma_{A_k^3}$-controllable with respect to $\mathcal{I}_k^3$, so is the composition of these languages since each of the languages is prefix-closed. Otherwise stated, $\mathcal{H}_1^1 \cap \mathcal{H}_1^2 \cap \ldots \cap \mathcal{H}_{n_2}^2 \cap \mathcal{I}_1^2 \cap \ldots \cap \mathcal{I}_{n_2}^2$ is $\Sigma_{A_k^3}$-controllable with respect to the interface language $\mathcal{I}_k^3$. Repeating this logic, this high-level language can be shown to be $\Sigma_{A_k^3}$-controllable with respect to any interface language $\mathcal{I}_k^3$, $\forall k = \{1,\ldots,n_3\}$. Therefore, we have shown that Point 3 is satisfied. Since all level-wise nonblocking and weak interface consistency requirements are met for this two-level system, Theorem 1 then gives us:

$$\overline{\mathcal{H}_m^1 \cap \mathcal{H}_m^2 \cap \mathcal{I}_m^2 \cap \mathcal{H}_m^3 \cap \mathcal{I}_m^3 \cap \mathcal{I}_m^4} = \mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}^2 \cap \mathcal{H}^3 \cap \mathcal{I}^3 \cap \mathcal{I}^4$$

This logic is repeated until all modules on all p levels have been addressed. Low-level modules that do not have any interfaces below them are slightly different in that each module just has the form $\mathbf{H}_k^i$. However, they still satisfy the level-wise nonblocking and weak interface consistency requirements leading to the desired result. □

In the above proof, the "low-level" module always stands alone, thus Point 4 of the weak interface consistency definition is immediately satisfied (as well as all other low-level requirements).

Theorem 4 *If the multi-level interface system* Φ *is* multi-level controllable *with respect to the alphabet partitions given by (1) and (2), then the supervisor language* $\mathcal{S} = \mathcal{S}^1 \cap \mathcal{S}^2 \cap \mathcal{I}^2 \cap \ldots \cap \mathcal{S}^p \cap \mathcal{I}^p$ *is* Σ_u*-controllable with respect to the plant language* $\mathcal{G} = \mathcal{G}^1 \cap \ldots \cap \mathcal{G}^p$*, where* $\mathcal{S}^i = \mathcal{S}_1^i \cap \ldots \cap \mathcal{S}_{n_i}^i$, $\mathcal{I}^i = \mathcal{I}_1^i \cap \ldots \cap \mathcal{I}_{n_i}^i$, *and* $\mathcal{G}^i = \mathcal{G}_1^i \cap \ldots \cap \mathcal{G}_{n_i}^i$.

Proof. Beginning at the top of the hierarchy, consider the two-level interface system consisting of a high-level plant $\mathbf{G}_1^1$ and supervisor $\mathbf{S}_1^1$, a set of interfaces $\{\mathbf{I}_\ell^2\}$, and a corresponding set of low-level plants $\{\mathbf{G}_\ell^2||(||_{k\in J_\ell^2}\mathbf{I}_k^3)\}$ and supervisors $\{\mathbf{S}_\ell^2\}$ where $\ell = \{1,\ldots,n_2\}$. This high-level plant and supervisor have an alphabet of $\Sigma_{H_1^1}$, each

low-level plant and supervisor have an alphabet of $\Sigma_{H_\ell^2}$ since $\Sigma_{H_\ell^2} \supseteq \bigcup_{k \in J_\ell^2} \Sigma_{I_k^3}$ for each $\ell \in \{1, \ldots, n_2\}$, and each interface has an alphabet of $\Sigma_{I_\ell^2}$. Therefore by (1) and (2), the high-level module and each low-level module only share events through their common interface, that is, $\Sigma_{H_1^1} \cap \Sigma_{H_\ell^2} = \Sigma_{I_\ell^2}$. Furthermore, the alphabets of each interface is disjoint from the alphabets of the other interfaces. Since the overall system is multi-level controllable, we also have for the first level that $\mathcal{S}_1^1$ is Σ_u-controllable with respect to $\mathcal{G}_1^1 \cap \bigcap_{\ell=1}^{n_2} \mathcal{I}_\ell^2$. Similarly for the second level we have that each $\mathcal{S}_\ell^2 \cap \mathcal{I}_\ell^2$ is Σ_u-controllable with respect $\mathcal{G}_\ell^2 \cap \bigcap_{k \in J_\ell^2} \mathcal{I}_k^3$. These results provide that this two-level component is level-wise controllable. Therefore, Theorem 2 can be applied to show that the language $\mathcal{S}^1 \cap \mathcal{S}^2 \cap \mathcal{I}^2$ is Σ_u-controllable with respect to $\mathcal{G}^1 \cap \mathcal{G}^2 \cap \mathcal{I}^3$. Within a given level, all modules are included since the system is connected.

Now consider an interface system with a high-level plant $\mathbf{G}_1^1 \| \mathbf{G}_1^2 \| \ldots \| \mathbf{G}_{n_2}^2$ and supervisor $\mathbf{S}_1^1 \| \mathbf{S}_1^2 \| \ldots \| \mathbf{S}_{n_2}^2 \| \mathbf{I}_1^2 \| \ldots \| \mathbf{I}_{n_2}^2$, a set of interfaces $\{\mathbf{I}_k^3\}$, and a corresponding set of low-level plants $\{\mathbf{G}_k^3 || (\|_{j \in J_k^3} \mathbf{I}_j^4)\}$ and supervisors $\{\mathbf{S}_k^3\}$ where $k = \{1, \ldots, n_3\}$. The alphabet of the high-level plant and supervisor is $\Sigma_{H_1^1} \cup \Sigma_{H_1^2} \cup \ldots \cup \Sigma_{H_{n_2}^2}$ since each $\Sigma_{H_\ell^2} \supseteq \Sigma_{I_\ell^2}$, likewise, the alphabet of each low-level plant and supervisor is $\Sigma_{H_k^3}$, and the alphabet of each interface is $\Sigma_{I_k^3}$. Therefore, by (1) and (2) the high-level module and low-level module only share events through their common interface, that is, $(\Sigma_{H_1^1} \cup \Sigma_{H_1^2} \cup \ldots \cup \Sigma_{H_{n_2}^2}) \cap \Sigma_{H_k^3} = \Sigma_{I_k^3}$. Additionally, the alphabet of each interface is disjoint from the alphabets of the other interfaces. Since the overall system is multi-level controllable, the low-level satisfies the level-wise controllability definition. The high-level satisfies the level-wise controllability definition based on the previous step of this proof. Therefore, Theorem 2 can be applied again to show that the language $\mathcal{S}^1 \cap \mathcal{S}^2 \cap \mathcal{I}^2 \cap \mathcal{S}^3 \cap \mathcal{I}^3$ is Σ_u-controllable with respect to $\mathcal{G}^1 \cap \mathcal{G}^2 \cap \mathcal{G}^3 \cap \mathcal{I}^4$.

This logic is repeated until all modules on all p levels have been addressed. Lower-level modules that do not have any interfaces below them are slightly different in that their plant components just have the form $\mathbf{G}_k^i$. However, they still satisfy level-wise controllability leading to the desired result. □

5 Supervisor Synthesis

In the previous section we demonstrated that for a given multi-level interface system the local properties of Section 3 are sufficient for guaranteeing the global properties of nonblocking and controllability. It was not, however, specified how to construct the multi-level interface system. In this section we will outline a systematic approach for constructing the component supervisors for a multi-level hierarchical interface-based architecture that is guaranteed to meet the necessary requirements by construction. The approach presented here extends the work of [18] that provides results on how to construct high and low-level supervisors that are optimal with respect to a given specification and set of interfaces in the two-level case. The extension to the multi-level case specifically requires that a modular supervisor be synthesized to meet low and high-level requirements simultaneously.

The basic approach of [18] is similar to the traditional approach for constructing a supremal controllable and nonblocking sublanguage [25]. This involves first the construction of the language that represents the portion of the system's uncontrolled behavior that is allowed by the given specification. This language is then pruned to remove those strings that violate controllability or nonblocking. This construction is in general performed on the automaton generator of the language.

For the multi-level interface-based architecture of this paper, we will construct a supervisor $\mathbf{S}_k^i$ with respect to a component plant $\mathbf{G}_k^i$, specification $\mathbf{E}_k^i$, and set of interfaces $\mathbf{I}_k^i$, $\{\mathbf{I}_j^{i+1}\}$ where j represents all those indices in the set J_k^i for the given module. The starting point for the synthesis of the supervisor for the $(i,k)^{th}$ module will then be the automaton $\mathbf{Z}_k^i = \mathbf{G}_k^i \| \mathbf{E}_k^i \| \mathbf{I}_k^i \| (\|_{j \in J_k^i} \mathbf{I}_j^{i+1})$. The generated and marked languages for this automaton lifted to the global alphabet Σ are then:

$$\mathcal{Z}_k^i = P_{H_k^i}^{-1}(\mathcal{L}(\mathbf{Z}_k^i)) = \mathcal{G}_k^i \cap \mathcal{E}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}$$

$$\mathcal{Z}_{m_k}^i = P_{H_k^i}^{-1}(\mathcal{L}_m(\mathbf{Z}_k^i)) = \mathcal{G}_{m_k}^i \cap \mathcal{E}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1} \quad (5)$$

The synthesis of the modular supervisor languages then requires the removal of those strings that violate any of the properties required of the multi-level interface-based approach to control. Any continuations of these removed strings are also removed in order to generate a prefix-closed language. Some conditions of the definitions of Section 3 will, however, not be addressed by the synthesis algorithm and must be satisfied upfront. These requirements are captured in the following definition where Ψ represents a multi-level specification interface system similar to Φ. Here Ψ differs from Φ in that it includes modular specifications in place of modular supervisors since the supervisors have not been synthesized yet.

Definition 5 *The multi-level specification interface system Ψ is said to be* multi-level well-formed *with respect to the alphabet partitions given by (1) and (2), if for all $i \in \{1, \ldots, p\}$ and for all $k \in \{1, \ldots, n_i\}$ corresponding to each i, the following conditions are satisfied:*

1) The event set of $\mathbf{G}_k^i$ and $\mathbf{E}_k^i$ is $\Sigma_{H_k^i}$.

2) $\mathbf{I}_k^i$ *is a command-pair interface,* $i > 1$.

In the subsequent results of this section, it will be assumed that the multi-level specification interface system Ψ is multi-level well-formed with respect to the alphabet partitions given by (1) and (2). A language satisfying the remaining conditions of the definitions of Section 3 is then said to be $(i,k)^{th}$ *multi-level weak interface controllable* (MIC_k^i). This specific term is defined below. Whereas, the language $\mathcal{Z}_k^i$ is the starting point for the supervisor synthesis procedure for the $(i,k)^{th}$ module, the language $\mathcal{Z}$ in the following definitions is an arbitrary language.

Definition 6 *Let* $\mathcal{Z} \subseteq \Sigma^*$. *For system* Ψ, *the language* $\mathcal{Z}$ *is* $(i,k)^{th}$ multi-level weak interface controllable (MIC_k^i) *if for all* $s \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$, *the following conditions are satisfied:*

1) $Elig_{\mathcal{G}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1}}(s) \cap \Sigma_u \subseteq Elig_{\overline{\mathcal{Z}}\cap \mathcal{I}_k^i}(s)$

2) $Elig_{\mathcal{I}_{j'}^{i+1}}(s) \cap \Sigma_{A_{j'}^{i+1}} \subseteq Elig_{\mathcal{G}_k^i \cap \overline{\mathcal{Z}}}(s),\ \forall j' \in J_k^i$

3) $(\forall \rho \in \Sigma_{R_k^i})\ s\rho \in \mathcal{I}_k^i \Rightarrow$
$(\exists l \in \Sigma_{L_k^i}^*)\ sl\rho \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$

4) $(\forall \rho \in \Sigma_{R_k^i})(\forall \alpha \in \Sigma_{A_k^i})\ s\rho \in \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \wedge$
$s\rho\alpha \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$

5) $s \in \mathcal{I}_{m_k}^i \Rightarrow$
$(\exists l \in \Sigma_{L_k^i}^*)\ sl \in \mathcal{G}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_{m_j}^{i+1} \cap \mathcal{Z},\ i > 1$

Point 1 of the above definition corresponds to the multi-level controllability requirement, while Points 2-5 correspond to Points 3-6 of the multi-level weak interface consistency requirement.

To formally present this approach to supervisor synthesis, we now define for an arbitrary language $\mathcal{E} \subseteq \Sigma^*$ a class of sublanguages of $\mathcal{E}$ that are $(i,k)^{th}$ multi-level weak interface controllable for the given multi-level specification interface system Ψ:

$$\mathcal{C}_{M_k^i}(\mathcal{E}) := \{\mathcal{Z} \subseteq \mathcal{E} \mid \mathcal{Z} \text{ is } MIC_k^i \text{ with respect to } \Psi\}$$

The following proposition then demonstrates that the set $\mathcal{C}_{M_k^i}(\mathcal{E})$ is nonempty and closed under union. This, therefore, implies that a unique supremal element exists for this set.

Proposition 2 *Let* $\mathcal{E} \subseteq \Sigma^*$. *For system* Ψ, $\mathcal{C}_{M_k^i}(\mathcal{E})$ *is nonempty and closed under arbitrary union. In particular,* $\mathcal{C}_{M_k^i}(\mathcal{E})$ *contains a (unique) supremal element that we will denote* $\sup \mathcal{C}_{M_k^i}(\mathcal{E})$.

Proof. Available in [13]. □

Now that we have established that a supervisor exists that is maximally permissive with respect to a given specification and set of interfaces, we would now like to demonstrate that this language can be constructed. With this in mind, we will define the operator $\Omega_{M_k^i}$ and show that its fixpoint is $\sup \mathcal{C}_{M_k^i}(\mathcal{Z}_{m_k}^i)$. The language fixpoint operator $\Omega_{M_k^i}$ will be defined in terms of two intermediate operators $\Omega_{MNB_k^i}$ and $\Omega_{MIC_k^i}$ that we will define first. The operator $\Omega_{MNB_k^i}$ specifically returns those strings of a given prefix-closed language that are marked in $\mathcal{Z}_{m_k}^i$.

Definition 7 *For system* Ψ, *we define the* nonblocking operator $\Omega_{MNB_k^i} : \Sigma^* \to \Sigma^*$, *for arbitrary* $\mathcal{Z} \subseteq \Sigma^*$ *as* $\Omega_{MNB_k^i}(\mathcal{Z}) := \mathcal{Z} \cap \mathcal{Z}_{m_k}^i$.

The next operator $\Omega_{MIC_k^i}$ removes from a given prefix-closed language those strings that fail any of the elements of Definition 6, $(i,k)^{th}$ multi-level weak interface controllability. Continuations of the failed strings are also removed to maintain prefix-closure, as indicated by the $\text{Ext}_{\overline{\mathcal{Z}}}$ operator, that returns the continuations in $\overline{\mathcal{Z}}$ of a given set of strings. The function $\text{Ext}_{\overline{\mathcal{Z}}} : \Sigma^* \to \Sigma^*$ is defined formally as follows, $\text{Ext}_{\overline{\mathcal{Z}}}(K) = \{t \in \overline{\mathcal{Z}} | s \leq t \text{ for some } s \in K\}$.

Definition 8 *For system* Ψ, *we define the* interface controllable operator $\Omega_{MIC_k^i} : \Sigma^* \to \Sigma^*$, *for arbitrary* $\mathcal{Z} \subseteq \Sigma^*$ *as* $\Omega_{MIC_k^i}(\mathcal{Z}) := \overline{\mathcal{Z}} - Ext_{\overline{\mathcal{Z}}}(FailMIC_k^i(\overline{\mathcal{Z}}))$, *where*

$$\begin{aligned}
&FailMIC_k^i(\overline{\mathcal{Z}}) := \{s \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \textstyle\bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}} \mid \\
&\neg[Elig_{\mathcal{G}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1}}(s) \cap \Sigma_u \subseteq Elig_{\overline{\mathcal{Z}}\cap \mathcal{I}_k^i}(s)] \\
&\vee [\exists j \in J_k^i \mid \neg(Elig_{\mathcal{I}_j^{i+1}}(s) \cap \Sigma_{A_k^i} \subseteq Elig_{\mathcal{G}_k^i \cap \overline{\mathcal{Z}}}(s))] \\
&\vee \neg[(\forall \rho \in \Sigma_{R_k^i}) s\rho \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*) \\
&sl\rho \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \textstyle\bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}] \\
&\vee \neg[(\forall \rho \in \Sigma_{R_k^i})(\forall \alpha \in \Sigma_{A_k^i}) s\rho \in \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \textstyle\bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \wedge \\
&s\rho\alpha \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \textstyle\bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}] \\
&\vee \neg[s \in \mathcal{I}_{m_k}^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*) \\
&sl \in \mathcal{G}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \textstyle\bigcap_{j\in J_k^i} \mathcal{I}_{m_j}^{i+1} \cap \mathcal{Z},\ i > 1]\}
\end{aligned}$$

We can now define our fixpoint operator $\Omega_{M_k^i}$.

Definition 9 *For system* Ψ, *we define the* $(i,k)^{th}$ *fixpoint operator,* $\Omega_{M_k^i} : \Sigma^* \to \Sigma^*$, *for arbitrary* $\mathcal{Z} \subseteq \Sigma^*$ *as* $\Omega_{M_k^i} := \Omega_{MNB_k^i}(\Omega_{MIC_k^i}(\mathcal{Z}))$.

The following important result demonstrates that if $\Omega_{M_k^i}(\mathcal{Z}_k^i)$ reaches a fixpoint, then the fixpoint is equal to $\sup \mathcal{C}_{M_k^i}(\mathcal{Z}_{m_k}^i)$.

Theorem 5 *For system Ψ, if there exists $j \in \{0,1,2,\ldots\}$ such that $\Omega^j_{M^i_k}(\mathcal{Z}^i_k)$ is a fixpoint, then $\Omega^j_{M^i_k}(\mathcal{Z}^i_k) = \sup \mathcal{C}_{M^i_k}(\mathcal{Z}^i_{m_k})$.*

Proof. Available in [13]. □

Finally, the following demonstrates that if the resulting supremal element is employed as our supervisor language, then the necessary interface-based requirements of Section 3 are satisfied.

Corollary 1 *For system Ψ, if there exists $j \in \{0,1,2,\ldots\}$ such that $\Omega^j_{M^i_k}(\mathcal{Z}^i_k)$ is a fixpoint, then system Φ with $\mathcal{S}^i_{m_k} = \Omega^j_{M^i_k}(\mathcal{Z}^i_k)$ and $\mathcal{S}^i_k = \overline{\mathcal{S}^i_{m_k}}$ satisfies Points 3, 4, 5, and 6 of the multi-level weak interface consistency definition, Point ii) of the multi-level controllability definition, and the multi-level nonblocking definition.*

Proof. Available in [13]. □

The fixpoint operators that have been presented so far are language-based. The specific supervisor synthesis algorithms presented in [4] for the two-level case are, however, automata-based. While we will not present a specific algorithm for automata-based supervisor construction, like [4] we can show an equivalence between removing strings from a language and removing states from an automaton. The basic approach to showing the equivalence between a language-based algorithm and an automata-based algorithm is to show that if a string that reaches a state q fails to meet some necessary condition, then all strings that reach this state q will also fail to meet the necessary condition. This way, removing a state from an automaton only removes strings that violate the given requirement.

A standard result from automaton theory addresses the property of blocking, that is, if two strings reach the same state then one string is in the prefix closure of the marked language if and only if the other string is [2]. The following proposition similarly addresses the requirements of the $(i,k)^{th}$ multi-level weak interface controllability definition. In the statement of the proposition, we will employ the automaton $\mathbf{H}^{i\prime}_k$ which is equal to the automaton $\mathbf{G}^i_k \| \mathbf{I}^i_k \| (\|_{j \in J^i_k} \mathbf{I}^{i+1}_j) \| \mathbf{S}^i_k$, with self-loops added for all events in the set $\Sigma - \Sigma_{H^i_k}$. Therefore, $\mathbf{H}^{i\prime}_k$ has an event set of Σ and $\mathcal{L}(\mathbf{H}^{i\prime}_k) = \mathcal{G}^i_k \cap \mathcal{I}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j \cap \mathcal{S}^i_k$. In the following, the language $\mathcal{S}^i_k$ does not necessarily satisfy the necessary interface-based requirements. More specifically, $\mathcal{S}^i_k$ may be an intermediate language that is in the process of being pruned to ultimately generate a supervisor that does satisfy the necessary interface-based requirements.

Proposition 3 *For system Φ, Let $\mathbf{H}^{i\prime}_k = (Q^i_k, \Sigma, \delta^i_k, q^i_{0_k}, Q^i_{m_k})$. It thus follows that for all $s,t \in \mathcal{L}(\mathbf{H}^{i\prime}_k)$, if $\delta^i_k(q^i_{0_k}, s) = \delta^i_k(q^i_{0_k}, t)$ then*

1) $Elig_{\mathcal{G}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j}(s) \cap \Sigma_u \subseteq Elig_{\mathcal{S}^i_k \cap \mathcal{I}^i_k}(s) \Leftrightarrow$
$Elig_{\mathcal{G}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j}(t) \cap \Sigma_u \subseteq Elig_{\mathcal{S}^i_k \cap \mathcal{I}^i_k}(t)$

2) $Elig_{\mathcal{I}^{i+1}_j}(s) \cap \Sigma_{A^{i+1}_j} \subseteq Elig_{\mathcal{H}^i_k}(s) \Leftrightarrow$
$Elig_{\mathcal{I}^{i+1}_j}(t) \cap \Sigma_{A^{i+1}_j} \subseteq Elig_{\mathcal{H}^i_k}(t),\ \forall j \in J^i_k$

3) $(\forall \rho \in \Sigma_{R^i_k})$
$\Big[[s\rho \in \mathcal{I}^i_k] \Rightarrow [(\exists l \in \Sigma^*_{L^i_k}) sl\rho \in \mathcal{H}^i_k \cap \mathcal{I}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j]\Big] \Leftrightarrow$
$\Big[[t\rho \in \mathcal{I}^i_k] \Rightarrow [(\exists l \in \Sigma^*_{L^i_k}) tl\rho \in \mathcal{H}^i_k \cap \mathcal{I}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j]\Big]$

4) $(\forall \rho \in \Sigma_{R^i_k})(\forall \alpha \in \Sigma_{A^i_k})$

$[[s\rho \in \mathcal{H}^i_k \cap \mathcal{I}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j \wedge s\rho\alpha \in \mathcal{I}^i_k] \Rightarrow$

$[(\exists l \in \Sigma^*_{L^i_k})\ s\rho l\alpha \in \mathcal{H}^i_k \cap \mathcal{I}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j]] \Leftrightarrow$

$[[t\rho \in \mathcal{H}^i_k \cap \mathcal{I}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j \wedge t\rho\alpha \in \mathcal{I}^i_k] \Rightarrow$

$[(\exists l \in \Sigma^*_{L^i_k})\ t\rho l\alpha \in \mathcal{H}^i_k \cap \mathcal{I}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j]]$

5)
$\Big[[s \in \mathcal{I}^i_{m_k}] \Rightarrow [(\exists l \in \Sigma^*_{L^i_k}) sl \in \mathcal{H}^i_{m_k} \cap \mathcal{I}^i_{m_k} \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_{m_j}]\Big] \Leftrightarrow$
$\Big[[t \in \mathcal{I}^i_{m_k}] \Rightarrow [(\exists l \in \Sigma^*_{L^i_k}) tl \in \mathcal{H}^i_{m_k} \cap \mathcal{I}^i_{m_k} \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_{m_j}]\Big]$

Proof. Available in [13]. □

Based on the above, it becomes apparent that an application of the language-based fixpoint operator $\Omega_{M^i_k}$ is equivalent to removing at least one state from the automaton $\mathbf{H}^{i\prime}_k$, or results in a fixpoint. The removal of a state consequently removes strings with continuations from that state from the language $\mathcal{L}(\mathbf{H}^{i\prime}_k)$. This is consistent with the definition of the operator $\Omega_{M^i_k}$. Assuming that we have regular languages, our automata generators have a finite number of states. Therefore, even a naive algorithm that tests each state of an automaton one at a time, and removes states that are not coreachable or that are reached by strings that violate $(i,k)^{th}$ multi-level weak interface controllability, will reach a fixpoint in finite time.

6 Implementation Example and Discussion

In this section we will demonstrate an application of this new theory to the flexible manufacturing system (FMS)

example shown in Fig. 4 (modified from [7]). This existing example was chosen for the purposes of comparison, and though the system was not designed hierarchically, the hierarchical structure imposed on the system provides a means for reducing complexity and improving reconfigurability. With this system parts enter from the left via the conveyor **C2**. From **C2** the parts pass through buffer **B2** to a handling robot **R1**. This robot then passes parts, through buffer **B4**, to a lathe that can generate two different types of parts. After the lathe has finished an operation and returned a part to the robot **R1**, again through buffer **B4**, the robot then passes the part to either buffer **B6** or buffer **B7** depending on the part type. If passed to **B7**, the part is then sent to a painting machine **PM** via conveyor **C3** and buffer **B8**. Once the painting operation is finished, the part is passed back through the same sequence by which it arrived. From buffers **B6** and **B7**, the two different parts are passed to the machine **AM** and onto another handling robot **R2** through buffer **B9**. The handling robot then passes parts to a **Mill** for finishing via buffer **B3** before being returned to the robot. In this example, the machines are considered the component plants and the buffers are considered the component specifications where it is desired that the buffers not underflow or overflow. At the end of this section, we will also discuss the complexity of this multi-level approach, in particular, its scalability as compared to the original two-level architecture.

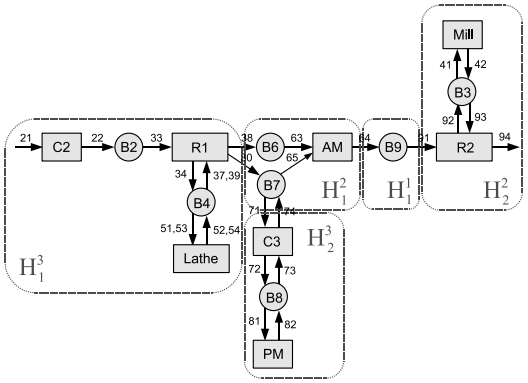

Fig. 4. Flexible manufacturing system example

6.1 Flexible Manufacturing System Example

The automaton models of the machines and buffers in our FMS example can be found in Fig. 5 and Fig. 6 respectively where states with double circles represent marked states and states with a short arrow represent initial states. The convention employed here is that odd numbers represent controllable events and even numbers correspond to uncontrollable events. A table describing each of the events is given below.

Application of our approach depends in part on designer understanding. Specifically, how the system components are partitioned into modules, how request and answer events are chosen, and how interfaces are constructed, are all areas where designer intuition can enter in. In this section we will present a procedure for implementing a multi-level interface-based architecture where we provide some heuristics for making some of the necessary design choices. Ultimately, it may be necessary to try multiple combinations to find a satisfactory solution.

Table 1
Description of FMS events

Event	Description
21	part to conveyor 2
22	part from conveyor 2 to buffer 2
30	part type 2 to buffer 7
33	part from buffer 2
34	part to buffer 4
37	part type 1 from buffer 4
38	part type 1 to buffer 6
39	part type 2 from buffer 4
41	part from buffer 3 and mill begins
42	mill finishes and part to buffer 3
51	part from buffer 4 and lathe begins type 1
52	lathe finishes type 1 and part to buffer 4
53	part from buffer 4 and lathe begins type 2
54	lathe finishes type 2 and part to buffer 4
61	AM performs preparation operation
63	AM begins operation on part type 1
64	AM finishes operation and part to buffer 9
65	AM begins operation on part type 2
71	part from buffer 7 to conveyor 3 forward
72	part from conveyor 3 to buffer 8
73	part from buffer 8 to conveyor 3 backward
74	part from conveyor 3 and to buffer 7
81	part from buffer 8 and PM begins
82	PM finishes and part to buffer 8
91	part from buffer 9
92	part to buffer 3
93	part from buffer 3
94	part exits manufacturing system

Application of the Multi-Level Interface-Based Approach to Supervisory Control

Step 1: Group system components into modules - The grouping of the components of the global system into modules has many different possibilities that in general do not lead to a unique solution. Two requirements on

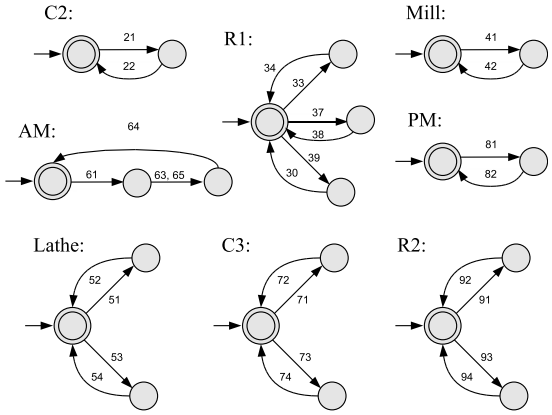

Fig. 5. Automaton models of plant components

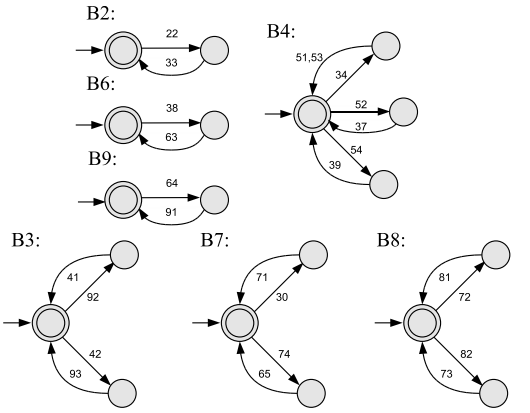

Fig. 6. Automaton models of specification components

the partitioning are that it must satisfy (1) and (2). This means, for one, all interaction between modules must take place through interfaces, and each interface must be completely disjoint from all other interfaces. Additionally, while each module can interact with multiple modules on the level of hierarchy immediately below it, it can only interact with a single module on the level of hierarchy above it. Since the hierarchy is required to have the structure of a connected tree, it is also implied that there can be no closed loops formed among the modules, that is, a module cannot be simultaneously above and below another module in the hierarchy. Despite this restriction, a multi-level architecture can still be applied to systems whose components interact with one another such that they form a closed-loop. The complexity reduction that can be achieved for a system whose components are tightly coupled, however, may be limited. Some guidelines for choosing a partitioning that will limit the resulting computational complexity is to keep the number of components in a given module small, while at the same time limiting the number of interfaces that each module interacts with. Other heuristic guidelines for constructing a multi-level interface-based hierarchy can be found in [12].

The dotted boxes in Fig. 4 demonstrate the partition chosen for this example. For instance, the supervisor $\mathbf{S}_1^2$ will be constructed for the module $\mathbf{H}_1^2$ with respect to the plant $\mathbf{G}_1^2 = \mathbf{AM}$ and specification $\mathbf{E}_1^2 = \mathbf{B6} \| \mathbf{B7}$. Figure 7 illustrates the hierarchy imposed upon the system and the flow of information.

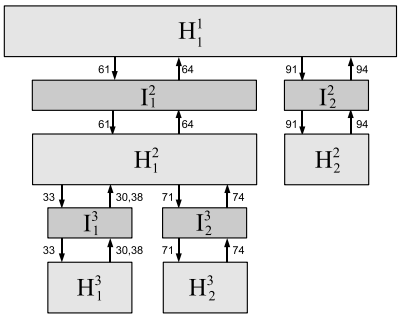

Fig. 7. Hierarchy imposed on flexible manufacturing system example

Step 2: Determine sets of request and answer events - Examine the plant components of each module and choose which events are to comprise the request and answer events associated with each interface. This again is a heuristic process that depends on designer understanding of the system and may require some iteration. Note that all events shared between any pair of modules must be included in either the request or answer event set for the associated interface. Additionally, it is often helpful to think of request events as events that start a process and answer events as events that finish a process. Examining the plant automata of a module can give some indication of which events begin and which ones finish a process.

For our example, the events shared between module $\mathbf{H}_1^3$ and module $\mathbf{H}_1^2$ are 30 and 38. Examining the automaton model of **R1** in Fig. 5, it can be seen that both of these events represent the completion of a process since they both end at a marked state. Therefore, we will consider them answer events $\Sigma_{A_1^3} = \{30, 38\}$. It can also be seen by inspection that the events that start these two processes correspond to events 39 and 37 respectively. In this case, however, we will take some liberties in what we consider a "process." We will consider our process to be the successive occurrence of two smaller operations. In this instance, the request event is the beginning of the first operation and the answer event is the completion of the second operation. Therefore, we will consider event 33 to be the request event corresponding to both answers, thus $\Sigma_{R_1^3} = \{33\}$. Following this general procedure, we further arrive at the following sets of request and answer events: $\Sigma_{R_2^3} = \{71\}$, $\Sigma_{A_2^3} = \{74\}$, $\Sigma_{R_1^2} = \{61\}$, $\Sigma_{A_1^2} = \{64\}$, $\Sigma_{R_2^2} = \{91\}$, and $\Sigma_{A_2^2} = \{94\}$.

Step 3: Assume a form for each of the interface automata

- Based on the set of request and answer events from the previous step of this procedure, along with the designer's understanding of the system, a form for the interface models needs to be assumed. Based on the requirements of our approach, the interfaces must satisfy the command-pair interface format of Definition 1. Figure 8 shows the interface automata designed for this example.

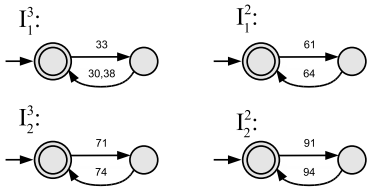

Fig. 8. Proposed interfaces for flexible manufacturing example

Step 4: Synthesize supervisors - Supervisors can now be synthesized by the approach of Section 5. The basic idea is that the interface, specification, and plant component automata associated with a given module are composed. States of this resulting automaton are then removed if they are reached by strings that fail any of the requirements of Definition 6 or if they are not coreachable.

For example, the supervisor $\mathbf{S}_1^2$ is synthesized for the plant $\mathbf{G}_1^2 = \mathbf{AM}$ in order to satisfy requirements with respect to the specification $\mathbf{E}_1^2 = \mathbf{B6}\|\mathbf{B7}$ and the set of neighboring interfaces $\{\mathbf{I}_1^3, \mathbf{I}_2^3, \mathbf{I}_1^2\}$. Note, in the process of building the supervisor for the module $\mathbf{H}_1^2$, the neighboring modules $\mathbf{H}_1^3$, $\mathbf{H}_2^3$, and $\mathbf{H}_1^2$ did not need to be considered at all. All necessary information was passed through the interfaces. This illustrates how global properties are met through the construction of local supervisors.

Once supervisors have been constructed for each module, the procedure is done. See Table 2 for a summary. ◇

For the purposes of comparison, the composition of all plant and specification components in the FMS example results in an automaton with 218,592 states and 929,904 transitions. Furthermore, the supremal controllable sublanguage for the monolithic system is generated by an automaton with 10,980 states and 42,666 transitions. A traditional modular solution greatly reduces the complexity of generating control for this example, but results in blocking.

In the generation of the multi-level hierarchical interface-based control, the largest automaton that was constructed had 60 states and 135 transitions. This automaton was built in the process of constructing the supervisor for module $\mathbf{H}_1^2$. The resulting global closed-loop behavior is safe and nonblocking. The sizes of the automata built in this process are substantially smaller than those required in building the monolithic supervisor, thereby giving some indication of the advantage of this approach. A drawback of the interface-based solution is the loss of global optimality. Specifically, the interface-based control only allows for four pieces to be active in the factory at any given time. The monolithic solution allows for a maximum of seven pieces to be active at one time. The supervisor for each module, however, is locally optimal with respect to the interface-based conditions. One of the reasons for the loss of global optimality is that the interfaces hide information. For example, based on interface $\mathbf{I}_1^3$ it cannot be determined whether part type 1 or part type 2 is being produced until the operation is finished. Since the finishing events

Table 2
Details of FMS example

Step	Automaton Built	# of States (# of Transitions)
1	$\mathbf{G}_1^3 = \mathbf{C2}\|\mathbf{R1}\|\mathbf{Lathe}$	24(92)
	$\mathbf{E}_1^3 = \mathbf{B2}\|\mathbf{B4}$	8(22)
	$\mathbf{G}_2^3 = \mathbf{C3}\|\mathbf{PM}$	6(14)
	$\mathbf{E}_2^3 = \mathbf{B8}$	3(4)
	$\mathbf{G}_1^2 = \mathbf{AM}$	3(4)
	$\mathbf{E}_1^2 = \mathbf{B6}\|\mathbf{B7}$	6(14)
	$\mathbf{G}_2^2 = \mathbf{R2}\|\mathbf{Mill}$	6(14)
	$\mathbf{E}_2^2 = \mathbf{B3}$	3(4)
	$\mathbf{G}_1^1$	empty
	$\mathbf{E}_1^1 = \mathbf{B9}$	2(2)
2	$\Sigma_{R_1^3} = \{33\}$ $\Sigma_{A_1^3} = \{30, 38\}$	
	$\Sigma_{R_2^3} = \{71\}$ $\Sigma_{A_2^3} = \{74\}$	
	$\Sigma_{R_1^2} = \{61\}$ $\Sigma_{A_1^2} = \{64\}$	
	$\Sigma_{R_2^2} = \{91\}$ $\Sigma_{A_2^2} = \{94\}$	
3	$\mathbf{I}_1^3$	2(3)
	$\mathbf{I}_2^3$	2(2)
	$\mathbf{I}_1^2$	2(2)
	$\mathbf{I}_2^2$	2(2)
4	$\mathbf{Z}_1^3 = \mathbf{G}_1^3\|\mathbf{E}_1^3\|\mathbf{I}_1^3$	36(65)
	$\mathbf{S}_1^3$	11(12)
	$\mathbf{Z}_2^3 = \mathbf{G}_2^3\|\mathbf{E}_2^3\|\mathbf{I}_2^3$	6(6)
	$\mathbf{S}_2^3$	6(6)
	$\mathbf{Z}_2^2 = \mathbf{G}_2^2\|\mathbf{E}_2^2\|\mathbf{I}_2^2$	6(6)
	$\mathbf{S}_2^2$	6(6)
	$\mathbf{Z}_1^2 = \mathbf{G}_1^2\|\mathbf{E}_1^2\|\mathbf{I}_1^2\|\mathbf{I}_1^3\|\mathbf{I}_2^3$	60(135)
	$\mathbf{S}_1^2$	13(19)
	$\mathbf{Z}_1^1 = \mathbf{G}_1^1\|\mathbf{E}_1^1\|\mathbf{I}_1^2\|\mathbf{I}_2^2$	8(12)
	$\mathbf{S}_1^1$	6(8)

30 and 38 are uncontrollable, this means that module $\mathbf{H}_1^2$ will not accept two parts at the same time, even if they are parts of a different type. A modification to the design of the interfaces, and possibly the system models themselves, could increase the permissiveness of the resulting control. The use of the low data events of [16] may also help.

6.2 Complexity discussion

For a two-level interface-based architecture, efficient algorithms for the verification of properties [15] [24] and the synthesis of component supervisors [4] [24] have been developed. For each module with its interfaces, these algorithms have complexity that is polynomial in the number of states and events. Note that verification of the Point 4 of the multi-level weak interface consistency definition has complexity that is cubic in the number of states of the involved automata while verification of the Point 4 of the multi-level interface consistency definition is quadratic. Since the high-level module is composed with all of its interfaces at once, it is in most cases the factor limiting how large of a system can be constructed and verified. It has been demonstrated that an interface-based approach is often worthwhile in terms of complexity savings if the interfaces are at least an order of magnitude smaller than their corresponding low-level modules [20].

Since it is required that the low-level modules be completely disjoint from one another, if the global system is made larger, it is often the case that the high-level module will grow and the number of low-level modules will increase. Therefore, the scalability of a two-level architecture is limited since the number of states of a synchronous composition grows exponentially with the number of components. This is where the advantage of a multi-level architecture becomes apparent. Figure 9 illustrates by thicker solid lines a possible partitioning of a larger version of the FMS example used in the previous section for a two-level architecture. In the example of the previous section, the proposed two-level partitioning resulted in a high-level module consisting of four automata and three low-level modules. For the expanded system of Fig. 9, the high-level module H has grown to include thirteen automata and the number of low-level modules L_j has increased to six. Note that this larger example is hypothetical and its details have not actually been developed.

For the multi-level architecture proposed in this paper, we have not yet developed efficient algorithms for the verification of properties and have not yet implemented algorithms for the synthesis of component supervisors. We believe, however, that algorithms can be implemented that will have polynomial complexity in the number of states and events of a given module and its interfaces just like in the two-level case.

As stated earlier, the true advantage of the multi-level architecture is its scaling properties. We have argued that, in the two-level case, as the global system grows the high-level module will grow and the number of low-level modules will increase. In the multi-level case, however, it is possible to limit the size of the modules and the number of corresponding low-level components by increasing the number of levels in the hierarchy. Therefore, we can in many cases put a bound on the number of interfaces that any given module must be analyzed with respect to at once. In terms of complexity, this means the number of states grows approximately linearly with the number of components in the multi-level case, while it had grown exponentially in the two-level case. Furthermore, if a processor is available for each module, parallel computation will allow the global system to be verified/synthesized with the speed of a single module, where the multi-level approach has allowed the global system to be partitioned into smaller modules. Considering the FMS example from the previous section with the multi-level partitioning shown in Fig. 4, the most automata in a given module was five and the maximum number of interfaces for a given module was three. For the larger FMS example of this section, Fig. 9 shows by dotted lines a possible partitioning for the multi-level architecture. For this partitioning, the most automata in a given module is also five, while the maximum number of interfaces for a single module has increased to four. Here one can see the size of the individual modules H_j^i with their interfaces has stayed roughly the same, even though the global system has grown significantly. The smaller modules provided by the multi-level approach are also advantageous in that they make the system easier to understand and reconfigure. If the components of the global system are tightly coupled, the number of modules and levels of hierarchy may be limited. In the worst case, the complexity will approach that of the monolithic solution.

7 Conclusion

The main contribution of this paper is the introduction of requirements for a multi-level interface-based architecture by which global controllability and nonblocking can be verified locally. This generalized architecture is an improvement over the special two-level case of [17] [19] [20] because in many cases it enables much larger systems to be addressed by an interface-based approach to control. Specifically, as the number of components in a system increases, the complexity in many instances will grow linearly if more levels are added to the architecture. If the architecture is limited to two levels, the complexity will grow exponentially. An increase in the number of levels in the hierarchy, however, can result in less permissive control. A secondary contribution of this paper is the relaxation of the interface consistency requirements that allows some systems to be modeled more compactly, even with the two-level approach. This

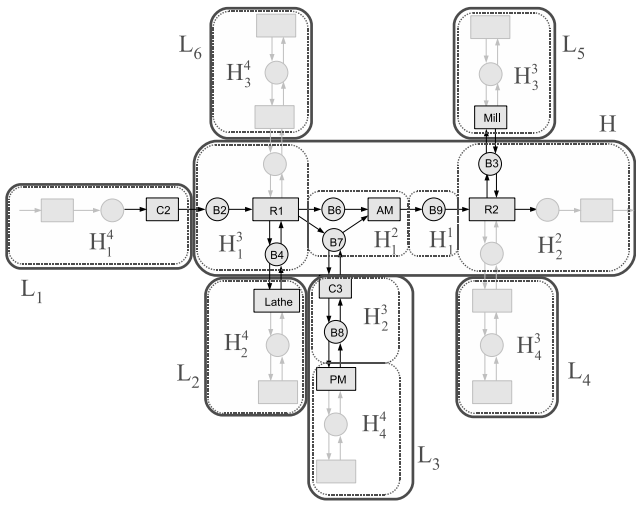

Fig. 9. Extended FMS example with two-level and multi-level partitioning

compactness, however, may come at the cost of introducing other complications.

The final contribution of this paper is the development of a supervisor synthesis method for the multi-level architecture. Building on the work of [18], we show that these supervisors are maximally permissive with respect to a given specification and set of interfaces and can be constructed using automata-based methods.

Some directions for future work include attempting to further generalize the hierarchical interface-based architecture. Specifically, it could be useful if conditions were found under which a single "low-level" module could interact with more than one "high-level" module. Also, it would be useful if an explicit algorithm could be developed for the construction of interfaces. Some preliminary results in this direction have been presented in [12]. So far, interfaces have been constructed based on designer understanding of the system. A final direction for this work would be to develop and implement the algorithms for the verification of the requirements of the multi-level architecture and for the synthesis of supervisors in the multi-level case.

References

[1] 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*, 12(3):387–401, May 2004.

[2] J. Carrol and D. Long. *Theory of Finite Automata with an Introduction to Formal Languages*. Prentice-Hall International, USA, 1989.

[3] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, New York, NY, 2^{nd} edition, 2007.

[4] P. Dai. Synthesis method for hierarchical interface-based supervisory control. Master's thesis, Dept. of Computing and Software, McMaster University, Hamilton, Ont, 2006.

[5] L. de Alfaro and T. A. Henzinger. Interface automata. In *Foundations of Software Engineering*, pages 109–120, 2001.

[6] L. de Alfaro and T. A. Henzinger. Interface theories for component-based design. In *Embedded Software*, pages 148–165. Springer-Verlag, 2001.

[7] M. H. de Queiroz, J. E. R. Cury, and W. M. Wonham. Multitasking supervisory control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Application*, 15(4):375–395, 2005.

[8] E. W. Endsley and D. M. Tilbury. Modular verification of modular finite state machines. In *Proc. 43rd IEEE Conf. Decision & Control*, pages 972–979, The Bahamas, 2004.

[9] L. Feng and W. M. Wonham. Computationally efficient supervisor design: Abstraction and modularity. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 3–8, Ann Arbor, USA, 2006.

[10] H. Flordal and R. Malik. Modular nonblocking verification using conflict equivalence. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 100–106, Ann Arbor, USA, 2006.

[11] H. Flordal and R. Malik. Supervision equivalence. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 155–160, Ann Arbor, USA, 2006.

[12] R. C. Hill. *Modular Verification and Supervisory Controller Design for Discrete-Event Systems Using Abstraction and Incremental Construction*. PhD thesis, University of Michigan, Ann Arbor, USA, 2008. Available at http://www-personal.umich.edu/∼rchill/.

[13] R. C. Hill, J. E. R. Cury, M. H. de Queiroz, D. M. Tilbury, and S. Lafortune. Hierarchical interface-based supervisory control with multiple levels. Technical Report CGR-09-04, Control Group, University of Michigan, Ann Arbor, USA, April 2009.

[14] R. C. Hill and D. M. Tilbury. Incremental hierarchical construction of modular supervisors for discrete-event systems. *Int. J. of Control*, 81(9):1364–1381, 2008.

[15] R. J. Leduc. *Hierarchical Interface-based Supervisory Control*. PhD thesis, University of Toronto, Toronto, Canada, 2002.

[16] R. J. Leduc. Hierarchical interface-based supervisory control with data events. In *Proc. 46th IEEE Conf. Decision & Control*, pages 5910–5917, New Orleans, USA, 2007.

[17] R. J. Leduc, B. A. Brandin, M. Lawford, and W. M. Wonham. Hierarchical interface-based supervisory control–part I: Serial case. *IEEE Transactions on Automatic Control*, 50(9):1322–1335, 2005.

[18] R. J. Leduc and P. Dai. Synthesis method for hierarchical interface-based supervisory control. In *Proc. American Control Conf.*, pages 4260–4267, New York, USA, 2007.

[19] R. J. Leduc, M. Lawford, and P. Dai. Hierarchical interface-based supervisory control of a flexible manufacturing system. *IEEE Transactions on Control Systems Technology*, 14(4):654–668, 2006.

[20] R. J. Leduc, M. Lawford, and W. M. Wonham. Hierarchical interface-based supervisory control–part II: Parallel case. *IEEE Transactions on Automatic Control*, 50(9):1336–1348, 2005.

[21] P. N. Pena, J. E. R. Cury, and S. Lafortune. Testing modularity of local supervisors: An approach based on

abstractions. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 107–112, Ann Arbor, USA, 2006.

[22] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proc. of IEEE, Special Issue on Discrete Event Dynamic Systems*, 77(1):81–98, January 1989.

[23] K. Schmidt, T. Moor, and S. Perk. A hierarchical architecture for nonblocking control of discrete event systems. In *Mediterranean Conference on Control and Automation*, 2005.

[24] R. Song and R. J. Leduc. Symbolic synthesis and verification of hierarchical interface-based supervisory control. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 419–426, Ann Arbor, USA, 2006.

[25] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *Siam J. of Control Optim.*, 25(3):637–659, 1987.