Hierarchical Interface-Based Supervisory Control—Part I: Serial Case

Ryan J. Leduc, *Member, IEEE*, Bertil A. Brandin, *Member, IEEE*, Mark Lawford, *Member, IEEE*, and W. M. Wonham, *Life Fellow, IEEE*

***Abstract*—In this paper, we present a hierarchical method that decomposes a system into two subsystems, and restricts the interaction of the subsystems by means of an interface. We present definitions for two types of interfaces [represented as discrete-event systems (DESs)], and define a set of interface consistency properties that can be used to verify if a DES is nonblocking and controllable. Each clause of the definitions can be verified using only one of the two subsystems; thus, the complete system model never needs to be constructed, offering potentially significant savings in computational effort. Additionally, the development of clean interfaces facilitates reuse of the component subsystems. Finally, we examine a simple example to illustrate the method.**

***Index Terms*—Automata, discrete-event systems (DESs), formal methods, hierarchical systems, interfaces.**

I. Introduction

IN THE AREA of discrete-event systems (DESs), two common tasks are to verify that a composite system, based on a cartesian product of subsystems, is: i) nonblocking and ii) controllable. The main obstacle to performing these tasks is the combinatorial explosion of the product state space. Although many methods have been developed to deal with this problem, large-scale systems are still problematic, particularly for verification of nonblocking. In this paper, our goal is to develop an architecture for DES that supports a scalable method for the design and verification of nonblocking supervisory controllers.

For inspiration, we first turn to digital logic circuits. Complexity is routinely managed by designers of microprocessors for personal computers. These circuits are hierarchical in nature, and are designed by using interfaces to limit the interaction between different levels of the hierarchy. A complex system is designed by first creating basic components, then adding an interface that encapsulates the behavior of the component, and provides an abstract model of the component's operation with a well-defined method of interacting with the component. These components are combined to create a new, more complex, component, with its own interface. At each step in the process, a component is treated as a black box, and the designer only utilizes the component's interface. At no time is the designer allowed to modify the inner workings of a component or to "look below" the interface. This keeps the complexity at each level manageable. Although designing a single-level ("flat") circuit might generally be more efficient in terms of timing delay and number of gates used, it would not be a practical or efficient design approach for large circuits as the amount of detail would become prohibitive.

In software program design, similar ideas are at work. To deal with the complexity of large scale systems, software engineers have long advocated the decomposition of software into modules (components) that interact via well-defined interfaces (e.g., [1] and [2]). This approach is referred to as *information hiding*. Some of its advantages are given here.

- Limits complexity by hiding unnecessary detail behind interfaces.
- Promotes independent development as once the module interfaces are defined, each module can be designed separately.
- Provides a high degree of changeability by encapsulating the behavior of a module. The implementation of a module can be changed without affecting the modules that use it since they are not permitted to reflect the inner details or interact with the internals of the module.
- Provides a high degree of comprehensibility. Because information is localized in modules and unnecessary details are hidden by the interface, it is much easier to understand a module.
- Provides a well defined hierarchical structure. This structure guarantees that we can remove the upper levels of our hierarchy, and what is left can be reused in another application.

Both the black box methodology of circuits and the information hiding approach of software, manage the complexity of designing large scale systems by restricting the design to render it easier to analyze, maintain and conceptualize. Our goal in this paper is to develop a similar architectural approach for DES. The method utilizes well defined interfaces between components that are themselves DES. These "interface DES" provide a structure allowing local checks to guarantee global properties such as controllability and nonblocking. In order to achieve our ultimate goal of scalability, we restrict the permissible system architectures and sacrifice global maximal permissiveness to obtain a (generally) suboptimal solution, but one that is more tractable.

A. Literature Review

Researchers in supervisory control have recently begun to advocate interface based architectural solutions to dealing

Manuscript received August 27, 2003. Recommended by Associate Editor R. Boel.

R. J. Leduc and M. Lawford are with the Department of Computing and Software, McMaster University, Hamilton, ON L8S 4K1, Canada (e-mail: leduc@mcmaster.ca; lawford@mcmaster.ca).

B. A. Brandin is with Siemens Medical Solutions Health Services, Milano 20126, Italy (e-mail: bertil.brandin@siemens.com).

W. M. Wonham is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: wonham@control.utoronto.ca).

Digital Object Identifier 10.1109/TAC.2005.854586

with complexity [3], [4]. These approaches develop interfaces between components to provide structure that guarantees global properties such as controllability [4] or nonblocking [3]. In this paper, we present an interface-based hierarchical method, called hierarchical interface-based supervisory control (HISC), to verify if a system is nonblocking and controllable. Early results of this work can be found in [3], [5]. While, in general, the method can decompose the system into multiple "parallel" subsystems (see [6] and [7]), for the purposes of this paper (Part I of II) we restrict our attention to the special case where the system is split into two subsystems that interact via a single interface DES. The most significant feature that distinguishes this work from [4] is the results on nonblocking.

In [8], interface automata are used to model software components and verify their compatibility. This work has independently derived conditions for software component interface compatibility that are similar to the interface consistency properties presented in Section III-A. In [8], automata representing component interfaces are directly composed to produce the interface of the new composite component and a refinement relation is developed to aid in refining a component interface specification into an implementation. There is no explicit concept of control, though implicitly component inputs are considered uncontrollable and the component outputs are effectively controllable. In contrast we propose an interface automaton that mediates communication between the components in order to decompose the verification of global nonblocking and controllability into "local" checks on each of the components and their interface.

Related work by Fabian *et al.* [9], [10] applied object-oriented concepts in the design of DES control software, and extended supervisory control theory to the nondeterministic supervisors which that approach required. Later, Shayman *et al.* [11] introduced the concept of control and observation masks to encapsulate process logic. These approaches have two disadvantages relative to interface based supervisory control: i) they do not address issues related to nonblocking and ii) they require a more complex mathematical setting than the deterministic automata with synchronous product operator that is commonly employed in supervisory control theory. By using interface DES to regulate subsystem interaction, we are able to impose architecture without change to the standard DES setting.

One of the earliest and most useful methods designed to handle the combinatorial explosion of the product state space that results from systems composed of interacting subsystems is *modular control* [12]–[15]. This method involves designing multiple supervisors as opposed to a centralized supervisor, each supervisor implementing a portion of the control specification. While the method scales well in practice for the verification of controllability (see e.g., [16] and [17]), verifying nonblocking of the closed loop system is still a problem.

In *decentralized control* [18]–[23], local supervisors, with only partial observations of the plant, are designed as a group to implement a global specification. While this is an effective method to design distributed controllers, it still requires the computation of the synchronous product of all of the plant subcomponents (the composite plant) and thus offers no computational savings over a centralized solution.

One way to improve the scalability of modular and decentralized schemes is to exploit the existing architecture of the system. In [24], the concept of a specification that is separable over the component subsystems is introduced and shown to be necessary and sufficient for a decentralized control scheme to exist that optimally meets the specification. The work does not consider nonblocking supervision. These results are extended to a more general architecture in [25] that deals with nonblocking by detecting potential blocking states locally and then backtracking globally to determine their reachability. The structure associated with the event sets of subsystems is exploited in [14] to obtain a reduction in complexity for the nonconflicting check of modular control. Similarly the standard controllability definition has been refined and localized in [26] to check on a per subplant basis only those uncontrollable events that can occur locally.

Another approach is embodied by *vector DES* (VDES) [12], [27], [28] and *Petri nets* (PNs) [29]–[31]. These state based methods make use of the algebraic regularity inherent in certain systems. They are used when the state of the system can be represented as a vector of integers, whose components are incremented or decremented by events. These methods are primarily useful for systems with a high degree of regularity that lend themselves to vector representation. However, the VDES/PN models are not well adapted to the synthesis or verification of nonblocking controllers without first converting the models to automata by means of the reachability graph [32].

A promising approach is the development of a multilevel hierarchy. In order to aid in classification, we make a distinction between structural multilevel hierarchies with explicit mechanisms (modeling constructs) to facilitate hierarchy (e.g., [33]–[36]) as opposed to aggregate (bottom up) multilevel hierarchies which we will discuss later. In structural multilevel hierarchies, plants and supervisors are modeled as multilevel structures similar to automata, except that certain states at a given level can be expanded into a more detailed lower level model. Although [35] allowed a system to be represented hierarchically using cartesian products (AND superstates) or disjoint unions (OR superstates), AND states had to be converted to OR states using the synchronous product before computations could be effectively performed. Similarly, [34] was restricted to using only OR states. Both approaches could verify controllability, but did not address nonblocking. Recently, these limitations have been overcome by Ma *et al.* [36] who, with the use of *binary decision diagrams* (BDDs) [37], has been able to verify controllability and nonblocking for a system on the order of 10^{24} states.

The next approach of interest is the model aggregation methods [38]–[47]. In these approaches, aggregate models are derived from low-level models by using either state-based or language-based aggregation methods. Although this approach can be effective in constructing high-level models with reduced state–spaces, they have some drawbacks.

- In hierarchical methods such as [46], [47], and [43], there is no direct connection between control actions at the high-level, and at lower levels. To create an implementation, a control action at the high-level may need to be "interpreted" as equivalent control action(s) at the low-level.

- Aggregate models must be constructed sequentially from the bottom up, starting from the lowest level; thus, a given level cannot be constructed and verified in parallel with the levels below it, making a distributed design process difficult.
- The DES methods provide necessary and sufficient conditions for checking controllability, and in many cases nonblocking, using the aggregate models. While this is desirable, it causes the individual levels to be tightly coupled; a change made to the lowest level may require that all aggregate models and results have to be re-evaluated. In contrast, the sufficient conditions of interface based supervisory control that we develop allow us to design and verify levels independently, ensuring that a change to one level of the hierarchy will not impact the others. This independence comes at the cost of possible false negatives forcing an overly conservative design.

We also note the related work in hybrid systems of Moor *et al.* [48] who have developed a multilevel aggregation approach inspired by [46], [47]. This new approach is different as they use an input–output structure to represent both time and event driven system dynamics, allowing them to verify both controllability and nonblocking results.

In contrast to the majority of approaches which apply mathematical techniques to produce aggregate models of an existing system, our method of restricting component interaction to well defined interfaces provides a design heuristic to guarantee scalability by construction.

The last approach we discuss is the use of symbolic methods to represent the transition structures underlying DES [49]. Zhang *et al.* [50], [51] have recently developed algorithms that use *integer decision diagrams* (an extension of BDDs) to verify centralized DES systems on the order of 10^{23} states. That work builds on results of symbolic model checking [52], [53] that have successfully used BDDs to handle systems of similar size. The ability of such symbolic methods to handle large state spaces is highly dependent upon finding a variable ordering for the data structures that can exploit the system's regularity. In [50] and [51], it was found that the symbolic methods were most effective when the interconnection matrix of system components was diagonalizable (i.e., interaction of subsystems was limited to "adjacent" components). By forcing the interaction of subsystems to pass through interface DES, our interface based supervisory control should have the effect of limiting component interaction in just the right way to allow the effective application of symbolic techniques.

Finally, we note that the interface DES that support our system architecture differ from the "interface processes" employed in compositional model checking [54]. In the latter, an interface process is an aggregate model that is used as a replacement for a particular subsystem to produce a reduced-state model that facilitates verification. For example, let $\mathbf{P}_i, i = 1, 2$ be subsystem models and ψ be the temporal logic formula of interest. In order to verify that $\mathbf{P}_1 \,||\, \mathbf{P}_2 \models \psi$ by compositional model checking, $\mathbf{P}_2$ might be replaced by an aggregate "interface process" $\mathbf{A}_2$ such that if $\mathbf{P}_1 \,||\, \mathbf{A}_2 \models \psi$ then $\mathbf{P}_1 \,||\, \mathbf{P}_2 \models \psi$.

In the following sections, we first describe the general setting and provide preliminary definitions. We then present a set of (local) consistency requirements that the interface and subsystems must satisfy to guarantee global nonblocking and controllability. We then provide a small illustrative example.

II. DES Preliminaries

Ramadge–Wonham supervisory control [55], [56], [12] provides a theoretical framework for the control of DESs, systems that are discrete in space and time. For a detailed exposition of DES, see [12]. Here, we present a summary of the terminology that we use in this paper.

Let Σ be a finite set of distinct symbols (*events*), and Σ^* be the set of all finite sequences of events, including ϵ, the *empty string*. Let $L \subseteq \Sigma^*$ be a *language* over Σ. A string $t \in \Sigma^*$ is a prefix of $s \in \Sigma^*$ (written $t \leq s$) if $s = tu$, for some $u \in \Sigma^*$. The *prefix closure* of a language $L \subseteq \Sigma^*$ (denoted $\bar{L}$) is defined as

$$\bar{L} = \{t \in \Sigma^* \,|\, t \leq s \text{ for some } s \in L\}.$$

Let $\mathrm{Pwr}(\Sigma)$ denote the power set of Σ. For language L, the eligibility operator $\mathrm{Elig}_L \colon \Sigma^* \to \mathrm{Pwr}(\Sigma)$ is defined as below. Let $s \in \Sigma^*$. Then

$$\mathrm{Elig}_L(s) := \{\sigma \in \Sigma \,|\, s\sigma \in L\}.$$

A DES automaton is represented as a 5-tuple

$$\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$$

where Y is the state set, Σ is the event set, the partial function $\delta \colon Y \times \Sigma \to Y$ is the transition function, y_o is the initial state, and Y_m is the set of marker states. We will also use the notation $\Sigma_{\mathbf{G}}$ as a shorthand for the event set that DES $\mathbf{G}$ is defined over. This is an easy way to refer to the alphabet given in the 5-tuple definition of $\mathbf{G}$, particularly in situations when it is not explicitly stated (for example, in the case of a DES created by the synchronous product operator below). The function δ is extended to $\delta : Y \times \Sigma^* \to Y$ in the natural way. The notation $\delta(y, s)!$ means that δ is defined for $s \in \Sigma^*$ at state y.

For DES $\mathbf{G}$, the language generated is denoted by $L(\mathbf{G})$, and is defined to be

$$L(\mathbf{G}) := \{s \in \Sigma^* \,|\, \delta(y_o, s)!\}$$

The marked behavior of $\mathbf{G}$, $L_m(\mathbf{G})$, is defined as follows:

$$L_m(\mathbf{G}) := \{s \in L(\mathbf{G}) \,|\, \delta(y_o, s) \in Y_m\}.$$

Let $\Sigma = \Sigma_1 \cup \Sigma_2$, $L_1 \subseteq \Sigma_1^*$, and $L_2 \subseteq \Sigma_2^*$. For $i = 1, 2$, $s \in \Sigma^*$, and $\sigma \in \Sigma$, we define the *natural projection* $P_i \colon \Sigma^* \to \Sigma_i^*$ according to:

$$P_i(\epsilon) = \epsilon \quad P_i(\sigma) = \begin{cases} \epsilon, & \text{if } \sigma \notin \Sigma_i \\ \sigma, & \text{if } \sigma \in \Sigma_i \end{cases}$$
$$P_i(s\sigma) = P_i(s)P_i(\sigma).$$

The synchronous product of languages L_1 and L_2, denoted $L_1 \,||\, L_2$, is defined to be

$$L_1 \,||\, L_2 = P_1^{-1}(L_1) \cap P_2^{-1}(L_2)$$

where P_i^{-1}: $\mathrm{Pwr}(\Sigma_i^*) \rightarrow \mathrm{Pwr}(\Sigma^*)$ is the inverse image function of P_i (see, e.g., [12]).

The synchronous product of DES $\mathbf{G}_1 = (Y_1, \Sigma_1, \delta_1, y_{o_1}, Y_{m_1})$ and $\mathbf{G}_2 = (Y_2, \Sigma_2, \delta_2, y_{o_2}, Y_{m_2})$, denoted $\mathbf{G}_1||\mathbf{G}_2$, is defined to be a reachable DES $\mathbf{G}$ with the properties[1]

$$L_m(\mathbf{G}) = L_m(\mathbf{G}_1) \,||\, L_m(\mathbf{G}_2)$$
$$L(\mathbf{G}) = L(\mathbf{G}_1) \,||\, L(\mathbf{G}_2)$$
$$\text{and event set } \Sigma = \Sigma_1 \cup \Sigma_2.$$

For DES, the two main properties we want to check are *nonblocking* and *controllability*.

Definition 1: A DES $\mathbf{G}$ is said to be *nonblocking* if the following is true:

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

To control the plant, we define a *supervisor*. As this paper focuses on verification and not synthesis, we will use the terms supervisor and specification interchangeably as we require that all specifications be controllable. The supervisors are represented as automata and defined as follows:

$$\mathbf{S} = (X, \Sigma_S, \xi, x_o, X_m).$$

In this paper, we will use the synchronous product operator to specify the closed-loop behavior as this makes data entry easier and less error prone, particularly when using a graphical editor to specify and display the supervisor. This means that the behavior of a plant $\mathbf{G}_1 = (Y_1, \Sigma_1, \delta_1, y_{o_1}, Y_{m_1})$ under the control of a supervisor $\mathbf{S}$ is

$$\mathbf{System} := \mathbf{G}_1 \,||\, \mathbf{S}.$$

We now present a formal definition for controllability. We adopt the standard partition $\Sigma = \Sigma_u \dot\cup \Sigma_c$ splitting our alphabet into *uncontrollable* and *controllable events*. We then need to define the event set Σ, the natural projections P_1 and P_S, and languages $\mathcal{L}_{G_1}$ and $\mathcal{L}_S$ as follows:

$$\Sigma := \Sigma_1 \cup \Sigma_S \quad P_1 \colon \Sigma^* \rightarrow \Sigma_1^* \quad P_S : \Sigma^* \rightarrow \Sigma_S^*$$
$$\mathcal{L}_{G_1} := P_1^{-1} L(\mathbf{G}_1) \quad \mathcal{L}_S := P_S^{-1} L(\mathbf{S}).$$

Definition 2: A supervisor $\mathbf{S}$ is *controllable* for a plant $\mathbf{G}_1$ if

$$\mathcal{L}_S \Sigma_u \cap \mathcal{L}_{G_1} \subseteq \mathcal{L}_S$$

or, equivalently

$$(\forall s \in \mathcal{L}_{G_1} \cap \mathcal{L}_S)\, \mathrm{Elig}_{\mathcal{L}_{G_1}}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\mathcal{L}_S}(s).$$

[1]We are overloading the || operator here by using it for both languages and DES, but this should not cause confusion as the choice of arguments will make the meaning clear.

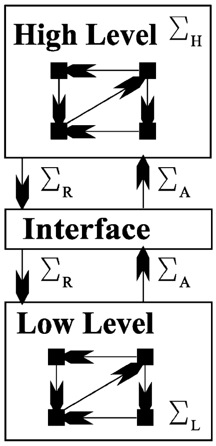


Fig. 1. Interface block diagram.

III. Serial Case Setting

With the serial case of *hierarchical interface-based supervisory control*, we propose essentially a master–slave system, where a *high-level subsystem* sends a command to a *low-level subsystem*, which then performs the indicated task and sends back a reply. Fig. 1 shows conceptually the structure of the system.

To allow the system to be designed, maintained and verified on a component-wise basis, we impose an interface between the two subsystems that limits their interaction and knowledge of each other. The goal is to be able to work with each subsystem individually, requiring no information about the other subsystem beyond that provided by the interface.

To capture the restriction of the flow of information imposed by the *interface*, we split the alphabet of the system (Σ) into four pairwise disjoint alphabets: $\Sigma_H, \Sigma_L, \Sigma_R$, and Σ_A. The events in Σ_H are called *high-level events* and the events in Σ_L *low-level events* as these events appear only in the high-level and low-level models, respectively.

The alphabets Σ_R and Σ_A are called collectively *interface events*. These events are common to both levels of the hierarchy and represent communication between the two subsystems. Events in Σ_R are *request events* and represent commands sent from the high-level subsystem to the low-level subsystem. The events in Σ_A are *answer events* and represent the low-level's responses to the *request events* (high-level commands). Fig. 1 shows conceptually the flow of information in our setting.

In the remainder of this section, we will first define two types of interfaces, and then some useful terminology and notation.

A. Interface Definitions

In this section, we present two interface definitions: *star interfaces* and *command-pair interfaces*. As we will see later, star interfaces are a special case of command-pair interfaces.

We start by describing a star interface as it has a regular structure and is thus easy to construct. To define a star interface, the designer selects a set of request events, and then for each request event, the designer defines a set of answer events. In essence, the designer defines a map $\mathbf{Answer}\colon \Sigma_R \rightarrow \mathrm{Pwr}(\Sigma_A)$. For $\rho \in \Sigma_R, \mathbf{Answer}(\rho)$ is the set of possible answers (referred to as

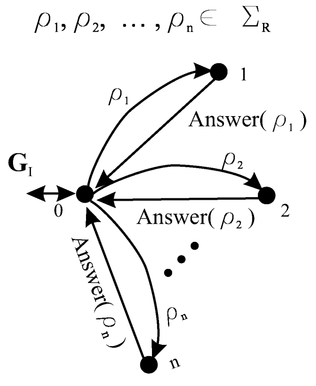


Fig. 2. Star interface.

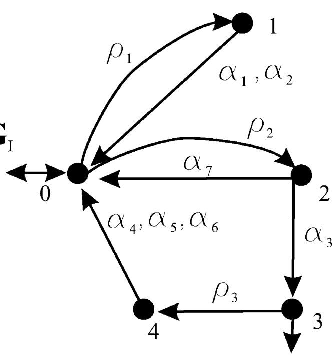


Fig. 3. Example command-pair interface.

the *answer set*) the low-level subsystem could provide after receiving request ρ. We also add the constraints that the low-level subsystem must provide at least one response for each request it receives, and that Σ_A does not contain any unused events. Finally, we see in Fig. 2 how a star interface, with $n = |\Sigma_R|$, is expressed as a DES. The required structure is given by DES $\mathbf{G}_I$. We also require that the event set of $\mathbf{G}_I$ be set to $\Sigma_R \dot{\cup} \Sigma_A$.

We now define command-pair interfaces which were designed as a generalization of star interfaces. A key difference is that the "star" shape is no longer required. A command-pair interface still always has a request event followed by an answer event, but it can now contain additional state information. For example, in Fig. 2 all possible request events are defined at the initial state. When an answer event has occurred, it always returns the star interface to the initial state, and thus the same choice of potential request events. With a command-pair interface we can have a DES structure as illustrated in Fig. 3. Request events ρ_1 and ρ_2 might represent the regular behavior of the system, while α_3 and ρ_3 represent breakdown and repair of the system. A command-pair interface allows the flexibility of only having the repair event eligible after a breakdown.

For the remainder of this work, when we refer to an interface we will mean explicitly a command-pair interface, and we will use the two synonymously. We define a command-pair interface as follows.

Definition 3: A DES $\mathbf{G}_I = (X, \Sigma_R \dot{\cup} \Sigma_A, \xi, x_o, X_m)$ is a *command-pair interface* if the following conditions are satisfied:

A) $L(\mathbf{G}_I) \subseteq \overline{(\Sigma_R.\Sigma_A)^*}$;
B) $L_m(\mathbf{G}_I) = (\Sigma_R.\Sigma_A)^* \cap L(\mathbf{G}_I)$.

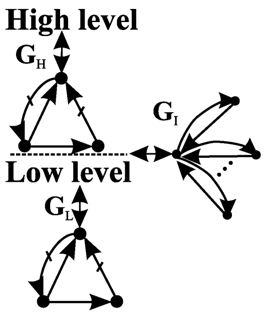


Fig. 4. Two-tiered structure of system.

Note $\mathbf{G}_I$'s event set is restricted to request and answer events and that the two sets must be disjoint. Taken together, **points A)** and **B)** imply that request event transitions are only defined at marker states and that there are no answer events defined at marker states. **Point A)** says that in the language of $\mathbf{G}_I$, a request event always occurs first and then request and answer events alternate. Finally, **point B)** implies that the marked language of $\mathbf{G}_I$ consists of the empty string, and strings that end in an answer event. We observe that star interfaces are a special case of command-pair interfaces.

B. Terminology and Notation

We now present some terminology and notation that will be useful in simplifying proofs. For our setting, we assume the high-level subsystem is modeled by DES $\mathbf{G}_H$ (defined over event set $\Sigma_H \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A$), the low-level subsystem by DES $\mathbf{G}_L$ (defined over event set $\Sigma_L \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A$), and the interface by DES $\mathbf{G}_I$. Also, the term *high-level* will mean the DES $\mathbf{G}_H \,||\, \mathbf{G}_I$, and the term *low-level* the DES $\mathbf{G}_L \,||\, \mathbf{G}_I$. The overall structure of the system is displayed in Fig. 4.

We next assume that the alphabet partition is specified by $\Sigma := \Sigma_H \dot{\cup} \Sigma_L \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A$ and define the *flat system* $\mathbf{G}$ as below. By flat system, we refer to the equivalent DES that would represent our system if we ignored the interface structure

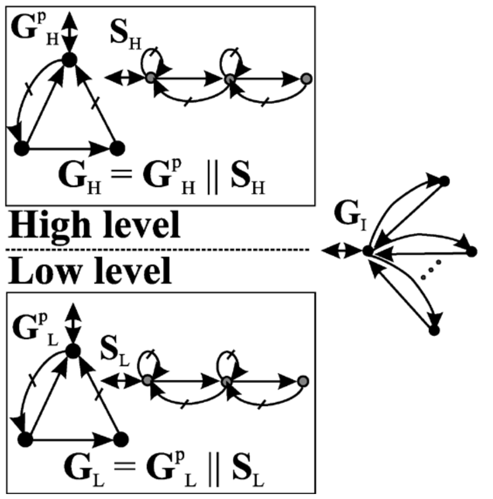
$$\mathbf{G} = \mathbf{G}_H \,||\, \mathbf{G}_L \,||\, \mathbf{G}_I.$$

To simplify the notation in proofs, we introduce the following event sets, natural projections, and useful languages:

$$\begin{aligned}
\Sigma_I &:= \Sigma_R \dot{\cup} \Sigma_A \quad P_{IH}\colon \Sigma^* \to \Sigma_{IH}^* \\
\Sigma_{IH} &:= \Sigma_H \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A \quad P_{IL}\colon \Sigma^* \to \Sigma_{IL}^* \\
\Sigma_{IL} &:= \Sigma_L \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A \quad P_I\colon \Sigma^* \to \Sigma_I^* \\
\mathcal{H} &:= P_{IH}^{-1}(L(\mathbf{G}_H)), \quad \mathcal{H}_m := P_{IH}^{-1}(L_m(\mathbf{G}_H)) \subseteq \Sigma^* \\
\mathcal{L} &:= P_{IL}^{-1}(L(\mathbf{G}_L)), \quad \mathcal{L}_m := P_{IL}^{-1}(L_m(\mathbf{G}_L)) \subseteq \Sigma^* \\
\mathcal{I} &:= P_I^{-1}(L(\mathbf{G}_I)), \quad \mathcal{I}_m := P_I^{-1}(L_m(\mathbf{G}_I)) \subseteq \Sigma^*.
\end{aligned}$$

Whereas the representation of the system as given in Fig. 4 (called the *serial subsystem based form*) is useful for verifying nonblocking as it simplifies the notation, it ignores the distinction between plants and supervisors. For controllability, we need to split the subsystems into their plant and supervisor components. We will do so as shown in Fig. 5.

We next define the *high-level plant* to be $\mathbf{G}_H^p$, and the *high-level supervisor* to be $\mathbf{S}_H$ (both defined over event set Σ_{IH}). Similarly, the *low-level plant* and *supervisor* are $\mathbf{G}_L^p$ and $\mathbf{S}_L$



$\mathbf{G}^p_H$
$\mathbf{S}_H$
$\mathbf{G}_H = \mathbf{G}^p_H \parallel \mathbf{S}_H$
High level
Low level
$\mathbf{G}_I$
$\mathbf{G}^p_L$
$\mathbf{S}_L$
$\mathbf{G}_L = \mathbf{G}^p_L \parallel \mathbf{S}_L$

Fig. 5. Plant and supervisior subplant decomposition.

(defined over event set Σ_{IL}). To be consistent with the serial subsystem-based form, we define the following identities for the high and low-level subsystems as below. We will refer to this new representation as the *serial system general form* as the original representation can be recovered from applying these identities

$$\mathbf{G}_H := \mathbf{G}^p_H \parallel \mathbf{S}_H \quad \mathbf{G}_L := \mathbf{G}^p_L \parallel \mathbf{S}_L.$$

We now define our *flat supervisor* and *plant* as well as some useful languages as follows:

$$\begin{aligned}\mathbf{Plant} &:= \mathbf{G}^p_H \parallel \mathbf{G}^p_L \quad \mathbf{Sup} := \mathbf{S}_H \parallel \mathbf{S}_L \parallel \mathbf{G}_I\\ \mathcal{H}^p &:= P^{-1}_{IH} L(\mathbf{G}^p_H), \quad \mathcal{S}_H := P^{-1}_{IH} L(\mathbf{S}_H) \quad \subseteq \Sigma^*\\ \mathcal{L}^p &:= P^{-1}_{IL} L(\mathcal{G}_L), \quad \mathcal{S}_L := P^{-1}_{IL} L(\mathcal{S}_L) \quad \subseteq \Sigma^*.\end{aligned}$$

IV. Serial Interface Consistency, Nonblocking, and Controllable

In this section, we present the interface properties that our system must satisfy to ensure that it interacts with the interface correctly, as well as the nonblocking and controllability requirements each level must satisfy. Together they provide a set of local conditions that can be evaluated using at most one level of our hierarchy at a time. We then present several useful propositions for nonblocking, followed by our main nonblocking result. This is followed by controllability propositions and our main controllability result.

A. Definitions

Our first definition is the *serial level-wise nonblocking* definition. It requires that each level be individually nonblocking.

Definition 4: The system composed of DES $\mathbf{G}_H, \mathbf{G}_L$, and $\mathbf{G}_I$, is said to be *serial level-wise nonblocking* with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, if the following conditions are satisfied:

I) $\overline{\mathcal{H}_m \cap \mathcal{I}_m} = \mathcal{H} \cap \mathcal{I}$;

II) $\overline{\mathcal{L}_m \cap \mathcal{I}_m} = \mathcal{L} \cap \mathcal{I}$.

The next definition states that the system is *serial level-wise controllable* if, for the given distributed supervisor, the high-level supervisor is controllable for the high-level plant combined with the interface (by **III**) and that the low-level supervisor synchronized with the interface is controllable for the low-level plant (**by II**). At first glance, it may appear that the language $\mathcal{I}$ is unnecessary in **III** and could be removed, but doing so would be too restrictive. This can be seen from the example in Section V where not only is the high-level supervisor by itself uncontrollable for the high-level plant, the supremal controllable sublanguage would be the empty set.

Definition 5: The system composed of plant components $\mathbf{G}^p_H$, $\mathbf{G}^p_L$, supervisors $\mathbf{S}_H$, $\mathbf{S}_L$, and interface $\mathbf{G}_I$, is said to be *serial level-wise controllable* with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, if the following conditions are satisfied.

I) The alphabet of $\mathbf{G}^p_H$ and $\mathbf{S}_H$ is Σ_{IH}, the alphabet of $\mathbf{G}^p_L$ and $\mathbf{S}_L$ is Σ_{IL}, and the alphabet of $\mathbf{G}_I$ is Σ_I.

II) $(\forall s \in \mathcal{L}^p \cap \mathcal{S}_L \cap \mathcal{I})\mathrm{Elig}_{\mathcal{L}^p}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\mathcal{S}_L \cap \mathcal{I}}(s)$.

III) $(\forall s \in \mathcal{H}^p \cap \mathcal{I} \cap \mathcal{S}_H)\mathrm{Elig}_{\mathcal{H}^p \cap \mathcal{I}}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\mathcal{S}_H}(s)$.

Finally, we present the *serial interface consistency* definition. It defines the interface properties that our system must satisfy to ensure that it interacts with the interface correctly. It limits the information each level can have about the other, and what assumptions they can make about each other. We will then briefly discuss each property.

Definition 6: The system composed of DES $\mathbf{G}_H, \mathbf{G}_L$, and $\mathbf{G}_I$, is *serial interface consistent* with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, if the following properties are satisfied.

Multilevel Properties

1) The event set of $\mathbf{G}_H$ is Σ_{IH}, and the event set of $\mathbf{G}_L$ is Σ_{IL}.

2) $\mathbf{G}_I$ is a command-pair interface.

High-Level Properties

3) $(\forall s \in \mathcal{H} \cap \mathcal{I})\mathrm{Elig}_{\mathcal{I}}(s) \cap \Sigma_A \subseteq \mathrm{Elig}_{\mathcal{H}}(s)$.

Low-Level Properties

4) $(\forall s \in \mathcal{L} \cap \mathcal{I})\mathrm{Elig}_{\mathcal{I}}(s) \cap \Sigma_R \subseteq \mathrm{Elig}_{\mathcal{L}}(s)$.

5) $(\forall s \in \Sigma^*.\Sigma_R \cap \mathcal{L} \cap \mathcal{I})$

$$\mathrm{Elig}_{\mathcal{L} \cap \mathcal{I}}(s\Sigma^*_L) \cap \Sigma_A = \mathrm{Elig}_{\mathcal{I}}(s) \cap \Sigma_A$$

where $\mathrm{Elig}_{\mathcal{L} \cap \mathcal{I}}(s\Sigma^*_L) := \cup_{l \in \Sigma^*_L} \mathrm{Elig}_{\mathcal{L} \cap \mathcal{I}}(sl)$.

6) $(\forall s \in \mathcal{L} \cap \mathcal{I}) s \in \mathcal{I}_m \Rightarrow (\exists l \in \Sigma^*_L) sl \in \mathcal{L}_m \cap \mathcal{I}_m$.

Prop 1) This property asserts that the high and low-levels can only share request and answer events. This is an information hiding statement. It restricts the high-level subsystem from knowing (and directly affecting) internal details about the low-level subsystem (i.e., to be able to view/disable low-level events) and *vice versa*.

Prop 2) This property states that DES $\mathbf{G}_I$ satisfies the definition of a command-pair interface.

Prop 3) This property asserts that the high-level subsystem $(\mathbf{G}_H)$ must always accept an answer event if the event is eligible in the interface. In other words, the high-level subsystem is forbidden to assume more about when an answer event can occur than what is provided by the interface.

Prop 4) This property asserts that the low-level subsystem ($\mathbf{G}_L$) must always accept a request event if the event is eligible in the interface. In other words, the low-level subsystem is forbidden to assume more about when a request event can occur than what is provided by the interface.

Prop 5) This property asserts that immediately after a request event (some $\rho \in \Sigma_R$) has occurred (and before it is followed by any low-level events), there exist one or more paths via strings in Σ_L^* to each answer event (i.e., all $\alpha \in \mathbf{Answer}(\rho)$, assuming that we are dealing with a star interface) that $\mathbf{G}_I$ says can follow the request event. However, as soon as a single low-level event has occurred, one or more answer events may no longer be reachable.

Prop 6) This property asserts that every string marked by the interface and accepted by the low-level subsystem, can be extended by a low-level string to a string marked by the low-level (both $\mathbf{G}_I$ and $\mathbf{G}_L$).

B. Nonblocking Propositions and Theorem

We will now present **Propositions 1)–5)**, followed by our main nonblocking result. The following propositions perform two tasks: they break down the main theorem into a more manageable size, as well as provide useful results that can be reused in future work.

Our first proposition is the *low-level nonblocking proposition*. It asserts that the low-level is not dependent on high-level events to reach a marker state.

Proposition 1: If the system composed of DES $\mathbf{G}_H, \mathbf{G}_L$, and $\mathbf{G}_I$ is serial level-wise nonblocking and serial interface consistent with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, then

$$(\forall s \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I})(\exists l \in \Sigma_{IL}^*)(sl \in \mathcal{H} \cap \mathcal{L}_m \cap \mathcal{I}_m).$$

Proof: See the proof in [7]. ■

Our next proposition is the *low-level linkage proposition*. It asserts that if the high-level can be driven to a marker state, the low-level can be brought to a marker state by a string containing events that are ignored by the high-level.

Proposition 2: If the system composed of DES $\mathbf{G}_H, \mathbf{G}_L$, and $\mathbf{G}_I$ is serial level-wise nonblocking and serial interface consistent with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, then

$$(\forall s \in \mathcal{L} \cap \mathcal{H}_m \cap \mathcal{I}_m)(\exists l \in \Sigma_L^*) sl \in \mathcal{H}_m \cap \mathcal{L}_m \cap \mathcal{I}_m$$

Proof: See the proof in [7]. ■

We group the next three "construction" propositions together, as each builds upon the previous one. Our first proposition is the *one-step construction proposition*. It asserts that we can use string h as a basis to construct string u by adding low-level events so that the low-level subsystem will accept the request and answer event contained in h. As these events are common to both levels, they must agree on their occurrence.

Proposition 3: If the system composed of DES $\mathbf{G}_H, \mathbf{G}_L$, and $\mathbf{G}_I$ is serial level-wise nonblocking and serial interface consistent with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, then

$$\begin{aligned}&(\forall s \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I})(\forall h \in \Sigma_H^*.\Sigma_R.\Sigma_H^*.\Sigma_A)\\ &\quad sh \in \mathcal{H} \cap \mathcal{I} \Rightarrow (\exists u \in \Sigma^*)(su \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m)\\ &\qquad \wedge (P_{IH}(u) = h).\end{aligned}$$

Proof: See the proof in [7]. ■

Our next proposition is the *inductive construction proposition*. This proposition is different from **Proposition 3)** as that proposition only handled the case that the string h contains exactly one answer event (i.e., only one command-pair), while this proposition allows h to contain one or more answer events (i.e., multiple command-pairs).

Proposition 4: If the system composed of DES $\mathbf{G}_H$, $\mathbf{G}_L$, and $\mathbf{G}_I$ is serial level-wise nonblocking and serial interface consistent with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, then

$$\begin{aligned}&(\forall s \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m)(\forall h \in \Sigma_{IH}^*.\Sigma_A)\\ &\quad sh \in \mathcal{H} \cap \mathcal{I} \Rightarrow (\exists u \in \Sigma^*)(P_{IH}(u) = h)\\ &\qquad \wedge (su \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m).\end{aligned}$$

Proof: See the Appendix. ■

The last proposition of the three is the *general construction proposition*. This proposition is more general than **Proposition 4)** as it handles the case that string h does not contain answer events or does not end in an answer event. It makes use of **Proposition 4)** to handle the other cases.

Proposition 5: If the system composed of DES $\mathbf{G}_H, \mathbf{G}_L$, and $\mathbf{G}_I$ is serial level-wise nonblocking and serial interface consistent with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, then

$$\begin{aligned}&(\forall s \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m)(\forall h \in \Sigma_{IH}^*)\\ &\quad sh \in \mathcal{H}_m \cap \mathcal{I}_m \Rightarrow (\exists u \in \Sigma^*)(su \in \mathcal{H}_m \cap \mathcal{L}_m \cap \mathcal{I}_m)\\ &\qquad \wedge (P_{IH}(u) = h) \wedge (P_I(u) \in \{\epsilon\} \cup \Sigma_R.\Sigma_I^*).\end{aligned}$$

Proof: See the Appendix. ■

We now present our main result for this section. In essence, the theorem says that if the high-level and low-level are individually nonblocking, and the system is serial interface consistent, then the nonblocking property will be preserved by the synchronous product operation.

Theorem 1: If the system composed of DES $\mathbf{G}_H, \mathbf{G}_L$, and $\mathbf{G}_I$ is serial level-wise nonblocking and serial interface consistent with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, then

$$L(\mathbf{G}) = \bar{L}_m(\mathbf{G}), \quad \text{where } \mathbf{G} = \mathbf{G}_H \,\|\, \mathbf{G}_L \,\|\, \mathbf{G}_I.$$

Proof: Assume system is serial level-wise nonblocking and serial interface consistent. (1)

As $\overline{L_m(\mathbf{G})} \subseteq L(\mathbf{G})$ is automatic, it suffices to show $L(\mathbf{G}) \subseteq \overline{L_m(\mathbf{G})}$.

Let $s \in L(\mathbf{G}) = \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}$ (2)

We will now show this implies $s \in \overline{L_m(\mathbf{G})}$

It is sufficient to show

$$(\exists u \in \Sigma^*) su \in L_m(\mathbf{G}) = \mathcal{H}_m \cap \mathcal{L}_m \cap \mathcal{I}_m.$$

As $s \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}$ [by (1)] and by **Proposition 1** we can conclude: $(\exists l \in \Sigma_{IL}^*) sl \in \mathcal{H} \cap \mathcal{L}_m \cap \mathcal{I}_m$ (3)

As $sl \in \mathcal{H} \cap \mathcal{I}$ [by (3)], we can apply **Point I** of the level-wise nonblocking definition, and conclude

$$(\exists h' \in \Sigma^*) slh' \in \mathcal{H}_m \cap \mathcal{I}_m. \quad (4)$$

We next note that: (5)

$$P_{IH}(slh') = P_{IH}(sl)P_{IH}(h') = P_{IH}(sl)P_{IH}(P_{IH}(h')) \\ = P_{IH}(slP_{IH}(h')).$$

As $\mathcal{H}_m := P_{IH}^{-1}(L_m(\mathbf{G}_H))$, (4) and (5) implies that: $P_{IH}(slP_{IH}(h')) \in L_m(\mathbf{G}_H)$ and thus $slP_{IH}(h') \in \mathcal{H}_m$. (6)

As $\Sigma_I \subseteq \Sigma_{IH}$, we can use (5) to conclude

$$P_I(slh') = P_I(slP_{IH}(h')).$$

Noting that $\mathcal{I}_m := P_I^{-1}(L_m(\mathbf{G}_I))$, we can use the same logic as (6) and conclude that: $slP_{IH}(h') \in \mathcal{I}_m$.

Combining with (6), we have: $slP_{IH}(h') \in \mathcal{H}_m \cap \mathcal{I}_m$

Thus, take $h = P_{IH}(h')$ and we have

$$h \in \Sigma_{IH}^* \quad \text{and} \quad slh \in \mathcal{H}_m \cap \mathcal{I}_m. \quad (7)$$

Since $sl \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m$ [by (3)] and by **Proposition 5** (take sl to be string s in proposition) we can conclude

$$(\exists u' \in \Sigma^*) slu' \in \mathcal{H}_m \cap \mathcal{L}_m \cap \mathcal{I}_m.$$

We then take $u = lu'$ and we have $su \in \mathcal{H}_m \cap \mathcal{L}_m \cap \mathcal{I}_m = L_m(\mathbf{G})$, as required. ■

C. Controllability Propositions and Theorem

We will now present two supporting propositions, followed by our main controllability result. Our first proposition asserts that if the system is serial level-wise controllable, then $\mathbf{G}_I$ and $\mathbf{S}_L$ are together controllable for the system's flat plant.

Proposition 6: If the system composed of plant components $\mathbf{G}_H^p, \mathbf{G}_L^p$, supervisors $\mathbf{S}_H, \mathbf{S}_L$, and interface $\mathbf{G}_I$, is serial level-wise controllable with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, then

$$(\forall s \in L(\mathbf{Plant}) \cap \mathcal{S}_L \cap \mathcal{I}) \\ \mathrm{Elig}_{L(\mathbf{Plant})}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\mathcal{S}_L \cap \mathcal{I}}(s).$$

Proof: See proof in [7]. ■

The last proposition asserts that if the system is serial level-wise controllable, then $\mathbf{S}_H$ is controllable for our flat plant when it is already under the control of the interface.

Proposition 7: If the system composed of plant components $\mathbf{G}_H^p, \mathbf{G}_L^p$, supervisors $\mathbf{S}_H, \mathbf{S}_L$, and interface $\mathbf{G}_I$, is serial level-wise controllable with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, then

$$(\forall s \in L(\mathbf{Plant}) \cap \mathcal{I} \cap \mathcal{S}_H) \\ \mathrm{Elig}_{L(\mathbf{Plant}) \cap \mathcal{I}}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\mathcal{S}_H}(s).$$

Proof: See the proof in [7]. ■

We now present our main result for this section. In essence, it asserts that if the system is serial level-wise controllable, then controllability can be checked for each level separately in order to determine that the system's flat supervisor is controllable for the system's flat plant.

Theorem 2: If the system composed of plant components $\mathbf{G}_H^p, \mathbf{G}_L^p$, supervisors $\mathbf{S}_H, \mathbf{S}_L$, and interface $\mathbf{G}_I$, is serial level-wise controllable with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, then

$$(\forall s \in L(\mathbf{Plant}) \cap L(\mathbf{Sup})) \\ \mathrm{Elig}_{L(\mathbf{Plant})}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{L(\mathbf{Sup})}(s).$$

Proof: Assume the system is serial level-wise controllable. (1)

Let $s \in L(\mathbf{Plant}) \cap L(\mathbf{Sup})$ and $\sigma \in \mathrm{Elig}_{L(\mathbf{Plant})}(s) \cap \Sigma_u$. (2)

From (2), we have $s \in L(\mathbf{Plant}) \cap L(\mathbf{Sup}) = \mathcal{H}^p \cap \mathcal{L}^p \cap \mathcal{S}_H \cap \mathcal{S}_L \cap \mathcal{I}$. (3)

We also have $s\sigma \in L(\mathbf{Plant}) = \mathcal{H}^p \cap \mathcal{L}^p$. (4)

It suffices to show that $s\sigma \in \mathcal{S}_H \cap \mathcal{S}_L \cap \mathcal{I} = L(\mathbf{Sup})$.

From (3), we have $s \in \mathcal{H}^p \cap \mathcal{L}^p \cap \mathcal{S}_L \cap \mathcal{I} = L(\mathbf{Plant}) \cap \mathcal{S}_L \cap \mathcal{I}$.

By **Proposition 6)**, we can conclude: $s\sigma \in \mathcal{S}_L \cap \mathcal{I}$. (5)

Using (4) and (5), we have $s\sigma \in \mathcal{H}^p \cap \mathcal{L}^p \cap \mathcal{I}$.

$\Rightarrow \sigma \in \mathrm{Elig}_{L(\mathbf{Plant}) \cap \mathcal{I}}(s) \cap \Sigma_u$.

As $s \in L(\mathbf{Plant}) \cap \mathcal{I} \cap \mathcal{S}_H$ [by (3)], **Proposition 7)** implies: $s\sigma \in \mathcal{S}_H$.

Combining with (5), we have $s\sigma \in \mathcal{S}_H \cap \mathcal{S}_L \cap \mathcal{I}$, as required. ■

The HISC approach achieves its computational advantage over the standard monolithic approach by transforming the problem from verifying properties for a single large system (nonblocking and controllability), to a series of local verifications (using the structure of the system) on two smaller systems that together represent the original system. If these conditions fail, we modify the two systems until the local conditions are satisfied. In essence, we test that the two smaller systems are nonblocking and controllable and then we provide conditions that guarantee that these properties are closed under the synchronous product.

The advantage of this can be seen immediately when we consider the case that each of the two subsystems has on the order of 10^6 states, the limit with our personal current computing resources of monolithic automata based algorithms.[2] This means that we can now handle a system on the order of 10^{12} states with the same computing resources.

[2]By automata-based algorithms, we mean algorithms that extensionally represent the states and transitions of the DES, as opposed to using symbolic methods as in [50] and [51].

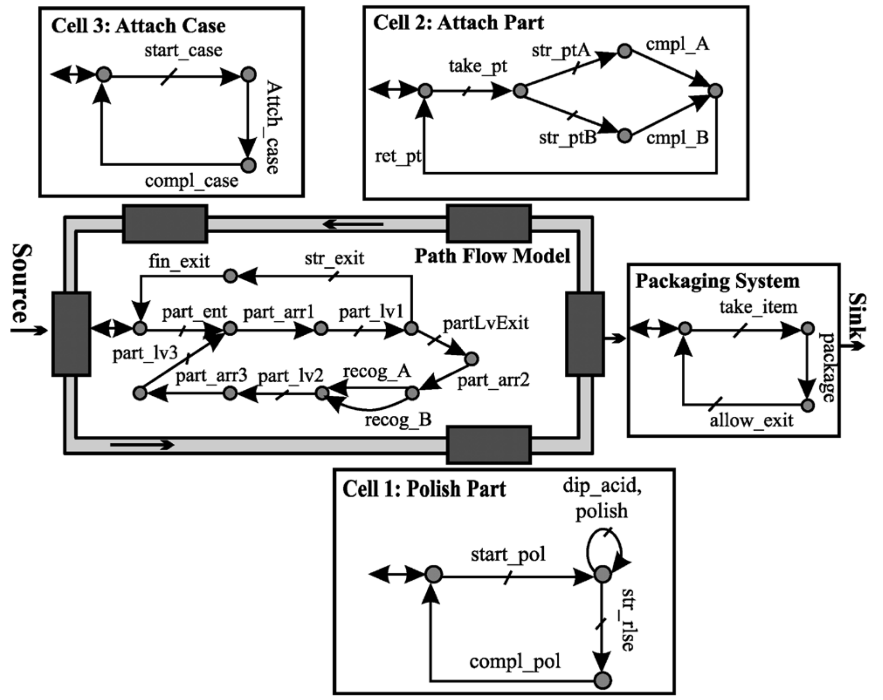


Fig. 6. Block diagram of plant.

Of course, this increased scalability comes with a price: A more restrictive architecture and, thus, the possible loss of global maximal permissiveness. We feel the tradeoff is worthwhile due to the increase in scalability and the other benefits of our approach. In particular, restricting the flow of information and localizing behavior into components has great benefits in terms of maintainability and reusability. This means that changes to one component won't affect the others, and that once the interface is defined each component can be designed and verified separately. For instance, a low-level can be designed and verified once and be used with any high-level that satisfies the interface conditions. This would allow vendors to create preverified libraries. As similar information-hiding approaches are standard practice in hardware and software design, we are confident that our approach will prove to be valuable.

In [6], we generalize the HISC setting to the *parallel case.* In the parallel case, we allow $n \geq 1$ low-levels. We will defer a formal complexity analysis on the HISC method until [6], as the serial case is the special case of $n = 1$.

V. Simple Manufacturing Example

We now present a simple manufacturing example to illustrate the method for the serial case. The example presented was inspired in part by the examples given in [35] and [57]. A larger example making use of the parallel extension of the theory will be presented in [6].

In the following sections, we will describe our problem setting, and then present the original plant components. We will then assign them to a particular level of our hierarchy, augmenting if necessary the low-level plant models so that they work better with an interface. We will then define the interface, supervisors, and finally present the complete system. We will conclude by demonstrating that the flat system is nonblocking and that the flat supervisor is controllable for the flat plant.

A. Description of Manufacturing Unit

As shown in Fig. 6, the manufacturing unit is composed of three cells connected by a conveyor belt. In front of each cell, is a part acquisition unit that automatically stops a part and holds it until it is given a release command. Parts enter the system at the far left and exit at the far right. After the item exits the conveyor system, it goes to a packaging machine.

The associated plant models can be seen in Fig. 7, namely **Attach Case to Assembly, Polish Part, Attach Part to Assembly, Packaging System**, and **Path Flow Model**. Table I defines abbreviations used for the event labels in the figures. For example, event *str_ptA*, which occurs in **Cell 2** of Fig. 6 and the **Attach Part to Assembly** subplant in Fig. 7, corresponds to the cell starting work on a type A part.

B. Defining Infrastructure

The first step in the process is to decide which plant models belong to the high-level subsystem, and which to the low-level subsystem. The division we have chosen can be seen in Fig. 7. We have added plant model **Define New Events** which introduces events *attch_ptA*, *attch_ptB*, *finA_attch*, and *finB_attch*. These events provide a more versatile means to interact with cell 2.

We define our interface to be the DES $\mathbf{G}_I$ shown in Fig. 7. We define the alphabet partition $\Sigma := \Sigma_H \dot{\cup} \Sigma_L \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A$ as follows:

$$\Sigma_R = \{\text{start\_pol},\ \text{attch\_ptA}, \text{attch\_ptB}, \text{start\_case}\}$$
$$\Sigma_A = \{\text{comp\_pol}, \text{finA\_attch}, \text{finB\_attch}, \text{compl\_case}\}$$
$$\begin{aligned}\Sigma_H = \{&\text{part\_ent}, \text{part\_arr1}, \text{part\_lv1}, \text{partLvExit}\\ &\text{str\_exit}, \text{fin\_exit}, \text{part\_arr2}, \text{recog\_A}, \text{recog\_B}\\ &\text{part\_lv2}, \text{part\_arr3}, \text{part\_lv3}, \text{take\_item}\\ &\text{allow\_exit}, \text{package}\}\end{aligned}$$
$$\begin{aligned}\Sigma_L = \{&\text{take\_pt}, \text{str\_ptA}, \text{str\_ptB}, \text{compl\_A}, \text{compl\_B}\\ &\text{ret\_pt}, \text{dip\_acid}, \text{polish}, \text{str\_rlse}, \text{attch\_case}\}.\end{aligned}$$

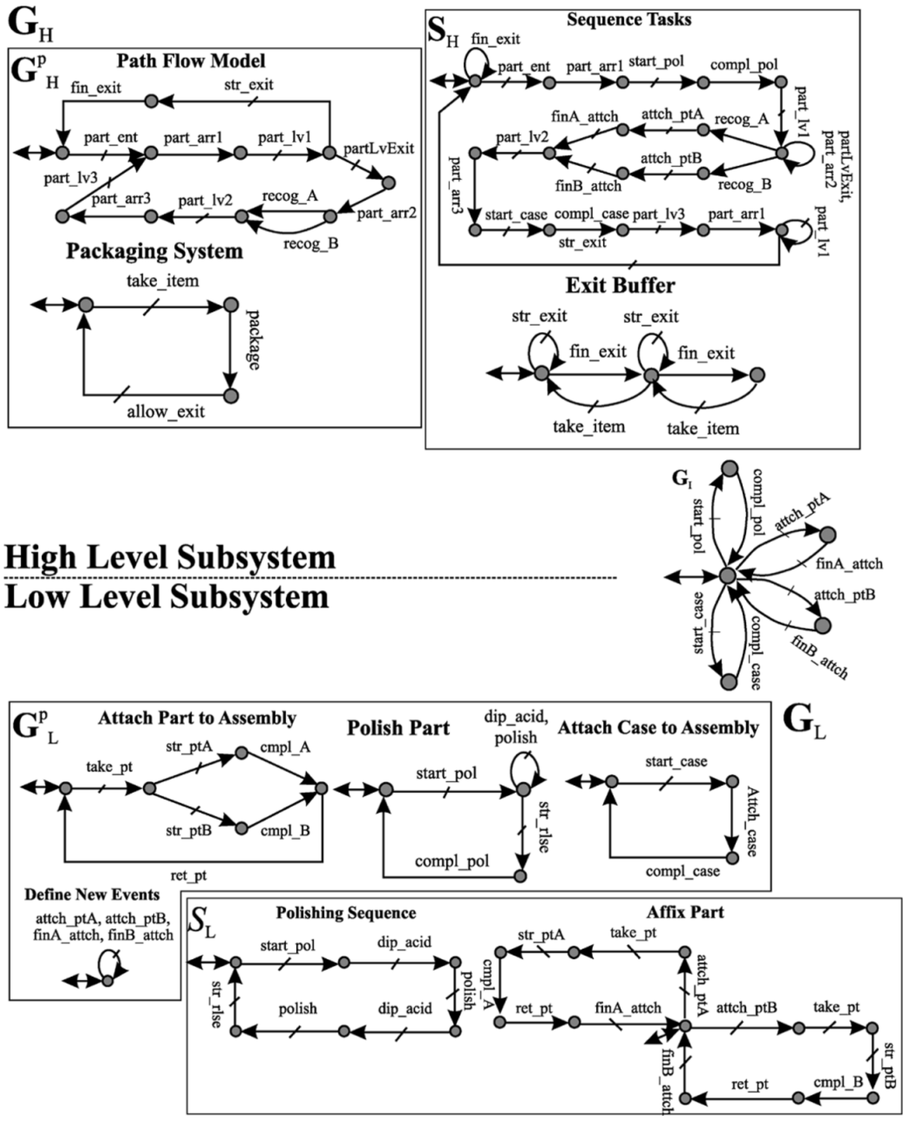


Fig. 7. Complete system definition.

C. Designing Supervisors

Now, that we have defined our interface, we design the low-level supervisors that will provide the functionality for the request events, and give meaning to the answer events. The idea is for the low-level to offer well-defined "services" to the high-level.

For cell one, we want the sequence *dip_acid-polish* to be repeated twice, after a *start_pol* event occurs. The supervisor is shown in Fig. 7, and is labeled **Polishing Sequence**. For cell two, we have to provide supervisors so that the cell reacts appropriately when events *attch_ptA* and *attch_ptB* occur. We also must guarantee that answer events *finA_attch* and *finB_attch* only occur when they have the appropriate meaning. The DES **Affix Part** in Fig. 7 shows how this is done.

We now design high-level supervisors that use the interface. Fig. 7 shows a supervisor (**Sequence Tasks**) that allows a part to visit each cell, executes the appropriate command for the cell and the part type, and then allows the part to leave the conveyor system. The figure also shows a supervisor (**Exit Buffer**) that implements a two item buffer for the packaging system.

In this paper, all supervisors are designed for their level as modular supervisors. The supervisors are designed by hand to meet the given specifications, and then verified that they are locally controllable and that they do not cause the local plant to block (i.e., they satisfy their portion of the serial level-wise nonblocking and serial level-wise controllable definitions). If they are not, they are modified until they are controllable and nonblocking. If a subsystem fails to satisfy its share of the interface properties, then it is modified until it does satisfy them.

D. Final System

We are now ready to define our system components. Fig. 7 shows our high-level subsystem, plant, and supervisor, DES $\mathbf{G}_H$, $\mathbf{G}_H^p$, and $\mathbf{S}_H$. We also have our low-level subsystem, plant, and supervisor, DES $\mathbf{G}_L$, $\mathbf{G}_L^p$, and $\mathbf{S}_L$. They are defined to be the synchronous product of the indicated automata.

Examining our system, we found it to be serial interface consistent, serial level-wise nonblocking, and serial level-wise controllable. We can thus conclude by **Theorem 1** that the flat system is nonblocking, and by **Theorem 2** that the flat supervisor is controllable for the flat plant.

VI. Conclusion

In this paper, we have presented a method for DES design and verification that implements many of the concepts of information hiding, thus providing benefits such as independent development, high degree of changeability and comprehensibility, and an excellent means to manage complexity by hiding unnecessary detail behind interfaces.

Hierarchical interface-based supervisory control offers an effective method to model systems with a natural client-server architecture. The method offers an intuitive way to model and design the system. As each requirement can be verified using only one of the two subsystems, the entire plant model never needs to be constructed or traversed (in computer memory), offering potentially significant savings in computation.

It is clear from the definitions in Section IV, that once we have defined our interface and event partition, evaluating our high and low-level subsystems for compliance can be done independently of each other. This means we can evaluate one high (low) level subsystem and use it with any low (high) level subsystem that satisfies the low (high) level portion of our definitions for the given interface and event partition. This provides us with the infrastructure required for component reuse.

Appendix
Proofs of Selected Propositions

Proposition 4:

Proof: Assume system is serial level-wise nonblocking and serial interface consistent. (1)

Let $s \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m, h \in \Sigma_{IH}^*.\Sigma_A$, and $sh \in \mathcal{H} \cap \mathcal{I}$ (2)

Let n be the number of answer events in string h. This implies

$$(\exists h_1, h_2, \ldots, h_n \in (\Sigma_H \cup \Sigma_R)^*.\Sigma_A) h_1 h_2 \ldots h_n = h. \quad (3)$$

From (2), we have: $sh_1h_2\ldots h_n \in \mathcal{H} \cap \mathcal{I}$ (4)

Using an inductive proof, we will now show

$$\begin{aligned}&(\exists u_0, u_1, \ldots, u_n \in \Sigma^*)\\ &\quad (su_0u_1\ldots u_n \in \mathcal{H}\cap\mathcal{L}\cap\mathcal{I}_m)\\ &\quad \wedge (P_{IH}(u_0u_1\ldots u_n) = h_0h_1\ldots h_n), \quad \text{where } h_0 := \epsilon.\end{aligned}$$

Claim to be Proven: For $k \in \{0, 1, \ldots, n\}$, there exists $u_0, u_1, \ldots, u_k \in \Sigma^*$ such that the following are true: (5)

a) $P_{IH}(u_0u_1\ldots u_k) = h_0h_1\ldots h_k$;

b) $su_0u_1\ldots u_k \in \mathcal{H}\cap\mathcal{L}\cap\mathcal{I}_m$.

We will first prove the initial case ($k = 0$), and then the general case of $k \in \{1, \ldots, n\}$. We can then conclude by induction that the claim has been proven.

Initial Case: $k = 0$

We take $u_0 = \epsilon$ and we have $P_{IH}(u_0) = h_0$.

We have $s = su_0 \in \mathcal{H}\cap\mathcal{L}\cap\mathcal{I}_m$ as $su_0 = s$ and $s \in \mathcal{H}\cap\mathcal{L}\cap\mathcal{I}_m$ by (2). **Initial case** complete.

Inductive Step: Let $k \in \{1, \ldots, n\}$. Assume $\exists u_0, u_1, \ldots, u_{k-1} \in \Sigma^*$ and that they satisfy (5) for $k-1$. (6)

We note: $sh_0h_1\ldots h_k \in \mathcal{H}\cap\mathcal{I}$, by (4). (7)

We now note

$$\begin{aligned}&P_{IH}(su_0u_1\ldots u_{k-1}h_k)\\ &\quad = P_{IH}(s)P_{IH}(u_0u_1\ldots u_{k-1})P_{IH}(h_k)\\ &\quad = P_{IH}(s)h_0h_1\ldots h_{k-1}P_{IH}(h_k) \text{ by (6).}\\ &\quad = P_{IH}(sh_0h_1\ldots h_k).\end{aligned} \quad (8)$$

As $\mathcal{H} := P_{IH}^{-1}(L(\mathbf{G}_H))$, (7) and (8) imply that $P_{IH}(su_0u_1\ldots u_{k-1}h_k) \in L(\mathbf{G}_H)$ and, thus

$$su_0u_1\ldots u_{k-1}h_k \in \mathcal{H}. \quad (9)$$

Since $\Sigma_I \subseteq \Sigma_{IH}$, we can conclude by (8) that $P_I(su_0u_1\ldots u_{k-1}h_k) = P_I(sh_0h_1\ldots h_k)$. As $\mathcal{I} := P_I^{-1}(L(\mathbf{G}_I))$, (7) implies that $P_I(su_0u_1\ldots u_{k-1}h_k) \in L(\mathbf{G}_I)$ and with (9)

$$su_0u_1\ldots u_{k-1}h_k \in \mathcal{H}\cap\mathcal{I}. \quad (10)$$

We note that

$$P_I(su_0u_1\ldots u_{k-1}) \in L_m(\mathbf{G}_I) \text{ by (6).} \quad (11)$$

By (10), we have $P_I(su_0u_1\ldots u_{k-1}h_k) \in L(\mathbf{G}_I)$. (12)

We note $h_k \in (\Sigma_H \cup \Sigma_R)^*.\Sigma_A$ [by (3)] implies that $P_I(h_k) \in \Sigma_R^*.\Sigma_A$

We have $P_I(su_0u_1\ldots u_{k-1}) \in (\Sigma_R.\Sigma_A)^* \cap L(\mathbf{G}_I)$ by (11) and **point B)** of the command-pair interface definition. (13)

Point A) of the command-pair interface definition implies that only a request event can follow $P_I(su_0u_1\ldots u_{k-1})$. (14)

From (12), we have: $P_I(su_0u_1\ldots u_{k-1}h_k) = P_I(su_0u_1\ldots u_{k-1})P_I(h_k) \in L(\mathbf{G}_I)$

$\Rightarrow P_I(h_k) \in \Sigma_R.\Sigma_R^*.\Sigma_A$ by (3) and (14).

By **point A)** of the command-pair interface definition, we have: $P_I(h_k) \in \Sigma_R.\Sigma_A$

$\Rightarrow h_k \in \Sigma_H^*.\Sigma_R.\Sigma_H^*.\Sigma_A$ by (3).

By **Proposition 3)** (take $su_0u_1\ldots u_{k-1}$ to be string s, and h_k to be string h), we conclude

$$\begin{aligned}&(\exists u' \in \Sigma^*) su_0u_1\ldots u_{k-1}u' \in \mathcal{H}\cap\mathcal{L}\cap\mathcal{I}_m)\\ &\quad \wedge (P_{IH}(u') = h_k)\\ &\qquad \Rightarrow P_{IH}(u_0u_1\ldots u_{k-1}u') = h_0h_1\ldots h_k \text{ by (6).}\end{aligned}$$

We now take $u_k = u'$ and the **inductive step** is complete.

We have now proven the **initial case** and the **inductive step**. We now conclude that the **claim** is true, by induction.

Thus, we take $k = n, u = u_0u_1\ldots u_n$ and we have $u \in \Sigma^*, su \in \mathcal{H}\cap\mathcal{L}\cap\mathcal{I}_m$, and $P_{IH}(u) = h$, as required. ∎

Proposition 5:

Proof: Assume system is serial level-wise nonblocking and serial interface consistent. (1)

Let $s \in \mathcal{H}\cap\mathcal{L}\cap\mathcal{I}_m, h \in \Sigma_{IH}^*$, and $sh \in \mathcal{H}_m \cap \mathcal{I}_m$ (2)

We have two cases to examine: **I)** $h \in \Sigma_H^*$ and **II)** $h \notin \Sigma_H^*$.

Case I) $h \in \Sigma_H^*$

Since $h \in \Sigma_H^*$, we have $P_{IL}(s) = P_{IL}(sh)$. As $\mathcal{L} := P_{IL}^{-1}(L(\mathbf{G}_L)), s \in \mathcal{L}$ [from (2)] implies that $P_{IL}(sh) \in L(\mathbf{G}_L)$ and thus by (2): $sh \in \mathcal{L}\cap\mathcal{H}_m\cap\mathcal{I}_m$.

By **Proposition 2)**, we have: $(\exists l \in \Sigma_L^*) shl \in \mathcal{H}_m\cap\mathcal{L}_m\cap\mathcal{I}_m$

We take $u = hl$ and **Case I)** is complete.

TABLE I
ABBREVIATIONS USED IN EVENT LABELS

Abbrev.	Meaning	Abbrev.	Meaning	Abbrev.	Meaning
pt	part (item)	str	start	arr	arrive
cmpl	complete	attch	attach	pol	polish
fin	finish	ent	enter	recog	recognize
rlse	release	lv	leave		

Case II) $h \notin \Sigma_H^*$

This implies $P_I(h) \neq \epsilon$. (3)

From (2), we have $s \in \mathcal{I}_m$ and thus: $P_I(s) \in L_m(\mathbf{G}_I)$.

$\Rightarrow P_I(s) \in (\Sigma_R.\Sigma_A)^* \cap L(\mathbf{G}_I)$ by **Point B** of the command-pair interface definition. (4)

Point A of the definition implies that only a request event can follow $P_I(s)$. (5)

From (2), we have $sh \in \mathcal{I}_m$ and thus: $P_I(s)P_I(h) \in L_m(\mathbf{G}_I)$

As $P_I(h) \neq \epsilon$ [by (3)], we can conclude by (5) that: $P_I(h) \in \Sigma_R.\Sigma_I^*$ (6)

By **Point B** of the command-pair interface definition we have: $P_I(s)P_I(h) \in \Sigma_I^*.\Sigma_A$

$\Rightarrow P_I(h) \in \Sigma_R.\Sigma_I^*.\Sigma_A$ by (6). (7)

$\Rightarrow h \in P_I^{-1}(\Sigma_R.\Sigma_I^*.\Sigma_A)$

$\Rightarrow h \in \Sigma_H^*.\Sigma_R.\Sigma_{IH}^*.\Sigma_A.\Sigma_H^*$ as $h \in \Sigma_{IH}^*$ (8)

$\Rightarrow (\exists h' \in \Sigma_{IH}^*.\Sigma_A)(h'' \in \Sigma_H^*)h'h'' = h.$ (9)

We can now conclude $sh' \in \mathcal{H} \cap \mathcal{I}$ as $sh \in \mathcal{H}_m \cap \mathcal{I}_m$, by (2).

By **Proposition 4)** (take h' to be string h), we can conclude

$$(\exists u' \in \Sigma^*)(P_{IH}(u') = h') \wedge (su' \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}). \qquad (10)$$

By (8) and (9), we have: $P_I(u') \in \Sigma_R.\Sigma_I^*.$ (11)

Since $h'' \in \Sigma_H^*$ [by (9)], we have $P_{IL}(su'h'') = P_{IL}(su')$. As $\mathcal{L} := P_{IL}^{-1}(L(\mathbf{G}_L)), su' \in \mathcal{L}$ [from (10)] implies that $P_{IL}(su'h'') \in L(\mathbf{G}_L)$ and, thus

$$su'h'' \in \mathcal{L}. \qquad (12)$$

From (2) and (9), we have: $sh'h'' \in \mathcal{H}_m \cap \mathcal{I}_m$ (13)

Since $P_{IH}(u') = h'$ [by (10)], we have: $P_{IH}(sh'h'') = P_{IH}(su'h'')$.

As $\mathcal{H}_m := P_{IH}^{-1}(L_m(\mathbf{G}_H))$, (13) implies that $P_{IH}(su'h'') \in L_m(\mathbf{G}_H)$ and, thus: $su'h'' \in \mathcal{H}_m$ (14)

As $\Sigma_I \subseteq \Sigma_{IH}$, we have: $P_I(sh'h'') = P_I(su'h'')$. As $\mathcal{I}_m := P_I^{-1}(L_m(\mathbf{G}_I))$, (13) implies that $P_I(su'h'') \in L_m(\mathbf{G}_I)$ and thus: $su'h'' \in \mathcal{I}_m$

Combining with (12) and (14), we can conclude

$$su'h'' \in \mathcal{L} \cap \mathcal{H}_m \cap \mathcal{I}_m.$$

By **Proposition 2)**, (take $su'h''$ to be string s), we can conclude: $(\exists l \in \Sigma_L^*)su'h''l \in \mathcal{H}_m \cap \mathcal{L}_m \cap \mathcal{I}_m$ (15)

We thus take $u = u'h''l$ and we have $P_{IH}(u) = h$ and $su \in \mathcal{H}_m \cap \mathcal{L}_m \cap \mathcal{I}_m$, and $P_I(u) \in \Sigma_R.\Sigma_I^*$ by (9), (11), and (15). **Case II)** is complete.

By **Cases I)** and **II)**, we now have constructed a string u with the required properties. ■

REFERENCES

[1] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, pp. 1053–1058, Dec. 1972.

[2] D. L. Parnas, P. C. Clements, and D. M. Weiss, "The modular structure of complex systems," *IEEE Trans. Software Eng.*, vol. SE-11, no. 3, pp. 259–266, Mar. 1985.

[3] R. Leduc, B. Brandin, and W. M. Wonham, "Hierarchical interface-based nonblocking verification," in *Proc. Canadian Conf. Electrical and Computer Engineering*, May 2000, pp. 1–6.

[4] E. W. Endsley, M. R. Lucas, and D. M. Tilbury. (2005) Modular design and verification of logic control for reconfigurable machining systems. [Online]http://www-personal.engin.umich.edu/~tilbury/papers.html

[5] R. Leduc, B. Brandin, W. M. Wonham, and M. Lawford, "Hierarchical interface-based supervisory control: Serial case," in *Proc. 40th Conf. Decision and Control*, Orlando, FL, Dec. 2001, pp. 4116–4121.

[6] R. Leduc, M. Lawford, and W. M. Wonham, "Hierarchical interface-based supervisory control, Part II: Parallel case," *IEEE Trans. Autom. Control*, vol. 50, no. 9, pp. 1336–1348, Sep. 2005.

[7] R. Leduc, "Hierarchical interface-based supervisory control," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 2002.

[8] L. de Alfaro and T. A. Henzinger, "Interface automata," in *Proc. 9th Annu. Symp. Foundations of Software Engineering*, 2001, pp. 109–120.

[9] M. Fabian, "On object-oriented non-deterministic supervisory control," Ph.D. dissertation, Chalmers Univ. Tech., Goteborg, Sweden, 1995.

[10] M. Fabian and B. Lennartson, "Petri nets and control synthesis: An object oriented approach," in *Proc. I.M.S.*, Vienna, Austria, Jun. 1994, pp. 365–370.

[11] M. Shayman and R. Kumar, "Process objects/masked composition: An object-oriented approach for modeling and control of discrete-event systems," *IEEE Trans. Autom. Control*, vol. 44, no. 10, pp. 1864–1869, Oct. 1999.

[12] W. M. Wonham. (2004, Jul.) Supervisory control of discrete-event systems. Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada. [Online]. Available: http://www.control.toronto.edu/DES/

[13] M. Courvoisier, M. Combacau, and A. de Bonneval, "Control and monitoring of large discrete event systems: A generic approach," in *Proc. ISIE 93*, Budapest, Hungary, 1993, pp. 571–576.

[14] M. Queiroz and J. Cury, "Modular supervisory control of large scale discrete event systems," in *Proc. WODES 2000*, Ghent, Belgium, Aug. 2000, pp. 103–110.

[15] G. Stremersch and R. Boel, "Decomposition of the supervisory control problem for Petri nets under preservation of maximal permissiveness," *IEEE Trans. Autom. Control*, vol. 46, no. 9, pp. 1490–1496, Sep. 2001.

[16] N. Alsop, "Formal techniques for the procedural control of industrial processes," Ph.D. dissertation, Dept. Chem. Eng. Chem. Technol., Imperial College of Science, Technology, and Medicine, London, U.K., 1996.

[17] R. Leduc, "PLC implementation of a DES supervisor for a manufacturing testbed: An implementation perspective," M.A.Sc. thesis, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 1996.

[18] G. Barrett and S. Lafortune, "Decentralized supervisory control with communicating controllers," *IEEE Trans. Autom. Control*, vol. 45, no. 9, pp. 1620–1638, Sep. 2000.

[19] F. Lin and W. M. Wonham, "Decentralized control and coordination of discrete-event systems with partial observations," in *Proc. 27th IEEE Conf. Decision Control*, Dec. 1988, pp. 1125–1130.

[20] K. Rudie and J. C. Willems, "The computational complexity of decentralized discrete-event control problems," *IEEE Trans. Autom. Control*, vol. 440, no. 7, pp. 1313–1319, Jul. 1995.

[21] K. Rudie and W. M. Wonham, "Think globally, act locally: Decentralized supervisory control," *IEEE Trans. Autom. Control*, vol. 37, no. 11, pp. 1692–1708, Nov. 1992.

[22] K. Wong and J. van Schuppen, "Decentralized supervisory control of discrete event systems with communication," in *Proc. WODES 1996*, Edinburgh, U.K., Aug. 1996, pp. 284–289.

[23] T. Yoo and S. Lafortune, "A general architecture for decentralized supervisory control of discrete-event systems," in *Proc. WODES 2000*, Ghent, Belgium, Aug. 2000, pp. 111–118.

[24] Y. Willner and M. Heymann, "Supervisory control of concurrent discrete-event systems," *Int. J. Control*, vol. 54, no. 5, pp. 1143–1169, 1991.

[25] S. Abdelwahed, "Interacting discrete event systems: Modeling, verification, and supervisory control," Ph.D. dissertation, Dept. of Elec. and Comp. Eng., Univ. Toronto, Toronto, ON, Canada, 2002.

[26] K. Åkesson, H. Flordal, and M. Fabian, "Exploiting modularity for synthesis and verification of supervisors," in *Proc. IFAC World Congr. Automatic Control*, Barcelona, Spain, 2002.

[27] S. Chen, "Control of discrete-event systems of vector and mixed structural type," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 1996.
[28] Y. Li, "Control of vector discrete-event systems," Ph.D. dissertation, Dept. Elect. Eng., Univ. Toronto, Toronto, ON, Canada, 1991.
[29] J. O. Moody and P. J. Antsaklis, *Supervisory Control of Discrete Event Systems Using Petri Nets*. Norwell, MA: Kluwer, 1998.
[30] M. Zhou, D. Wang, and I. Mayk, "Using Petri nets for object-oriented design of command and control systems," *Int. J. Intell. Control Syst.*, vol. 2, no. 2, pp. 287–300, 1998.
[31] M. Zhou and F. DiCesare, *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Norwell, MA: Kluwer, 1993.
[32] M. Uzam, "An optimal deadlock prevention policy for flexible manufacturing systems using Petri net models with resources and the theory of regions," *Int. J. Adv. Manuf. Technol.*, vol. 19, pp. 192–208, 2002.
[33] Y. Brave and M. Heymann, "Control of discrete event systems modeled as hierarchical state machines," *IEEE Trans. Autom. Control*, vol. 38, no. 12, pp. 1803–1819, Dec. 1993.
[34] P. Gohari-Moghadam, "A linguistic framework for controlled hierarchical DES," M.A.Sc. thesis , Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 1998.
[35] B. Wang, "Top-down design for RW supervisory control theory," M.A.Sc. thesis, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 1995.
[36] C. Ma and W. M. Wonham, "Control of state tree structures," in *Proc. 11th Mediterranean Conf. Control and Automation*, Jun. 2003. paper T4-005.
[37] R. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.
[38] A. Aziz, V. Singhal, and G. Swamy, "Minimizing interacting finite state machines: A compositional approach to language containment," in *Proc. IEEE Int. Conf. Computer Design: VLSI in Computers and Processors*, Cambridge, MA, Oct. 1994, pp. 255–261.
[39] P. Caines and Y. Wei, "The hierarchical lattices of a finite machine," *Syst. Control Lett.*, vol. 25, pp. 257–263, Jul. 1995.
[40] H. Chen and H.-M. Hanisch, "Model aggregation for hierarchical control synthesis of discrete event systems," in *Proc. 39th Conf. Decision Control*, Sydney, NSW, Australia, Dec. 2000, pp. 418–423.
[41] Y.-L. Chen and F. Lin, "Hierarchical modeling and abstraction of discrete event systems using finite state machines with parameters," in *Proc. 40th Conf. Decision Control*, Orlando, FL, Dec. 2001, pp. 4110–4115.
[42] J. M. Eyzell and J. E. Cury, "Exploiting symmetry in the synthesis of supervisors for discrete event systems," in *Proc. Amer. Control Conf.*, Philadelphia, PA, Jun. 1998, pp. 244–248.
[43] K. Q. Pu, "Modeling and control of discrete-event systems with hierarchical abstraction," MA.Sc. thesis, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 2000.
[44] R. G. Qiu and S. B. Joshi, "A structured adaptive supervisory control methodology for modeling the control of a discrete event manufacturing system," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 29, no. 6, pp. 573–586, Dec. 1999.
[45] G. Shen and P. E. Caines, "Hierarchically accelerated dynamic programming for finite-state machines," *IEEE Trans. Autom. Control*, vol. 47, no. 2, pp. 271–283, Feb. 2002.
[46] H. Zhong and W. M. Wonham, "On the consistency of hierarchical supervision in discrete-event systems," *IEEE Trans. Autom. Control*, vol. 35, no. 10, pp. 1125–1134, Oct. 1990.
[47] K. Wong, "Discrete-event control architecture: An algebraic approach," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 1994.
[48] T. Moor, J. Raisch, and J. Davoren, "Admissibility criteria for a hierarchical design of hybrid control systems," in *Proc. IFAC Conf. Analysis and Design of Hybrid Systems*, Saint-Malo, France, June 2003, pp. 389–394.
[49] J. Gunnarsson, "Symbolic methods and tools for discrete event dynamic systems," Ph.D. dissertation, Dept. Elect. Eng., Linköping Univ., Linköping, Sweden, 1997.
[50] Z. Zhang, "Smart TCT: An efficient algorithm for supervisory control design," M.A.Sc. thesis, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 2001.
[51] Z. Zhang and W. M. Wonham, "STCT: An efficient algorithm for supervisory control design," in *Proc. SCODES 2001*. Paris, France, Jul. 2001, pp. 82–93.
[52] J. Burch, E. M. Clarke, and K. McMillan, "Symbolic model checking: 10^{20} states and beyond," *Inform. Comput.*, vol. 98, pp. 142–170, 1992.
[53] K. McMillan, *Symbolic Model Checking*. Norwell, MA: Kluwer, 1992.
[54] S. Berezin, S. Campos, and E. M. Clarke, "Compositional reasoning in model checking," in *COMPOS'97*, ser. LNCS. New York: Springer-Verlag, 1998, vol. 1536, pp. 81–102.
[55] P. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete-event processes," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, 1987.
[56] W. M. Wonham and P. Ramadge, "On the supremal controllable sublanguage of a given language," *SIAM J. Control Optim.*, vol. 25, no. 3, pp. 637–659, 1987.
[57] B. Brandin and F. Charbonnier, "The supervisory control of the automated manufacturing system of the AIP," in *Proc. Rensselaer's 1994 4th Int. Conf. Computer Integrated Manufacturing and Automation Technology*, Troy, NY, Oct. 1994, pp. 319–324.

Ryan Leduc (M'02) received the B.Eng. degree in electrical engineering from the University of Victoria, Victoria, BC, Canada, in 1993, and the M.A.Sc. and Ph.D. degrees in electrical engineering from the University of Toronto, Toronto, ON, Canada, in 1996 and 2002, respectively.

In 1997 and 1998, he was a Guest Scientist at Siemens Corporate Technology, Munich, Germany. In 2001, he joined McMaster University, Hamilton, ON, Canada, where he is currently an Assistant Professor of Software Engineering. His research interests include supervisory control of discrete-event systems (DES) hierarchical structure, concurrency and implementation issues, and DES as software and hardware. He is also interested in hierarchical approaches to formal verification of software and hardware.

Dr. Leduc is the Chair of the IEEE Control Systems Society Technical Committee on Discrete-Event Systems.

Bertil A. Brandin (M'95) received the B.S. degree in mechanical engineering from the University of New South Wales, Australia, in 1984, and the M.A.Sc. and Ph.D. degrees in electrical engineering from the University of Toronto, Toronto, ON, Canada, in 1989 and 1993, respectively.

From 1993 to 1995, he was Project Leader in a collaborative project on the supervisory control of automated manufacturing systems between the Province of Ontario, Canada, and Région Rhône-Alpes, France. In 1994, he was an Invited Scientist at the Rockwell Science Center, Thousand Oaks, CA. From 1995 to 2003, he managed R&D on formal verification techniques for software and business process applications at Siemens Corporate Technology, Munich, Germany. He is currently General Manager of Siemens Medical Solutions Health Services, Italy. His research interests include discrete-event systems, supervisory control, formal verification techniques, model-based testing, and diagnostics.

Mark Lawford (S'88–M'97) received the B.Sc. degree in engineering mathematics from Queen's University, Kingston, ON, Canada, in 1989 (receiving the University Medal in engineering mathematics), and the M.A.Sc. and Ph.D. degrees in electrical engineering at the University of Toronto, Toronto, ON, Canada, in 1992 and 1997, respectively.

From 1997 to 1998, he was with Ontario Hydro as a consultant on the Darlington Nuclear Generating Station Shutdown Systems Redesign project, where he was a co-recipient of an Ontario Hydro New Technology Award. Currently, he is an Associate Professor in the Department of Computing and Software at McMaster University, Hamilton, ON, Canada, where he has helped develop the Software Engineering programs. His research interests include discrete-event systems, formal methods for real-time systems, and computer aided inspection of safety critical software. He is a licensed Professional Engineer in the province of Ontario.

W. M. Wonham (M'64–SM'76–F'77) received the B.Eng. degree in engineering physics from McGill University, Montreal, QC, Canada, in 1956, and the Ph.D. degree in control engineering from the University of Cambridge, Cambridge, U.K., in 1961.

From 1961 to 1969, he was associated with several U.S. research groups in control. Since 1970, he has been a Faculty Member in Systems Control with the Department of Electrical and Computer Engineering of the University of Toronto, Toronto, ON, Canada. In addition, he has held visiting lectureships at Washington University, St. Louis, MO, the Massachusetts Institute of Technology, Cambridge, the Institute of System Science of the Academia Sinica, Beijing, China, and other institutions. His research interests have included stochastic control and filtering, geometric multivariable control, and discrete-event systems. He is the author of *Linear Multivariable Control: A Geometric Approach* (New York: Springer-Verlag, 1985) and the coauthor (with C. Ma) of *Hierarchical Control of State Tree Structures* (New York: Springer-Verlag, 2005).

Dr. Wonham is a Fellow of the Royal Society of Canada, and (2005) a Foreign Associate of the (U.S.) National Academy of Engineering. In 1987, he received the IEEE Control Systems Science and Engineering Award, and, in 1990, was a Brouwer Medallist of the Netherlands Mathematical Society. In 1996, he was appointed University Professor in the University of Toronto, and in 2000, University Professor Emeritus.