

# Hierarchical Interface-based Supervisory Control - Part I: Serial Case

R.J. Leduc, *Member, IEEE* B.A. Brandin, *Member, IEEE*

M. Lawford, *Member, IEEE* and W.M. Wonham *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 (DES)), 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 re-use of the component subsystems. Finally, we examine a simple example to illustrate the method.

## I. INTRODUCTION

In the area of Discrete-Event Systems (DES), 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 work, our goal is to develop an architecture for DES that supports a scalable method for the design and verification of nonblocking supervisory controllers.

R.J. Leduc and M. Lawford are with the Dept. of Computing and Software, McMaster University, McMaster University, 1280 Main Street West, Hamilton, ON, Canada L8S 4K1. Email: leduc,lawford@mcmaster.ca

B.A. Brandin is with Siemens Corporate Technology, Munich, Germany. Email: bertil.brandin@siemens.com

W.M. Wonham is with the Dept. of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada, M5S 3G4. Email: wonham@control.utoronto.ca

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 behaviour 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.

Additionally, we note that all levels of the circuit's hierarchy operate concurrently: signals at a high level can be changing simultaneously with signals at lower levels of the hierarchy.

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], [2], [3], [4]). This approach is referred to as *information hiding*. Some of its advantages are given below.

- 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 behaviour 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 with complexity [5], [6]. These approaches develop interfaces between components to provide structure that guarantees global properties such as controllability [6] or nonblocking [5]. 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 [5], [7]. While in general the method can decompose the system into multiple “parallel” subsystems (see [8], [9]), for the purposes of this paper (Part I of II) we restrict 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 [6] is the results on nonblocking.

Recently Alfaro and Henzinger [10] have used interface automata similar to I/O automata [11] 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 [10], 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* [12], [13], [14] 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.* [15] 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* [16], [17], [18], [19]. 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. [20], [21]), verifying nonblocking of the closed loop system is still a problem.

Another approach is embodied by *Vector DES* (VDES) [16], [22], [23], [24] and *Petri Nets* (PN) [25], [26], [27]. 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 [28].

In *Decentralized control* [29], [30], [31], [32], [33], [34], 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.

A promising approach is the development of a multi-level hierarchy. In order to aid in classification, we make a distinction between structural multi-level hierarchies with explicit mechanisms (modeling constructs) to facilitate hierarchy (e.g. [35], [36], [37], [38], [39], [40]) as opposed to aggregate (bottom up) multi-level hierarchies which we will discuss later. In

structural multi-level hierarchies, plants and supervisors are modelled as multi-level structures similar to automata, except that certain states at a given level can be expanded into a more detailed lower level model. Although [39] 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, [36] 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.* [40] who, with the use of *binary decision diagrams* (BDDs) [41], 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 [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55]. 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 [55], [53], [50], 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 can’t 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.

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 [56]. Zhang *et al* [57], [58] 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 model checking and temporal logic [59], [60], [61], [62], [63] 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 [57], [58] 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 [64]. 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, in order to verify that  $P_1 \parallel P_2 \models \phi$  by compositional model checking,  $P_2$  might be replaced by an aggregate “interface process”  $A_2$  such that if  $P_1 \parallel A_2 \models \phi$  then  $P_1 \parallel P_2 \models \phi$ .

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. DISCRETE-EVENT SYSTEMS PRELIMINARIES

Ramadge-Wonham supervisory control [65], [66], [16] provides a theoretical framework for the control of discrete-event systems (DES), systems that are discrete in space and time. For a detailed exposition of DES, see [16]. Below, we present a summary of the terminology that we use in this paper.

Let  $\Sigma$  be a finite set of distinct symbols (*events*), and  $\Sigma^*$  be the set of all finite sequences of

events, including  $\epsilon$ , the *empty string*. Let  $L \subseteq \Sigma^*$  be a *language* over  $\Sigma$ . 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:

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

Let  $\text{Pwr}(\Sigma)$  denote the power set of  $\Sigma$ . For language  $L$ , the eligibility operator  $\text{Elig}_L : \Sigma^* \rightarrow \text{Pwr}(\Sigma)$  is defined as below. Let  $s \in \Sigma^*$ ; then

$$\text{Elig}_L(s) := \{\sigma \in \Sigma \mid s\sigma \in L\}$$

A DES automaton is represented as a 5-tuple:

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

where  $Y$  is the state set,  $\Sigma$  is the event set, the partial function  $\delta : Y \times \Sigma \rightarrow Y$  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  $G$ , 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  $\delta$  is extended to  $\delta : Y \times \Sigma^* \rightarrow Y$  in the natural way. The notation  $\delta(y, s)!$  means that  $\delta$  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:

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

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

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

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*  $P_i : \Sigma^* \rightarrow \Sigma_i^*$  according to:

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

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  $P_i^{-1} : \text{Pwr}(\Sigma_i^*) \rightarrow \text{Pwr}(\Sigma^*)$  is the inverse image function of  $P_i$  (see e.g. [16]).

The synchronous product of DES  $\mathbf{G}_1 = (Y_1, \Sigma_1, \delta_1, y_{o1}, Y_{m1})$  and  $\mathbf{G}_2 = (Y_2, \Sigma_2, \delta_2, y_{o2}, Y_{m2})$ , denoted  $\mathbf{G}_1 || \mathbf{G}_2$ , is defined to be a reachable DES  $\mathbf{G}$  with the properties:<sup>1</sup>

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

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 below:

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

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 behaviour of a plant  $\mathbf{G}_1 = (Y_1, \Sigma_1, \delta_1, y_{o1}, Y_{m1})$  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  $\Sigma$ , the natural projections  $P_1$  and  $P_S$ , and languages  $L_{G_1}$  and  $L_S$  as below:

$$\begin{aligned} \Sigma &:= \Sigma_1 \cup \Sigma_S & P_1 : \Sigma^* &\rightarrow \Sigma_1^* & P_S : \Sigma^* &\rightarrow \Sigma_S^* \\ L_{G_1} &:= P_1^{-1}L(\mathbf{G}_1) & L_S &:= P_S^{-1}L(\mathbf{S}) \end{aligned}$$

<sup>1</sup>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.

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

$$L_S \Sigma_u \cap L_{G_1} \subseteq L_S$$

or, equivalently:

$$(\forall s \in L_{G_1} \cap L_S) \text{Elig}_{L_{G_1}}(s) \cap \Sigma_u \subseteq \text{Elig}_{L_S}(s)$$

### 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. Figure 1 shows conceptually the structure of the system.

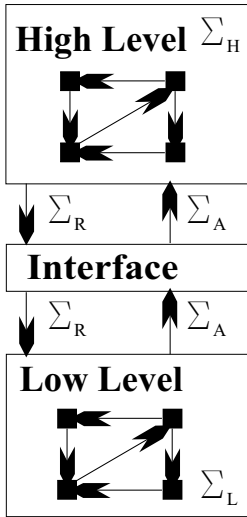


Fig. 1. Interface Block Diagram.

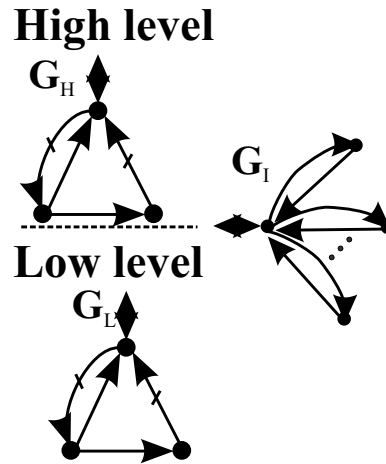


Fig. 2. Two Tiered Structure of Serial System.

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 ( $\Sigma$ ) into four pairwise disjoint alphabets:  $\Sigma_H$ ,  $\Sigma_L$ ,  $\Sigma_R$ , and  $\Sigma_A$ . The events in  $\Sigma_H$  are called *high level events* and the events in  $\Sigma_L$  *low level events* as these events appear only in the high level and low level models, respectively.

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

In the remainder of the 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 **Answer** :  $\Sigma_R \rightarrow \text{Pwr}(\Sigma_A)$ . For  $\rho \in \Sigma_R$ , **Answer**( $\rho$ ) is the set of possible answers (referred to as the *answer set*) the low level subsystem could provide after receiving request  $\rho$ . We also add the constraints that the low level subsystem must provide at least one response for each request it receives, and that  $\Sigma_A$  does not contain any unused events. Finally, we see in Figure 3 how a star interface, with  $n = |\Sigma_R|$ , is expressed as a DES. The required structure is given by DES  $G_I$ . We also require that the event set of  $G_I$  be set to  $\Sigma_R \dot{\cup} \Sigma_A$ .

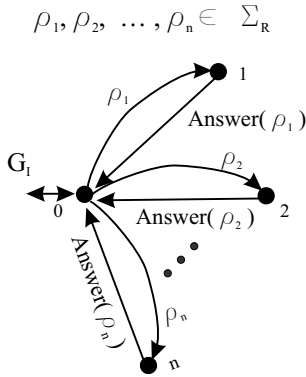


Fig. 3. Star Interface Specification.

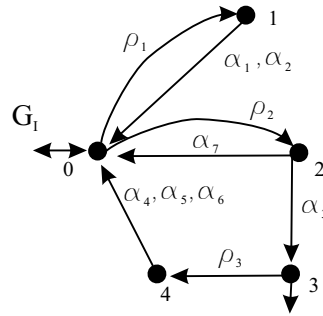


Fig. 4. Example Command-pair Interface.

We now define command-pair interfaces which were designed as a generalisation 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 Figure 3 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 Figure 4. Request events  $\rho_1$  and  $\rho_2$  might represent the regular behaviour of the system, while  $\alpha_3$  and  $\rho_3$  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 below:

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

- (A)  $L(G_I) \subseteq \overline{(\Sigma_R.\Sigma_A)^*}$
- (B)  $L_m(G_I) = (\Sigma_R.\Sigma_A)^* \cap L(G_I)$

Note  $G_I$ '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  $G_I$ , a request event always occurs first and then request and answer events alternate. Finally, **Point B** implies that the marked language of  $G_I$  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 modelled by DES  $G_H$  (defined over event set  $\Sigma_H \cup \Sigma_R \cup \Sigma_A$ ), the low level subsystem by DES  $G_L$  (defined over event set  $\Sigma_L \cup \Sigma_R \cup \Sigma_A$ ), and the interface by DES  $G_I$ . Also, the term *high level* will mean the DES  $G_H || G_I$ , and the term *low level* the DES  $G_L || G_I$ . The overall structure of the system is displayed in Figure 2.

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* as below. By flat system we refer to the equivalent DES that would represent our system if we ignored the interface structure.

$$G = G_H || G_L || G_I$$

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

$$\begin{aligned} \Sigma_I &:= \Sigma_R \dot{\cup} \Sigma_A \\ \Sigma_{IH} &:= \Sigma_H \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A \\ \Sigma_{IL} &:= \Sigma_L \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A \\ P_{IH} : \Sigma^* &\rightarrow \Sigma_{IH}^* \\ P_{IL} : \Sigma^* &\rightarrow \Sigma_{IL}^* \\ P_I : \Sigma^* &\rightarrow \Sigma_I^* \\ \mathcal{H} &:= P_{IH}^{-1}(L(G_H)), \quad \mathcal{H}_m := P_{IH}^{-1}(L_m(G_H)) \subseteq \Sigma^* \\ \mathcal{L} &:= P_{IL}^{-1}(L(G_L)), \quad \mathcal{L}_m := P_{IL}^{-1}(L_m(G_L)) \subseteq \Sigma^* \\ \mathcal{I} &:= P_I^{-1}(L(G_I)), \quad \mathcal{I}_m := P_I^{-1}(L_m(G_I)) \subseteq \Sigma^* \end{aligned}$$

Whereas the representation of the system as given in Figure 2 (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 Figure 5.

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

$$G_H := \mathcal{G}_H || \mathcal{S}_H \quad G_L := \mathcal{G}_L || \mathcal{S}_L$$

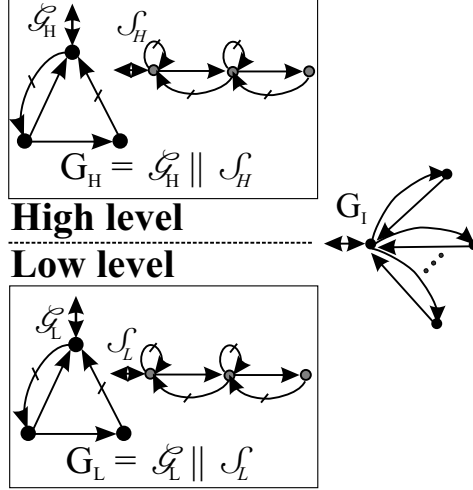


Fig. 5. Plant and Supervisor Subplant Decomposition

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

$$\begin{aligned}
 \mathbf{Plant} &:= \mathcal{G}_H || \mathcal{G}_L & \mathbf{Sup} &:= \mathcal{S}_H || \mathcal{S}_L || G_I \\
 \mathbf{H} &:= P_{IH}^{-1}L(\mathcal{G}_H), & \mathbf{H}_S &:= P_{IH}^{-1}L(\mathcal{S}_H), & \subseteq \Sigma^* \\
 \mathbf{L} &:= P_{IL}^{-1}L(\mathcal{G}_L), & \mathbf{L}_S &:= P_{IL}^{-1}L(\mathcal{S}_L), & \subseteq \Sigma^*
 \end{aligned}$$

#### 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  $G_H$ ,  $G_L$ , and  $G_I$ , is said to be *serial level-wise nonblocking* if the following conditions are satisfied:

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

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

We next present the *serial level-wise controllability* definition. To summarise the definition, **Point I** states that the plants, supervisors, and interface have the indicated event sets. **Point II** states that the interface and  $\mathcal{S}_L$  are together controllable for the low level plant  $\mathcal{G}_L$ . **Point III** states that supervisor  $\mathcal{S}_H$  is controllable for the high level plant  $\mathcal{G}_H$ , when it is already under the control of the interface. By treating the *interface* as a plant at the *high level*, we allow the high level supervisor,  $\mathcal{S}_H$ , to be more flexible as the *interface* may have more information about when interface events are eligible than the high level plant.

*Definition 5:* The system composed of plant components  $\mathcal{G}_H$ ,  $\mathcal{G}_L$ , supervisors  $\mathcal{S}_H$ ,  $\mathcal{S}_L$ , and interface  $G_I$ , 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  $\mathcal{G}_H$  and  $\mathcal{S}_H$  is  $\Sigma_{IH}$ , the alphabet of  $\mathcal{G}_L$  and  $\mathcal{S}_L$  is  $\Sigma_{IL}$ , and the alphabet of  $G_I$  is  $\Sigma_I$
- (II)  $(\forall s \in \mathbf{L} \cap \mathbf{L}_S \cap \mathcal{I}) \text{ Elig}_{\mathbf{L}}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathbf{L}_S \cap \mathcal{I}}(s)$
- (III)  $(\forall s \in \mathbf{H} \cap \mathcal{I} \cap \mathbf{H}_S) \text{ Elig}_{\mathbf{H} \cap \mathcal{I}}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathbf{H}_S}(s)$

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  $G_H$ ,  $G_L$  and  $G_I$ , 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:

#### Multi-level Properties

- 1) The event set of  $G_H$  is  $\Sigma_{IH}$ , and the event set of  $G_L$  is  $\Sigma_{IL}$ .
- 2)  $G_I$  is a command-pair interface.

#### High Level Properties

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

#### Low Level Properties

- 4)  $(\forall s \in \mathcal{L} \cap \mathcal{I}) \text{Elig}_{\mathcal{I}}(s) \cap \Sigma_R \subseteq \text{Elig}_{\mathcal{L}}(s)$
- 5)  $(\forall s \in \Sigma^* \cdot \Sigma_R \cap \mathcal{L} \cap \mathcal{I})$   
 $\text{Elig}_{\mathcal{L} \cap \mathcal{I}}(s \Sigma_L^*) \cap \Sigma_A = \text{Elig}_{\mathcal{I}}(s) \cap \Sigma_A$   
 where  $\text{Elig}_{\mathcal{L} \cap \mathcal{I}}(s \Sigma_L^*) := \cup_{l \in \Sigma_L^*} \text{Elig}_{\mathcal{L} \cap \mathcal{I}}(sl)$
- 6)  $(\forall s \in \mathcal{L} \cap \mathcal{I})$   
 $s \in \mathcal{I}_m \Rightarrow (\exists l \in \Sigma_L^*) sl \in \mathcal{L}_m \cap \mathcal{I}_m$

Property 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.

Property 2: This property states that DES  $G_I$  satisfies the definition of a command-pair interface.

Property 3: This property asserts that the high level subsystem ( $G_H$ ) 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.

Property 4: This property asserts that the low level subsystem ( $G_L$ ) 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.

Property 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  $\Sigma_L^*$  to each answer event (i.e. all  $\alpha \in \mathbf{Answer}(\rho)$ , assuming that we are dealing with a star interface) that  $G_I$  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.

Property 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  $G_I$  and  $G_L$ ).

## 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 re-used 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  $G_H$ ,  $G_L$ , and  $G_I$  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 Appendix. ■

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  $G_H$ ,  $G_L$ , and  $G_I$  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 proof in [9]. ■

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  $G_H$ ,  $G_L$ , and  $G_I$  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

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

*Proof:* See Appendix. ■

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  $G_H$ ,  $G_L$ , and  $G_I$  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

$$(\forall s \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m)(\forall h \in \Sigma_{IH}^* \cdot \Sigma_A) \\ sh \in \mathcal{H} \cap \mathcal{I} \Rightarrow (\exists u \in \Sigma^*) (P_{IH}(u) = h) \wedge (su \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m)$$

*Proof:* See 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  $h$  doesn't contain answer events or doesn't end in an answer event. It makes use of **Proposition 4** to handle the other cases.

*Proposition 5:* If the system composed of DES  $G_H$ ,  $G_L$ , and  $G_I$  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

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

*Proof:* See 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  $G_H$ ,  $G_L$ , and  $G_I$  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

$$L(G) = \overline{L_m(G)}, \text{ where } G = G_H || G_L || G_I$$

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

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

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

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

It is sufficient to show:  $(\exists u \in \Sigma^*) su \in L_m(G) = \mathcal{H}_m \cap \mathcal{L}_m \cap \mathcal{I}_m$

As  $s \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}$  (by (1)) and **Proposition 1** we can conclude:

$$(\exists l \in \Sigma_{IL}^*) sl \in \mathcal{H} \cap \mathcal{L}_m \cap \mathcal{I}_m \quad (3)$$

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)

$$\begin{aligned} P_{IH}(slh') &= P_{IH}(sl)P_{IH}(h') \\ &= P_{IH}(sl)P_{IH}(P_{IH}(h')) \\ &= P_{IH}(slP_{IH}(h')) \end{aligned}$$

As  $\mathcal{H}_m := P_{IH}^{-1}(L_m(G_H))$ , (4) and (5) implies that  $P_{IH}(slP_{IH}(h')) \in L_m(G_H)$  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  $\mathcal{I}_m := P_I^{-1}(L_m(G_I))$ , we can use the same logic as (6) and conclude that  $slP_{IH}(h') \in \mathcal{I}_m$ .

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

We thus take  $h = P_{IH}(h')$  and we have:

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

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:

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

We then take  $u = lu'$  and we have  $su \in \mathcal{H}_m \cap \mathcal{L}_m \cap \mathcal{I}_m = L_m(G)$ , 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  $G_I$  and  $\mathcal{S}_L$  are together controllable for the system's flat plant.

*Proposition 6:* If the system composed of plant components  $\mathcal{G}_H, \mathcal{G}_L$ , supervisors  $\mathcal{S}_H, \mathcal{S}_L$ , and interface  $G_I$ , 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:

$$(\forall s \in L(\mathbf{Plant}) \cap \mathbf{L}_S \cap \mathcal{I}) \quad \text{Elig}_{L(\mathbf{Plant})}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathbf{L}_S \cap \mathcal{I}}(s)$$

*Proof:* See Appendix. ■

The last proposition asserts that if the system is serial level-wise controllable, then  $\mathcal{S}_H$  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  $\mathcal{G}_H, \mathcal{G}_L$ , supervisors  $\mathcal{S}_H, \mathcal{S}_L$ , and interface  $G_I$ , 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:

$$(\forall s \in L(\mathbf{Plant}) \cap \mathcal{I} \cap \mathbf{H}_S) \quad \text{Elig}_{L(\mathbf{Plant}) \cap \mathcal{I}}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathbf{H}_S}(s)$$

*Proof:* See proof in [9]. ■

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  $\mathcal{G}_H, \mathcal{G}_L$ , supervisors  $\mathcal{S}_H, \mathcal{S}_L$ , and interface  $G_I$ , 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:

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

*Proof:*

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

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

From (2), we have  $s \in L(\mathbf{Plant}) \cap L(\mathbf{Sup}) = \mathbf{H} \cap \mathbf{L} \cap \mathbf{H}_S \cap \mathbf{L}_S \cap \mathcal{I}$  (3)

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

It suffices to show that  $s\sigma \in \mathbf{H}_S \cap \mathbf{L}_S \cap \mathcal{I} = L(\mathbf{Sup})$

From (3), we have  $s \in \mathbf{H} \cap \mathbf{L} \cap \mathbf{L}_S \cap \mathcal{I} = L(\mathbf{Plant}) \cap \mathbf{L}_S \cap \mathcal{I}$

By **Proposition 6** we can conclude:  $s\sigma \in \mathbf{L}_S \cap \mathcal{I}$  (5)

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

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

As  $s \in L(\mathbf{Plant}) \cap \mathcal{I} \cap \mathbf{H}_S$  (by (3)), **Proposition 7** implies:  $s\sigma \in \mathbf{H}_S$

Combining with (5), we have  $s\sigma \in \mathbf{H}_S \cap \mathbf{L}_S \cap \mathcal{I}$ , 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 (non-blocking 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.<sup>2</sup> This means that we can now handle a system on the order of  $10^{12}$  states with the same computing resources.

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 trade off 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.

<sup>2</sup>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 [57], [58].

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 the Part II companion paper [8], 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 Part II [8], 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 Wang [39], and in Brandin [67]. A larger example making use of the parallel extension of the theory will be presented in Part II of this work [8].

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 Figure 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 Figure 7, namely **Attach Case to Assembly**, **Polish Part**, **Attach Part to Assembly**, **Packaging System**, and **Path Flow Model**.

### 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 Figure 7. We have added plant model **Define New Events** which introduces events *attach\_ptA*, *attach\_ptB*, *finA\_attach*, and *finB\_attach*. These events provide a more versatile means to interact with cell 2.

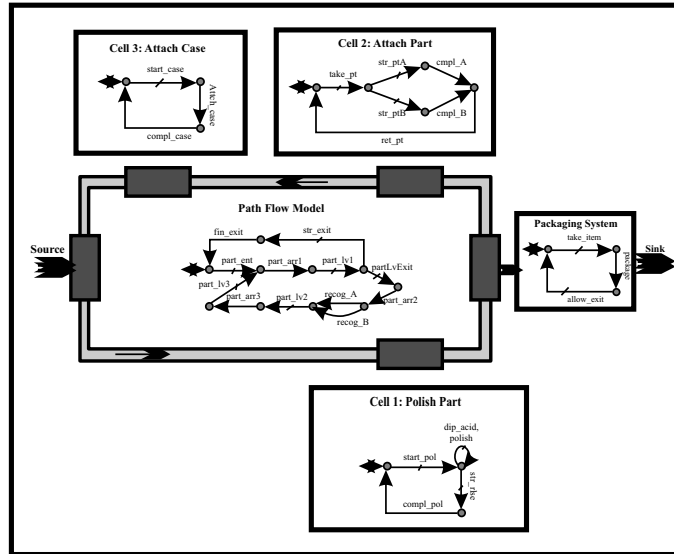


Fig. 6. Block Diagram of Plant

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

$$\Sigma_R = \{start\_pol, attach\_ptA, attach\_ptB, start\_case\}$$

$$\Sigma_A = \{comp\_pol, finA\_attach, finB\_attach, compl\_case\}$$

$$\Sigma_H = \{part\_ent, part\_arr1, part\_lv1, partLvExit, str\_exit, fin\_exit, part\_arr2, recog\_A, recog\_B, part\_lv2, part\_arr3, part\_lv3, take\_item, allow\_exit, package\}$$

$$\Sigma_L = \{take\_pt, str\_ptA, str\_ptB, compl\_A, compl\_B, ret\_pt, dip\_acid, polish, str\_rlse, attach\_case\}$$

### 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.

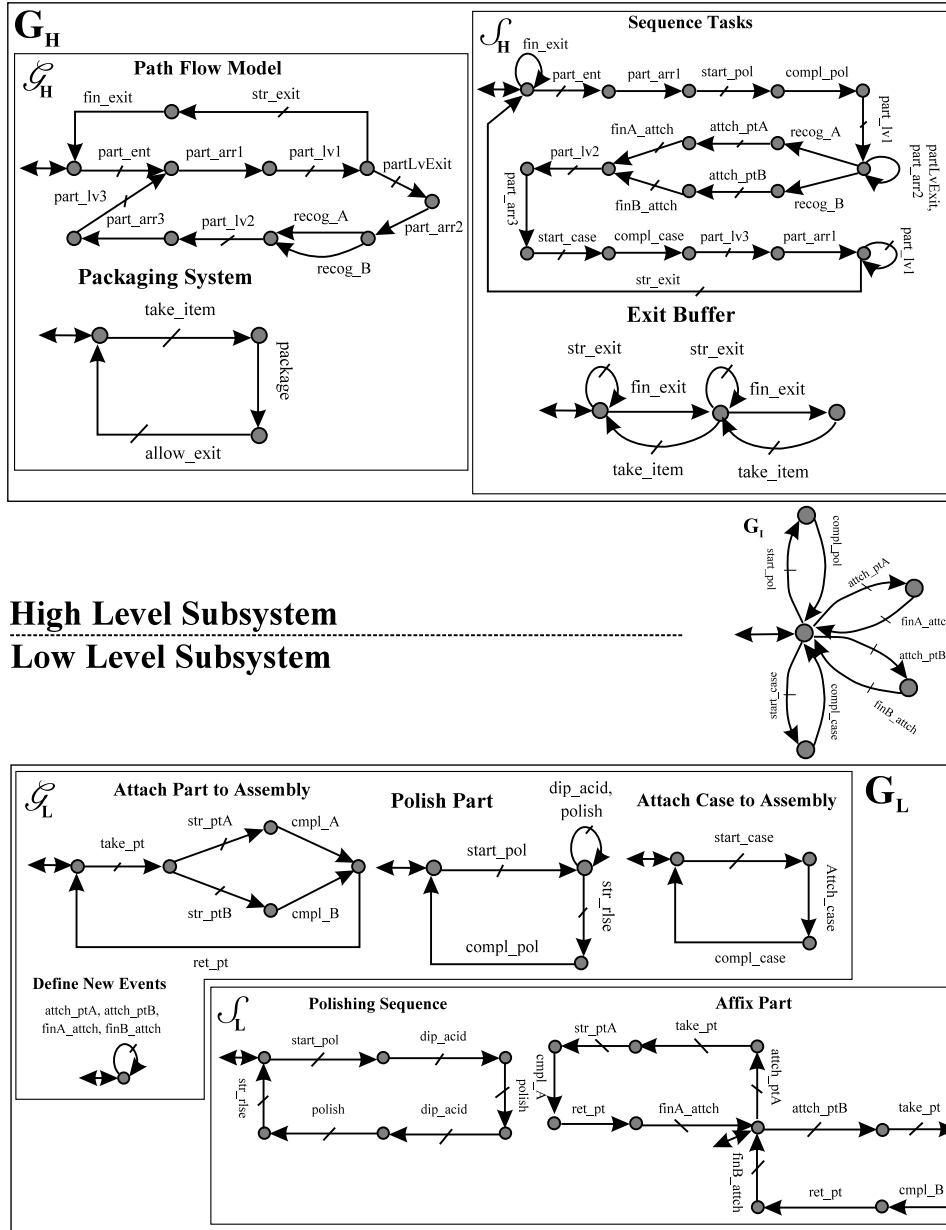


Fig. 7. Complete System Definition

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 Figure 7, and is labelled **Polishing Sequence**. For cell two, we have to provide supervisors so that the cell reacts appropriately when events *attach\_ptA* and *attach\_ptB* occur. We also must guarantee that answer events *finA\_attach* and *finB\_attach* only occur when they have the appropriate meaning. The DES **Affix Part** in Figure 7 shows how this is done.

We now design high level supervisors that use the interface. Figure 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 don't 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. The Final System*

We are now ready to define our system components. Figure 7 shows our high level subsystem, plant, and supervisor, DES  $G_H$ ,  $\mathcal{G}_H$ , and  $\mathcal{S}_H$ . We also have our low level subsystem, plant, and supervisor, DES  $G_L$ ,  $\mathcal{G}_L$ , and  $\mathcal{S}_L$ . 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. CONCLUSIONS

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.

In the Part II companion article [8] we extend the theory to handle the case when there are multiple low-level systems interacting with the high-level system. As we will see, the ability to divide the low level into multiple subsystems will allow further division of the computational burden required to guarantee controllability and nonblocking of DES while further improving reuse of DES components.

## APPENDIX

### PROOFS OF SELECTED PROPOSITIONS

Proof for **Proposition 1**: *If the system composed of DES  $G_H$ ,  $G_L$ , and  $G_I$  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:*

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

*Let  $s \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}$  (2)*

*By **Point II** of the serial level-wise nonblocking definition we have:*

$$(\exists s' \in \Sigma^*) ss' \in \mathcal{L}_m \cap \mathcal{I}_m \tag{3}$$

*Let  $l' = P_{IL}(s') \in \Sigma_{IL}^*$ .*

*We note:  $P_{IL}(ss') = P_{IL}(s)P_{IL}(P_{IL}(s')) = P_{IL}(sl')$*

*Since  $\Sigma_I \subseteq \Sigma_{IL}$ , we have:  $P_I(ss') = P_I(sl')$*

*As  $\mathcal{L}_m := P_{IL}^{-1}(L_m(G_L))$  and  $\mathcal{I}_m := P_I^{-1}(L_m(G_I))$ , (3) implies  $P_{IL}(sl') \in L_m(G_L)$  and  $P_I(sl') \in L_m(G_I)$ . We thus conclude:*

$$sl' \in \mathcal{L}_m \cap \mathcal{I}_m \quad (4)$$

If  $sl' \in \mathcal{H}$ , we can take  $l = l'$  and stop. We can thus, with no loss in generality, assume  $sl' \notin \mathcal{H}$ .

(5)

$$\Rightarrow (\exists l'' \in \Sigma_{IL}^*)(\exists \sigma \in \Sigma_{IL})(l''\sigma \leq l') \wedge (sl'' \in \mathcal{H}) \wedge (sl''\sigma \notin \mathcal{H}) \quad (6)$$

We note the following for later use:

$$\bullet \quad sl'' \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I} \quad (7)$$

$$\bullet \quad sl''\sigma \in \mathcal{I} \quad (8)$$

We now show that (6) implies  $\sigma \in \Sigma_R$ . We know:

- $\sigma \notin \Sigma_L$  as  $\sigma \in \Sigma_L$  would imply  $P_{IH}(sl''\sigma) = P_{IH}(sl'')$  and thus  $sl''\sigma \in \mathcal{H} = P_{IH}^{-1}(L(G_H))$ , which contradicts (6).
- $\sigma \notin \Sigma_A$  as  $\sigma \in \Sigma_A$ ,  $sl''\sigma \in \mathcal{I}$  (by (8)) and **Point 3** of the serial interface consistency definition would imply  $sl''\sigma \in \mathcal{H}$ , which contradicts (6).

$\Rightarrow \sigma \in \Sigma_R$  by process of elimination.

As  $\sigma \in \Sigma_R$  and  $sl''\sigma \in \mathcal{I}$  (from (8)), we have:  $P_I(sl'')\sigma \in L(G_I)$

By **Point B** of the command-pair interface definition we can conclude  $P_I(sl'') \in L_m(G_I)$  and hence:  $sl'' \in \mathcal{I}_m$

By (7) and **Point 6** of the serial interface consistency definition we can conclude:

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

We take  $l = l'''l'''$  and immediately have  $sl \in \mathcal{L}_m \cap \mathcal{I}_m$  and  $l \in \Sigma_{IL}^*$ .

Since  $l'', l''' \in \Sigma_L^*$ , we have  $P_{IH}(sl'') = P_{IH}(sl)$ . As  $\mathcal{H} := P_{IH}^{-1}(L(G_H))$ ,  $sl'' \in \mathcal{H}$  (by (6)) implies  $P_{IH}(sl) \in L(G_H)$  and we thus have  $sl \in \mathcal{H} \cap \mathcal{L}_m \cap \mathcal{I}_m$ , as required.  $\blacksquare$

**Proof for Proposition 3:** If the system composed of DES  $G_H$ ,  $G_L$ , and  $G_I$  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

$$(\forall s \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I})(\forall h \in \Sigma_H^* \cdot \Sigma_R \cdot \Sigma_H^* \cdot \Sigma_A)$$

$$sh \in \mathcal{H} \cap \mathcal{I} \Rightarrow (\exists u \in \Sigma^*)(su \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m) \wedge (P_{IH}(u) = h)$$

*Proof:*

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

Let  $s \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}$ ,  $h \in \Sigma_H^* \cdot \Sigma_R \cdot \Sigma_H^* \cdot \Sigma_A$ , and  $sh \in \mathcal{H} \cap \mathcal{I}$  (2)

We note  $h \in \Sigma_H^* \cdot \Sigma_R \cdot \Sigma_H^* \cdot \Sigma_A$  implies:

$$(\exists h' \in \Sigma_H^*)(\rho \in \Sigma_R)(h'' \in \Sigma_H^*)(\alpha \in \Sigma_A)h'\rho h''\alpha = h \quad (3)$$

As  $sh \in \mathcal{H} \cap \mathcal{I}$  (by (2)), and  $h'\rho \leq h$  (by (3)), we can conclude:

$$sh'\rho \in \mathcal{H} \cap \mathcal{I} \quad (4)$$

Since  $h' \in \Sigma_H^*$  (by (3)), we have  $P_{IL}(sh') = P_{IL}(s)$ . As  $\mathcal{L} := P_{IL}^{-1}(L(G_L))$ ,  $s \in \mathcal{L}$  (by (2)) implies that  $P_{IL}(sh') \in L(G_L)$ . We can thus immediately conclude:

$$sh' \in \mathcal{L} \quad (5)$$

As  $sh' \in \mathcal{L} \cap \mathcal{I}$ ,  $\rho \in \text{Elig}_{\mathcal{I}}(sh')$  (by (4)), and by **Point 4** of the serial interface consistency definition, we conclude:  $sh'\rho \in \mathcal{L}$

Since  $h'' \in \Sigma_H^*$  by (3), we can conclude  $P_I(sh'\rho h''\alpha) = P_I(sh'\rho\alpha)$ . As  $\mathcal{I} := P_I^{-1}(L(G_I))$ ,  $sh'\rho h''\alpha \in \mathcal{I}$  (by (2) and (3)) implies that  $P_I(sh'\rho\alpha) \in L(G_I)$  and thus:  $sh'\rho\alpha \in \mathcal{I}$

By **Point 5** of the interface consistency properties we conclude:

$$(\exists l \in \Sigma_L^*)sh'\rho l\alpha \in \mathcal{L} \cap \mathcal{I}. \quad (6)$$

As  $h'' \in \Sigma_H^*$  by (3), we have  $P_{IL}(sh'\rho l\alpha) = P_{IL}(sh'\rho l h''\alpha)$  and  $P_I(sh'\rho l\alpha) = P_I(sh'\rho l h''\alpha)$ . The definitions of  $\mathcal{L}$  and  $\mathcal{I}$  together with (6) imply that  $P_I(sh'\rho l h''\alpha) \in L(G_L) \cap L(G_I)$  and thus  $sh'\rho l h''\alpha \in \mathcal{L} \cap \mathcal{I}$  (7)

As  $\alpha \in \Sigma_A$  (by (3)), we can conclude:  $P_I(sh'\rho l h''\alpha) \in \Sigma_I^* \cdot \Sigma_A \cap L(G_I)$

Hence by **Point B** of the command-pair interface definition  $P_I(sh'\rho l h''\alpha) \in L_m(G_I)$ , so we can then conclude:

$$sh'\rho l h''\alpha \in \mathcal{I}_m \quad (8)$$

Since  $l \in \Sigma_L^*$  (by (6)), we have:  $P_{IH}(sh'\rho h''\alpha) = P_{IH}(sh'\rho l h''\alpha)$ . (9)

As  $\mathcal{H} := P_{IH}^{-1}(L(G_H))$ ,  $sh'\rho h''\alpha \in \mathcal{H}$  (from (2) and (3)) implies  $P_{IH}(sh'\rho l h''\alpha) \in L(G_H)$  and thus:  $sh'\rho l h''\alpha \in \mathcal{H}$

Taking  $u = sh'\rho l h''\alpha$  and combining with (7), (8) and (9), we have  $su \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m$  and  $P_{IH}(u) = h$ , as required. ■

*Proof for Proposition 4:* If the system composed of DES  $G_H$ ,  $G_L$ , and  $G_I$  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

$$(\forall s \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m)(\forall h \in \Sigma_{IH}^* \cdot \Sigma_A) \\ sh \in \mathcal{H} \cap \mathcal{I} \Rightarrow (\exists u \in \Sigma^*) (P_{IH}(u) = h) \wedge (su \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m)$$

*Proof:*

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

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

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

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

From (2), we have:  $sh_1 h_2 \dots h_n \in \mathcal{H} \cap \mathcal{I}$  (4)

Using an inductive proof, we will now show:

$$(\exists u_0, u_1, \dots, u_n \in \Sigma^*) (su_0 u_1 \dots u_n \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m) \wedge (P_{IH}(u_0 u_1 \dots u_n) = h_0 h_1 \dots h_n),$$

where  $h_0 := \epsilon$

Claim to be proven:

For  $k \in \{0, 1, \dots, n\}$ , there exists  $u_0, u_1, \dots, u_k \in \Sigma^*$  such that the following are true:

(5)

(a)  $P_{IH}(u_0 u_1 \dots u_k) = h_0 h_1 \dots h_k$

(b)  $su_0 u_1 \dots u_k \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m$

We will first prove the initial case ( $k = 0$ ), and then the general case of  $k \in \{1, \dots, n\}$ .

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  $k \in \{1, \dots, n\}$ . Assume  $\exists u_0, u_1, \dots, u_{k-1} \in \Sigma^*$  and that they satisfy (5) for  $k - 1$ .

(6)

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

We now note:

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

As  $\mathcal{H} := P_{IH}^{-1}(L(G_H))$ , (7) and (8) imply that  $P_{IH}(su_0u_1 \dots u_{k-1}h_k) \in L(G_H)$  and thus:  $su_0u_1 \dots u_{k-1}h_k \in \mathcal{H}$  (9)

Since  $\Sigma_I \subseteq \Sigma_{IH}$ , we can conclude by (8) that  $P_I(su_0u_1 \dots u_{k-1}h_k) = P_I(sh_0h_1 \dots h_k)$ . As  $\mathcal{I} := P_I^{-1}(L(G_I))$ , (7) implies that  $P_I(su_0u_1 \dots u_{k-1}h_k) \in L(G_I)$  and with (9):  $su_0u_1 \dots u_{k-1}h_k \in \mathcal{H} \cap \mathcal{I}$  (10)

We note that  $P_I(su_0u_1 \dots u_{k-1}) \in L_m(G_I)$  by (6). (11)

By (10), we have  $P_I(su_0u_1 \dots u_{k-1}h_k) \in L(G_I)$ . (12)

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

We have  $P_I(su_0u_1 \dots u_{k-1}) \in (\Sigma_R \cdot \Sigma_A)^* \cap L(G_I)$  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  $P_I(su_0u_1 \dots u_{k-1})$ . (14)

From (12), we have  $P_I(su_0u_1 \dots u_{k-1}h_k) = P_I(su_0u_1 \dots u_{k-1})P_I(h_k) \in L(G_I)$   
 $\Rightarrow P_I(h_k) \in \Sigma_R \cdot \Sigma_R^* \cdot \Sigma_A$  by (3) and (14).

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

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

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

$$(\exists u' \in \Sigma^*)su_0u_1 \dots u_{k-1}u' \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m \wedge (P_{IH}(u') = h_k)$$

$\Rightarrow P_{IH}(u_0u_1 \dots u_{k-1}u') = h_0h_1 \dots h_k$  by **(6)**.

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

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

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

*Proof for Proposition 5:* If the system composed of DES  $G_H$ ,  $G_L$ , and  $G_I$  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

$$(\forall s \in \mathcal{H} \cap \mathcal{L} \cap \mathcal{I}_m)(\forall h \in \Sigma_{IH}^*)$$

$$sh \in \mathcal{H}_m \cap \mathcal{I}_m \Rightarrow (\exists u \in \Sigma^*)(su \in \mathcal{H}_m \cap \mathcal{L}_m \cap \mathcal{I}_m) \wedge (P_{IH}(u) = h) \wedge (P_I(u) \in \{\epsilon\} \cup \Sigma_R \cdot \Sigma_I^*)$$

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

**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  $\mathcal{L} := P_{IL}^{-1}(L(G_L))$ ,  $s \in \mathcal{L}$  (from (2)) implies that  $P_{IL}(sh) \in L(G_L)$  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.

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:  $P_I(s) \in L_m(G_I)$

$\Rightarrow P_I(s) \in (\Sigma_R \cdot \Sigma_A)^* \cap L(G_I)$  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:  $P_I(s)P_I(h) \in L_m(G_I)$

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:

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

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  $\mathcal{L} := P_{IL}^{-1}(L(G_L))$ ,  $su' \in \mathcal{L}$  (from (10)) implies that  $P_{IL}(su'h'') \in L(G_L)$  and thus:

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

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

$$su'h'' \in \mathcal{H}_m \quad (14)$$

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

$$(\exists l \in \Sigma_L^*)su'h''l \in \mathcal{H}_m \cap \mathcal{L}_m \cap \mathcal{I}_m \quad (15)$$

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  $P_I(u) \in \Sigma_R \cdot \Sigma_I^*$  by **(9)** **(11)** **(15)**.

**Case II** complete.

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

*Proof for Proposition 6:* If the system composed of plant components  $\mathcal{G}_H$ ,  $\mathcal{G}_L$ , supervisors  $\mathcal{S}_H$ ,  $\mathcal{S}_L$ , and interface  $G_I$ , 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:

$$(\forall s \in L(\mathbf{Plant}) \cap \mathbf{L}_S \cap \mathcal{I}) \quad \text{Elig}_{L(\mathbf{Plant})}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathbf{L}_S \cap \mathcal{I}}(s)$$

*Proof:*

Assume system is serial level-wise controllable (1)

Let  $s \in L(\mathbf{Plant}) \cap \mathbf{L}_S \cap \mathcal{I}$  and  $\sigma \in \text{Elig}_{L(\mathbf{Plant})}(s) \cap \Sigma_u$  (2)

From **(2)**, we have  $s, s\sigma \in L(\mathbf{Plant}) = \mathbf{H} \cap \mathbf{L}$

Also using **(2)**, we can now conclude  $s \in \mathbf{L} \cap \mathbf{L}_S \cap \mathcal{I}$  and  $\sigma \in \text{Elig}_{\mathbf{L}}(s) \cap \Sigma_u$

Using **(1)**, we can conclude by **Point II** of the serial level-wise controllable definition that  $\sigma \in \text{Elig}_{\mathbf{L}_S \cap \mathcal{I}}(s)$ , as required. ■

## REFERENCES

- [1] D. M. Hoffman and D. M. Weiss, Eds., *Software Fundamentals. Collected Papers by David L. Parnas.* Addison Wesley, 2001.
- [2] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. Dec., pp. 1053–1058, Dec. 1972.
- [3] —, "Use of abstract interfaces in the development of software for embedded computer systems," Naval Research Laboratory, NRL Report 8047, 1977.
- [4] D. L. Parnas, P. C. Clements, and D. M. Weiss, "The modular structure of complex systems," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 3, pp. 259–66, Mar. 1985.
- [5] R. Leduc, B. Brandin, and W. M. Wonham, "Hierarchical interface-based non-blocking verification," in *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, May 2000, pp. 1–6.
- [6] E. W. Endsley, M. R. Lucas, and D. M. Tilbury, "Modular design and verification of logic control for reconfigurable machining systems," submitted to *Discrete Event Dynamic Systems: Theory and Applications*.
- [7] R. Leduc, B. Brandin, W. M. Wonham, and M. Lawford, "Hierarchical interface-based supervisory control: Serial case," in *Proc. of 40th Conf. Decision Contr.*, Orlando, USA, December 2001, pp. 4116–4121.

- [8] R. Leduc, M. Lawford, and W. M. Wonham, "Hierarchical interface-based supervisory control, part II: Parallel case," submitted to *IEEE Trans. Automatic Control*, 2003.
- [9] R. Leduc, "Hierarchical interface-based supervisory control," Ph.D. dissertation, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 2002.
- [10] L. de Alfaro and T. A. Henzinger, "Interface automata," in *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering*. ACM Press, 2001, pp. 109–120.
- [11] N. Lynch and M. Tuttle, "Hierarchical correctness proofs for distributed algorithms," 1987. [Online]. Available: [citeseer.nj.nec.com/tuttle87hierarchical.html](http://citeseer.nj.nec.com/tuttle87hierarchical.html)
- [12] M. Fabian, "On object-oriented non-deterministic supervisory control," Ph.D. dissertation, Chalmers University of Technology, Goteborg, Sweden, 1995.
- [13] M. Fabian and B. Lennartson, "Object oriented supervisory control with a class of nondeterministic specifications," in *Proc. 33th Conf. Decision Contr.*, Buena Vista, Florida USA, December 1994, pp. 3634–3635.
- [14] —, "Petri nets and control synthesis; an object oriented approach," in *Proc. I.M.S.*, Vienna, Austria, June 1994.
- [15] M. Shayman and R. Kumar, "Process objects/masked composition: an object-oriented approach for modeling and control of discrete-event systems," *IEEE Trans. Automatic Control*, vol. 44, no. 10, pp. 1864–1869, 1999.
- [16] W. M. Wonham, *Notes on Control of Discrete-Event Systems*, Department of Electrical and Computer Engineering, University of Toronto, July 2003, Notes and CTCT software can be downloaded at <http://www.control.toronto.edu/DES/>.
- [17] M. Courvoisier, M. Combacau, and A. de Bonneval, "Control and monitoring of large discrete event systems: a generic approach," in *Proc. of ISIE 93*, Budapest, 1993, pp. 571–576.
- [18] M. Queiroz and J. Cury, "Modular supervisory control of large scale discrete event systems," in *Proceedings of WODES 2000*, Ghent, Belgium, Aug 2000, pp. 103–110.
- [19] G. Stremersch and R. Boel, "Decomposition of the supervisory control problem for Petri nets under preservation of maximal permissiveness," *IEEE Trans. Automatic Control*, vol. 46, no. 9, pp. 1490–1496, 2001.
- [20] N. Alsop, "Formal techniques for the procedural control of industrial processes," Ph.D. dissertation, Department of Chemical Engineering and Chemical Technology, Imperial College of Science, Technology and Medicine, London, 1996.
- [21] R. Leduc, "PLC implementation of a DES supervisor for a manufacturing testbed: An implementation perspective," Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1996.
- [22] S. Chen, "Existence and design of supervisors for vector discrete event systems," Master's thesis, Department of Electrical Engineering, University of Toronto, Toronto, Ont, 1992.
- [23] —, "Control of discrete-event systems of vector and mixed structural type," Ph.D. dissertation, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1996.
- [24] Y. Li, "Control of vector discrete-event systems," Ph.D. dissertation, Department of Electrical Engineering, University of Toronto, Toronto, Ont, 1991.
- [25] J. O. Moody and P. J. Antsaklis, *Supervisory Control of Discrete Event Systems using Petri Nets*. Kluwer Academic Publishers, 1998.
- [26] M. Zhou, D. Wang, and I. Mayk, "Using petri nets for object-oriented design of command and control systems," *International Journal of Intelligent Control and Systems*, vol. 2, no. 2, pp. 287–300, 1998.
- [27] M. Zhou and F. DiCesare, *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Kluwer Academic Publishers, 1993.

- [28] 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.
- [29] G. Barrett and S. Lafortune, "Decentralized supervisory control with communicating controllers," *IEEE Trans. Automatic Control*, vol. 45, no. 9, pp. 1620–1638, 2000.
- [30] F. Lin and W. M. Wonham, "Decentralized control and coordination of discrete-event systems with partial observations," in *Proc. 27th IEEE Conf. Decision Contr.*, Dec 1988, pp. 1125–1130.
- [31] K. Rudie and J. C. Willems, "The computational complexity of decentralized discrete-event control problems," *IEEE Trans. Automatic Control*, vol. 44, no. 7, pp. 1313–1319, 1995.
- [32] K. Rudie and W. M. Wonham, "Think globally, act locally: decentralized supervisory control," *IEEE Trans. on Automatic Control*, vol. 37, no. 11, pp. 1692–1708, Nov 1992, reprinted in F.A. Sadjadi (Ed.), *Selected Papers on Sensor and Data Fusion*, 1996; ISBN 0-8194-2265-7.
- [33] K. Wong and J. van Schuppen, "Decentralized supervisory control of discrete event systems with communication," in *Proc. of WODES 1996*, Edinburgh, UK, Aug 1996, pp. 284–289.
- [34] T. Yoo and S. Lafortune, "A general architecture for decentralized supervisory control of discrete-event systems," in *Proc. of WODES 2000*, Ghent, Belgium, Aug 2000, pp. 111–118.
- [35] Y. Brave and M. Heymann, "Control of discrete event systems modeled as hierarchical state machines," *IEEE Trans. on Automatic Control*, vol. 38, no. 12, pp. 1803–1819, Dec 1993.
- [36] P. Gohari-Moghadam, "A linguistic framework for controlled hierarchical DES," Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1998.
- [37] H. Liu, J. Park, and R. Miller, "On hybrid synthesis for hierarchical structured petri nets," Department of Computer Science, University of Maryland, College Park, MD, Tech. Rep., 1996.
- [38] C. Ma, "A computational approach to top-down hierarchical supervisory control of DES," Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1999.
- [39] B. Wang, "Top-down design for RW supervisory control theory," Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1995.
- [40] C. Ma and W. M. Wonham, "Control of state tree structures," in *Proc. 11th Mediterranean Conference on Control and Automation*, June 2003, paper T4-005 (6pp.).
- [41] R. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, 1986.
- [42] R. Alur and T. Henzinger, "Local liveness for compositional modelling of fair reactive systems," in *Proc. of seventh Int. Conf. on Computer-aided Verification, Lecture Notes in Computer Science*, 1995, pp. 166–179.
- [43] A. Aziz, V. Singhal, and G. Swamy, "Minimizing interacting finite state machines: A compositional approach to language containment," in *Proc. of IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors*, Cambridge, Massachusetts, Oct 1994, pp. 255–261.
- [44] P. Caines and Y. Wei, "The hierarchical lattices of a finite machine," *Systems Control Letters*, vol. 25, pp. 257–263, July 1995.
- [45] H. Chen and H.-M. Hanisch, "Model aggregation for hierarchical control synthesis of discrete event systems," in *Proc. 39th Conf. Decision Contr.*, Sydney, Australia, December 2000, pp. 418–423.
- [46] 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 Contr.*, Orlando, USA, December 2001, pp. 4110–4115.

- [47] J. M. Eyzell and J. E. Cury, "Exploiting symmetry in the synthesis of supervisors for discrete event systems," in *Proc. of American Control Conference*, Philadelphia, USA, June 1998, pp. 244–248.
- [48] O. Grümberg and D. Long, "Model checking and modular verification," in *Proc. of CONCOUR'91*, ser. LNCS, no. 527. Springer-Verlag, 1991, pp. 361–375.
- [49] P. Hubbard and P. E. Caines, "Trace-DC hierarchical supervisory control with applications to transfer-lines," in *Proc. 37th Conf. Decision Contr.*, Tampa, Florida USA, December 1998, pp. 3293–3298.
- [50] K. Q. Pu, "Modeling and control of discrete-event systems with hierarchical abstraction," Master's thesis, Dept. of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 2000.
- [51] 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. Systems, Man, and Cybernetics, Part A*, vol. 29, no. 6, pp. 573–586, 1999.
- [52] G. Shen and P. E. Caines, "Hierarchically accelerated dynamic programming for finite-state machines," *IEEE Trans. Automatic Control*, vol. 47, no. 2, pp. 271–283, 2002.
- [53] K. Wong, "Discrete-event control architecture: An algebraic approach," Ph.D. dissertation, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1994.
- [54] W. Wu, H. Su, J. Chu, and H. Zhai, "Hierarchical control of DES based on colored petri nets," in *Proc. of IEEE Systems, Man, and Cybernetics*, vol. 3, 2001, pp. 1571–1576.
- [55] H. Zhong and W. M. Wonham, "On the consistency of hierarchical supervision in discrete-event systems," *IEEE Trans. on Automatic Control*, vol. 35, no. 10, pp. 1125–1134, Oct 1990.
- [56] J. Gunnarsson, "Symbolic methods and tools for discrete event dynamic systems," Ph.D. dissertation, Department of Electrical Engineering at Linköping University, Sweden, 1997.
- [57] Z. Zhang, "Smart TCT: an efficient algorithm for supervisory control design." Master's thesis, Dept. of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 2001.
- [58] Z. Zhang and W. M. Wonham, "STCT: an efficient algorithm for supervisory control design," in *Proc. of SCODES 2001*, INRIA, Paris, July 2001.
- [59] A. Arnold, *Finite Transition Systems*. Prentice Hall, 1994.
- [60] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 2001.
- [61] E. M. Clarke, E. Emerson, and A. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Trans. Programming Languages and Systems*, vol. 8, no. 2, pp. 244–263, Apr. 1986.
- [62] J. Burch, E. M. Clarke, and K. McMillan, "Symbolic model checking:  $10^{20}$  states and beyond," *Information and Computation*, vol. 98, pp. 142–170, 1992.
- [63] K. McMillan, *Symbolic Model Checking*. Kluwer, 1992.
- [64] S. Berezin, S. Campos, and E. M. Clarke, "Compositional reasoning in model checking," in *COMPOS'97*, ser. LNCS, vol. 1536. Springer-Verlag, 1998, pp. 81–102.
- [65] 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.
- [66] 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.
- [67] B. Brandin and F. Charbonnier, "The supervisory control of the automated manufacturing system of the AIP," in *Proc. Rensselaer's 1994 Fourth International Conference on Computer Integrated Manufacturing and Automation Technology*, Troy, Oct 1994, pp. 319–324.