Hierarchical Interface-Based Supervisory Control of a Flexible Manufacturing System

Ryan J. Leduc, *Member, IEEE*, Mark Lawford, *Member, IEEE*, and Pengcheng Dai

***Abstract*—Flexible manufacturing systems have long been touted as an application area for supervisory control theory. Unfortunately, due to the typical exponential growth of state space with the number of interacting subsystems, concurrent systems such as manufacturing applications have, for the most part, remained beyond the reach of existing supervisory control theory tools. This paper demonstrates how, by imposing a hierarchical, modular, interface-based architecture on the system, significant gains can be made in the size of applications that can be handled by supervisory control theory. We first review hierarchical interface-based supervisory control, providing the theory necessary to motivate the creation of well-defined automata-based interfaces between components. This architecture permits the verification of global safety (controllability) and nonblocking properties to be decomposed into a set of local checks, each of which only involves an individual component subsystem and its interface automata. The paper then provides a detailed description of how the theory can be applied to the design and verification of a flexible manufacturing system work cell. The work cell model is based on the Atelier Interétablissement de Productique flexible manufacturing workcell, a system that has been previously studied in the literature with limited success.**

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

I. Introduction

In SUPERVISORY control of discrete-event systems (DES), two common tasks are to verify that a composite system, based on a Cartesian product of subsystems, is: 1) nonblocking (a form of liveness) and 2) controllable (a form of safety). 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 (modular control [1]–[5], decentralized control [6]–[10] vector DES (VDES) [5], [11], [12] and petri nets (PN) [13], [14], model aggregation methods—e.g., [15]–[20], and multilevel hierarchy [21]–[25]), large-scale systems are still problematic, particularly for verification of nonblocking.

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. As we will demonstrate with an example, when the design heuristic is followed in the construction of a large-scale system, our theoretical results can be applied to verify the nonblocking and controllability of the complete system by checking local conditions. 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.

In [26]–[28], we developed hierarchical interface-based supervisory control (HISC), a method that decomposes a system into a high-level subsystem which communicates with $n \geq 1$ parallel low-level subsystems through separate interfaces that restrict the interaction of the subsystems. Each subsystem is modeled as a deterministic finite state automaton that is capable of generating different sequences of events. The interfaces are also modeled as automata, with the results that the past history of communication between components influences the current and future possible communications. Each interface between a pair of components effectively establishes a protocol for interaction of the connected subsystems. As a result of this structured interaction, it is possible to derive a set of local consistency properties that can be used to verify if a discrete-event system is globally nonblocking and controllable. Each of these consistency properties can be verified using a single subsystem and its interface(s); thus, the complete system model never needs to be stored in memory or traversed, offering potentially significant savings in computational resources.

In this paper, we show how HISC can be applied to the verification of a controller for a model of a relatively complex flexible manufacturing system. The model is based on the Atelier Inter-établissement de Productique (AIP), an automated manufacturing system consisting of a central loop and four external loops, three assembly stations, an input/output (I/O) station, and four inter-loop transfer units. The estimated worst case state space size of the plant model is 2×10^{43}. As a result, controller synthesis techniques that required the construction of the entire plant, such as most of those referenced above, could not be utilized to design a verifiably correct controller. Therefore, after modeling the open loop system, local controllers were designed based on heuristics and experience and then verified to result in globally nonblocking, controllable closed loop behavior using the HISC method.

Despite the limitations of existing controller synthesis techniques, when the architecture required by HISC is imposed upon the system, supervisory control theory still provides a useful framework for the design and verification of a complex system. In this paper, our goals are to provide a detailed example that

Manuscript received January 28, 2004; revised November 30, 2005. Manuscript received in final form March 7, 2006. Recommended by Associate Editor S. Kowalewski.

The authors are with the Department of Computing and Software, McMaster University, Hamilton, ON L8S 4K1, Canada (e-mail: leduc@mcmaster.ca; lawford@mcmaster.ca; daip@mcmaster.ca).

Digital Object Identifier 10.1109/TCST.2006.876635

serves effectively as a tutorial on how HISC might be applied to similar systems and to motivate further research on interface based supervisor synthesis methods.

Section II provides an overview of the HISC results of [26]–[28]. It defines interfaces, describes how they are used to restrict the information flow of the system, and then provides a set of local consistency properties that can be used to verify if the system is globally nonblocking and controllable. Section III provides an overview of the AIP system and describes the desired closed loop behavior of the system under a particular set of assumptions. We describe the modular decomposition (in the sense of [29]) of the system and provide the details of the component interfaces and supervisor design in Section IV. We close with Section V discussing the results of applying the method to the AIP system and Section VI drawing general conclusions and motivating future work while discussing the method's limitations.

We note here that the interface conditions that we present in this work are a modified version of the conditions used in [26]–[28]. The new set of conditions are more concise and clear, particularly with respect to what checks need to be performed on a given component. We first introduced the new conditions in [30], where we showed that our definitions are equivalent to the ones given in [26]–[28].

II. Overview of HISC

We begin with a brief review of supervisory control theory and then provide the HISC theory needed for the AIP example.

A. Discrete-Event Systems Preliminaries

Supervisory control theory [5], [31], [32] provides a framework for the control of discrete-event systems (DES), systems that are discrete in space and time. For a detailed exposition of DES, see [5]. Below, 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 language L (denoted $\overline{L}$) is defined as $\overline{L} = \{t \in \Sigma^* | t \leq s \text{ for some } s \in L\}$. Let $\mathrm{Pwr}(\Sigma)$ denote the power set of Σ. For language L, the eligibility operator $\mathrm{Elig}_L : \Sigma^* \to \mathrm{Pwr}(\Sigma)$ is given by $\mathrm{Elig}_L(s) := \{\sigma \in \Sigma | s\sigma \in L\}$ for $s \in \Sigma^*$.

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 $\delta : Y \times \Sigma \to Y$ is the transition function, y_o is the initial state, and Y_m is the set of marker states. 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 $\mathbf{G}$ is defined as $L_m(\mathbf{G}) := \{s \in L(G) | \delta(y_o, s) \in Y_m\}$. The reachable state subset of DES $\mathbf{G}$, denoted Y_r, is $Y_r := \{y \in Y | (\exists s \in \Sigma^*)\delta(y_o, s) = y\}$. A DES $\mathbf{G}$ is reachable if $Y_r = Y$. We will always assume $\mathbf{G}$ is reachable.

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

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

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 $P_i^{-1} : \mathrm{Pwr}(\Sigma_i^*) \to \mathrm{Pwr}(\Sigma^*)$ is the inverse image function of P_i (see, e.g., [5]).

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]

$$L_m(\mathbf{G}) = L_m(\mathbf{G}_1) || L_m(\mathbf{G}_2), \quad L(\mathbf{G}) = L(\mathbf{G}_1) || L(\mathbf{G}_2)$$

and event set $\Sigma = \Sigma_1 \cup \Sigma_2$.

For DES, the two main properties we want to check are nonblocking and controllability. A DES $\mathbf{G}$ is said to be *nonblocking* if $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$, i.e., any string can always be continued to a string resulting in a marked state.

To control the plant, we define a supervisor. A supervisor is represented as an automaton $\mathbf{S} = (X, \Sigma_S, \xi, x_o, X_m)$.

The synchronous product operator is used to specify the closed loop behavior of the system. The behavior of a plant $\mathbf{G}$ under the control of a supervisor $\mathbf{S}$ is, thus, $\mathbf{System} := \mathbf{G} || \mathbf{S}$.

We will denote the disjoint union of sets by $\dot{\cup}$ and adopt the standard partition $\Sigma = \Sigma_u \dot{\cup} \Sigma_c$, splitting our alphabet into uncontrollable and controllable events. The formal definition for controllability follows.

Definition 1: Let $\Sigma := \Sigma_1 \cup \Sigma_S$, $P_1 : \Sigma^* \to \Sigma_1^*$ and $P_S : \Sigma^* \to \Sigma_S^*$. Define $\mathcal{L}_{\mathbf{G}_1} := P_1^{-1}(L(\mathbf{G}_1))$ and $\mathcal{L}_{\mathbf{S}} := P_S^{-1}(L(\mathbf{S}))$. A supervisor $\mathbf{S}$ is *controllable* for a plant $\mathbf{G}_1$, if $\mathcal{L}_{\mathbf{S}} \Sigma_u \cap \mathcal{L}_{\mathbf{G}_1} \subseteq \mathcal{L}_{\mathbf{S}}$ or, equivalently, $(\forall s \in \mathcal{L}_{\mathbf{G}_1} \cap \mathcal{L}_{\mathbf{S}}) \mathrm{Elig}_{\mathcal{L}_{\mathbf{G}_1}}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\mathcal{L}_{\mathbf{S}}}(s)$. Thus, a supervisor is controllable for a plant if no uncontrollable plant actions can take the plant outside of the supervisor's specified behavior.

B. Hierarchical Interface Based Supervisory Control

In HISC there is a master–slave relationship. A high-level subsystem sends a command to a particular low-level subsystem, which then performs the indicated task and returns an answer. Fig. 1 shows conceptually the structure and information flow of the system in the special case when there is only a single low-level subsystem. Communication between the high-level subsystem and the low-level subsystem occurs in a serial fashion. A request from the high-level is followed by an answer from the low-level before the next request is issued to

[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.

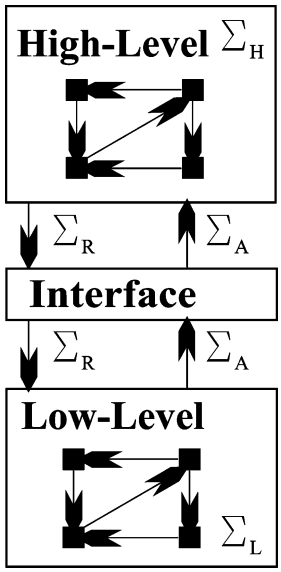


Fig. 1. Interface block diagram.

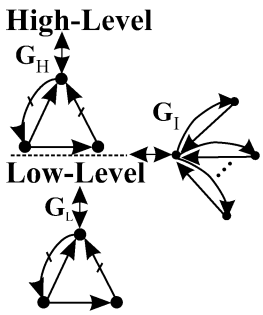


Fig. 2. Two-tiered structure of the system.

the low-level subsystem. This style of interaction is enforced by an interface that mediates communication between the two subsystems. All system components, including the interface, are modeled as automata as shown in Fig. 2 where our flat system would be $\mathbf{G} := \mathbf{G}_H \| \mathbf{G}_I \| \mathbf{G}_L$. By flat system, we mean the equivalent DES if we ignored the interface structure.

In order to restrict information flow and decouple the subsystems, the event set Σ is split into four disjoint alphabets: Σ_H, Σ_L, Σ_R, and Σ_A. The events in Σ_H are 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, $\mathbf{G}_H$ and $\mathbf{G}_L$, respectively. We then have $\mathbf{G}_H$ defined over $\Sigma_H \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A$ and $\mathbf{G}_L$ defined over $\Sigma_L \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A$.

As the interface automaton $\mathbf{G}_I$ is only concerned with communication between the two subsystems, it is defined over the events that are common to both levels of the hierarchy $\Sigma_R \dot{\cup} \Sigma_A$, which are collectively known as the set of interface events, denoted Σ_I. The events in Σ_R, called request events, 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 subsystem's responses to the request events. In order to enforce the serialization of requests and answers, we restrict the interface to the subclass of command-pair interfaces defined below.

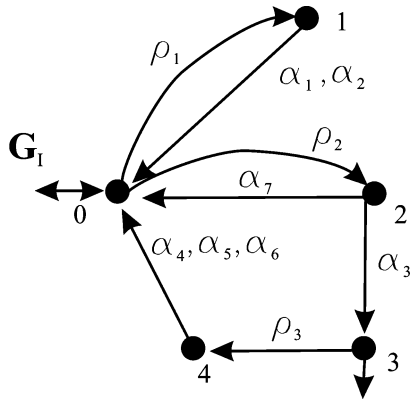


Fig. 3. Example interface.

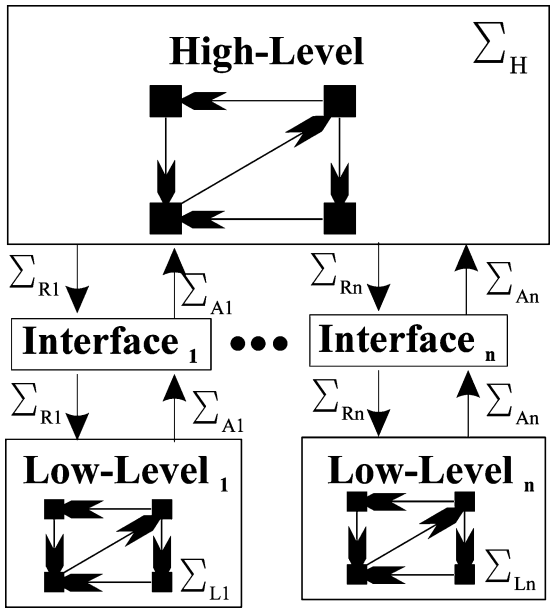


Fig. 4. Parallel interface block diagram.

Definition 2: A DES $\mathbf{G}_I = (X, \Sigma_R \dot{\cup} \Sigma_A, \xi, x_o, X_m)$ is a command-pair interface if:

1) $L(\mathbf{G}_I) \subseteq \overline{(\Sigma_R.\Sigma_A)^*}$, and
2) $L_m(\mathbf{G}_I) = (\Sigma_R.\Sigma_A)^* \cap L(\mathbf{G}_I)$.

Condition 1 says that request events and answer events must alternate (i.e., serialization of requests) while condition 2 states that every answered request results in a marked state. As we require $\mathbf{G}_I$ to be expressible as a tuple, it must have an initial state. It follows that $\epsilon \in L_m(\mathbf{G}_I)$ by condition 2. An example command pair interface with $\Sigma_R := \{\rho_i | i = 1, 2, 3\}$ and $\Sigma_A := \{\alpha_i | i = 1, \ldots, 7\}$ is shown in Fig. 3.

We now generalize the above "serial case," where there is a single low-level subsystem to the parallel case, where there are n low-level subsystems. In this case we say that the system is an nth degree parallel system. Fig. 4 shows conceptually the structure and flow of information. The single high-level subsystem interacts with $n \geq 1$ independent low-level subsystems, communicating with each low-level subsystem in parallel through a separate interface.

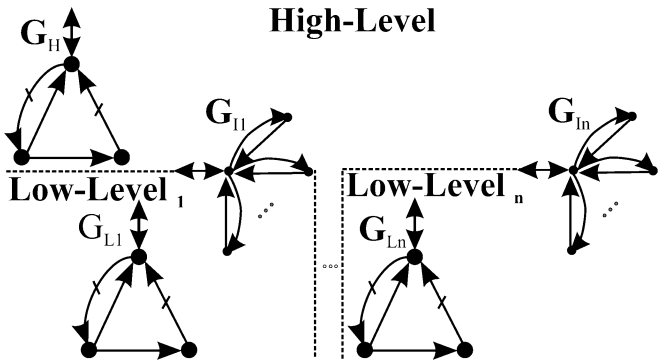


Fig. 5. Two-tiered structure of the system.

As in the serial case, to restrict the flow of information at the interface, we partition the system alphabet into pairwise disjoint alphabets

$$\Sigma := \Sigma_H \dot{\cup} \dot{\bigcup_{j=1,\ldots,n}} \left[\Sigma_{L_j} \dot{\cup} \Sigma_{R_j} \dot{\cup} \Sigma_{A_j}\right]. \tag{1}$$

The high-level subsystem is modeled by DES $\mathbf{G}_H$ (defined over event set $\Sigma_H \dot{\cup} (\dot{\cup}_{j\in\{1,\ldots,n\}}[\Sigma_{R_j} \dot{\cup} \Sigma_{A_j}])$). For $j \in \{1,\ldots,n\}$, the jth low-level subsystem is modeled by DES $\mathbf{G}_{L_j}$ (defined over event set $\Sigma_{L_j} \dot{\cup} \Sigma_{R_j} \dot{\cup} \Sigma_{A_j}$), and the jth interface by DES $\mathbf{G}_{I_j}$ (defined over event set $\Sigma_{R_j} \dot{\cup} \Sigma_{A_j}$). The overall system has the structure shown in Fig. 5. Thus, our flat system is $\mathbf{G} = \mathbf{G}_H \| \mathbf{G}_{I_1} \| \mathbf{G}_{L_1} \| \ldots, \| \mathbf{G}_{I_n} \| \mathbf{G}_{L_n}$.

To simplify notation in our exposition, we bring in the following event sets, natural projections, and languages. For the remainder of this section, the index j has range $\{1,\ldots,n\}$

$$\begin{aligned}
\Sigma_{I_j} &:= \Sigma_{R_j} \cup \Sigma_{A_j}, & P_{I_j} &: \Sigma^* \to \Sigma_{I_j}^* \\
\Sigma_{IL_j} &:= \Sigma_{L_j} \cup \Sigma_{I_j}, & P_{IL_j} &: \Sigma^* \to \Sigma_{IL_j}^* \\
\Sigma_{IH} &:= \Sigma_H \cup \bigcup_{k\in\{1,\ldots,n\}} \Sigma_{I_k} & P_{IH} &: \Sigma^* \to \Sigma_{IH}^* \\
\mathcal{H} &:= P_{IH}^{-1}(L(\mathbf{G}_H)), & \mathcal{H}_m &:= P_{IH}^{-1}(L_m(\mathbf{G}_H)) \subseteq \Sigma^* \\
\mathcal{L}_j &:= P_{IL_j}^{-1}(L(\mathbf{G}_{L_j})), & \mathcal{L}_{m_j} &:= P_{IL_j}^{-1}(L_m(\mathbf{G}_{L_j})) \subseteq \Sigma^* \\
\mathcal{I}_j &:= P_{I_j}^{-1}(L(\mathbf{G}_{I_j})), & \mathcal{I}_{m_j} &:= P_{I_j}^{-1}(L_m(\mathbf{G}_{I_j})) \subseteq \Sigma^*.
\end{aligned}$$

We now present the properties that the system must satisfy to ensure that it interacts with the interfaces correctly.

Definition 3: The nth degree $(n \geq 1)$ parallel interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is interface consistent with respect to the alphabet partition given by (1), if for all $j \in \{1,\ldots,n\}$, the following conditions are satisfied.

Multilevel Properties:

1) The event set of $\mathbf{G}_H$ is Σ_{IH}, and the event set of $\mathbf{G}_{L_j}$ is Σ_{IL_j}.
2) $\mathbf{G}_{I_j}$ is a command-pair interface.

High-Level Property:

3) $(\forall s \in \mathcal{H} \cap \bigcap_{k=1,\ldots,n} \mathcal{I}_k)$

$$\mathrm{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{A_j} \subseteq \mathrm{Elig}_{\mathcal{H} \cap \bigcap_{k\neq j} \mathcal{I}_k}(s).$$

Low-Level Properties:

4) $(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j) \mathrm{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{R_j} \subseteq \mathrm{Elig}_{\mathcal{L}_j}(s)$.
5) $(\forall s \in \Sigma^*.\Sigma_{R_j} \cap \mathcal{L}_j \cap \mathcal{I}_j)$.

$$\mathrm{Elig}_{\mathcal{L}_j \cap \mathcal{I}_j}\left(s\Sigma_{L_j}^*\right) \cap \Sigma_{A_j} = \mathrm{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{A_j} \quad \text{where}$$
$$\mathrm{Elig}_{\mathcal{L}_j \cap \mathcal{I}_j}\left(s\Sigma_{L_j}^*\right) := \bigcup_{l \in \Sigma_{L_j}^*} \mathrm{Elig}_{\mathcal{L}_j \cap \mathcal{I}_j}(sl).$$

6) $(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j)$

$$s \in \mathcal{I}_{m_j} \Rightarrow \left(\exists l \in \Sigma_{L_j}^*\right) sl \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}.$$

The first two properties assert that the system has the required basic architecture, with the high-level and low-level subsystems only sharing request and answer events and the interaction between the levels mediated by interfaces. This provides a form of information hiding as it restricts the high-level subsystem from knowing (and directly affecting) internal details of the low-level subsystems and *vice versa*.

The high-level property (3) asserts that when $\mathbf{G}_H$ is synchronized with all of the other subsystem interfaces $\mathbf{G}_{I_k}, k \neq j$, it must always accept an answer event if the event is eligible in the interface $\mathbf{G}_{I_j}$. 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. Similarly, low-level property (4) asserts that the low-level subsystem $(\mathbf{G}_{L_j})$ must always accept a request event if the event is eligible in its interface $\mathbf{G}_{I_j}$. We note that both (3) and (4) can be computed using the standard algorithms for controllability.

Condition (5) states that immediately after a request event (some $\rho \in \Sigma_{R_j}$) has occurred, and before it is followed by any low-level events in Σ_{L_j}, there exists one or more paths via strings in $\Sigma_{L_j}^*$ to each answer event that $\mathbf{G}_{I_j}$ says can follow the request event. Finally, (6) asserts that every string marked by the interface $\mathbf{G}_{I_j}$ and accepted by the low-level subsystem, can be extended by a low-level string to a string marked by $\mathbf{G}_{L_j}$.

C. Local Conditions for Global Nonblocking of the System

We now provide the conditions that the subsystems and their interface(s) must satisfy in addition to the interface consistency properties, if the system $\mathbf{G}$ is to be nonblocking.

Definition 4: The nth degree $(n \geq 1)$ parallel interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is said to be level-wise nonblocking if the following conditions are satisfied.

1) *Nonblocking at the high-level*:

$$\overline{\mathcal{H}_m \cap \bigcap_{k=1,\ldots,n} \mathcal{I}_{m_k}} = \mathcal{H} \cap \bigcap_{k=1,\ldots,n} \mathcal{I}_k.$$

2) *Nonblocking at the low-level*: for all $j \in \{1,\ldots,n\}$

$$\overline{\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}} = \mathcal{L}_j \cap \mathcal{I}_j.$$

The above definition can be paraphrased as saying that for each component subsystem synchronized with its interface(s), every reachable state must have a path to a state that is marked by both the subsystem and its interface(s). We are now ready to present our nonblocking theorem for parallel interface systems.

Theorem 1: If the nth degree $(n \geq 1)$ parallel interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is level-wise nonblocking and interface consistent with respect to the alphabet partition given by (1), then $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$, where $\mathbf{G} = \mathbf{G}_H \| \mathbf{G}_{I_1} \| \mathbf{G}_{L_1} \| \ldots \| \mathbf{G}_{I_n} \| \mathbf{G}_{L_n}$.

Proof: See proof in [30]. ∎

D. Local Conditions for Global Controllability of the System

The representation of the system given in Fig. 5 simplifies notation when verifying nonblocking by ignoring the distinction between plants and supervisors. For controllability, we need to split the subsystems into their plants and supervisor components. To do this, we 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 jth low-level plant and supervisor are $\mathbf{G}_{L_j}^p$ and $\mathbf{S}_{L_j}$ (defined over Σ_{IL_j}). The high-level subsystem and the jth low-level subsystem are then $\mathbf{G}_H := \mathbf{G}_H^p \| \mathbf{S}_H$ and $\mathbf{G}_{L_j} := \mathbf{G}_{L_j}^p \| \mathbf{S}_{L_j}$, respectively.

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

$$\begin{aligned}
\mathbf{Plant} &:= \mathbf{G}_H^p \left\| \mathbf{G}_{L_1}^p \right\| \ldots \| \mathbf{G}_{L_n}^p \\
\mathbf{Sup} &:= \mathbf{S}_H \| \mathbf{S}_{L_1} \| \ldots \| \mathbf{S}_{L_n} \| \mathbf{G}_{I_1} \| \ldots \| \mathbf{G}_{I_n} \\
\mathcal{H}^p &:= P_{IH}^{-1} L(\mathbf{G}_H^p), \quad \mathcal{S}_H := P_{IH}^{-1} L(\mathbf{S}_H), \subseteq \Sigma^* \\
\mathcal{L}_j^p &:= P_{IL_j}^{-1} L\left(\mathbf{G}_{L_j}^p\right), \quad \mathcal{S}_{L_j} := P_{IL_j}^{-1} L\left(\mathbf{S}_{L_j}\right), \subseteq \Sigma^*.
\end{aligned}$$

For the controllability requirements at each level, we adopt the standard partition $\Sigma = \Sigma_u \dot{\cup} \Sigma_c$, splitting our alphabet into uncontrollable and controllable events. Note that this partition may, in general, be independent of the partition (1).

Definition 5: The nth degree $(n \geq 1)$ parallel interface system composed of DES $\mathbf{G}_H^p, \mathbf{G}_{L_1}^p, \ldots, \mathbf{G}_{L_n}^p$, $\mathbf{S}_H, \mathbf{S}_{L_1}, \ldots, \mathbf{S}_{L_n}$, $\mathbf{G}_{I_1}, \ldots, \mathbf{G}_{I_n}$, is level-wise controllable with respect to the alphabet partition given by (1), if for all $j \in \{1, \ldots, n\}$ the following conditions hold.

1) The alphabet of $\mathbf{G}_H^p$ and $\mathbf{S}_H$ is Σ_{IH}, the alphabet of $\mathbf{G}_{L_j}^p$ and $\mathbf{S}_{L_j}$ is Σ_{IL_j}, and the alphabet of $\mathbf{G}_{I_j}$ is Σ_{I_j}.
2) $(\forall s \in \mathcal{L}_j^p \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j)$
$$\mathrm{Elig}_{\mathcal{L}_j^p}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s).$$
3) $(\forall s \in \mathcal{H}^p \cap \mathcal{I} \cap \mathcal{S}_H)$
$$\mathrm{Elig}_{\mathcal{H}^p \cap \mathcal{I}}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\mathcal{S}_H}(s),$$
where $\mathcal{I} := \bigcap_{i=1,\ldots,n} \mathcal{I}_i$.

The above definition states that the system is level-wise controllable if, for the given distributed supervisor, the high-level supervisor is controllable for the high-level plant combined with all of the interfaces (by 3) and that each low-level supervisor synchronized with the subsystem's interface is controllable for the subsystem's low-level plant (by 2). Point 1 is an information hiding statement. By restricting the supervisors to the indicated event sets, we allow them to only view and disable events for their specific component.

In the above definition, we treated interfaces as supervisors in point 2, and plants in point 3. This discrepancy is a result of our hierarchy. As the high-level is concerned with global behavior (behavior that affects more than one low-level), it sees request and answer events as abstract actions performed by the low-level, but is not concerned with the details of how these events are implemented. As a low-level's job is to implement the commands (request events) given it by the high-level, it thus, has sufficient details to verify that its interface, working in conjunction with the low-level supervisor, is indeed controllable. As we have verified that the interfaces are controllable in point 2, we do not need to verify them again in point 3. By treating the interfaces as plants at the high-level, we can allow the high-level supervisor to be less restrictive, as the high-level plant alone does not typically have enough information about when interface events are eligible to occur as the interfaces themselves. Removing the interfaces from point 3 would also be too restrictive as this could not only make the high-level supervisor uncontrollable for the high-level plant, but the the supremal controllable sublanguage (for the high-level) would likely be the empty set!

We now present a sufficient condition for controllability of parallel interface systems. At first glance, the controllability definition used below might seem slightly different than the one given in Section I, but this can be easily reconciled by noting that for *Theorem 2*, $\Sigma_1 = \Sigma_S = \Sigma$.

Theorem 2: If the nth degree $(n \geq 1)$ parallel interface system composed of plant components $\mathbf{G}_H^p, \mathbf{G}_{L_1}^p, \ldots, \mathbf{G}_{L_n}^p$, supervisors $\mathbf{S}_H, \mathbf{S}_{L_1}, \ldots, \mathbf{S}_{L_n}$, and interfaces $\mathbf{G}_{I_1}, \ldots, \mathbf{G}_{I_n}$, is level-wise controllable with respect to the alphabet partition given by (1), 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: See proof in [30]. ∎

E. Verifying Properties

To aid in investigating HISC, we have developed software routines to verify that a system satisfies the level-wise nonblocking, interface consistent, and level-wise controllable conditions. These routines were developed by Leduc during his collaboration with Siemens Corporate Research and are a part of an experimental software tool. Currently, an open source implementation of these routines is being developed, but has not yet been released.

Examining the conditions to be verified, one observes that most of them are either very straightforward (i.e., verifying two sets are disjoint) or can be verified using existing supervisory control algorithms after suitable definitions have been made. Points 3 and 4 of the interface consistency definition can be verified using standard controllability algorithms such as TCTs condat function [5]. For example, in the case of Point 4, we simply define $\mathbf{ThePlant} = \mathbf{G}_{I_j}$, $\mathbf{TheSpec} = \mathbf{G}_{L_j}$, $\Sigma_u = \Sigma_{R_j}$, and $\Sigma_c = \Sigma - \Sigma_{R_j}$).

The exceptions are Points 5 and 6 of the interface consistency definition. They both require new algorithms, which we presented in [27]. Further details of the algorithms, including discussion of counter example generation when the conditions fail, can be found in [27].

An important topic of current research is how to synthesize local controls that cause an interface inconsistent system to become consistent or produce a controllable nonblocking sublanguage of a specification when these conditions fail. This is currently the topic of the M.A.Sc. thesis [33].

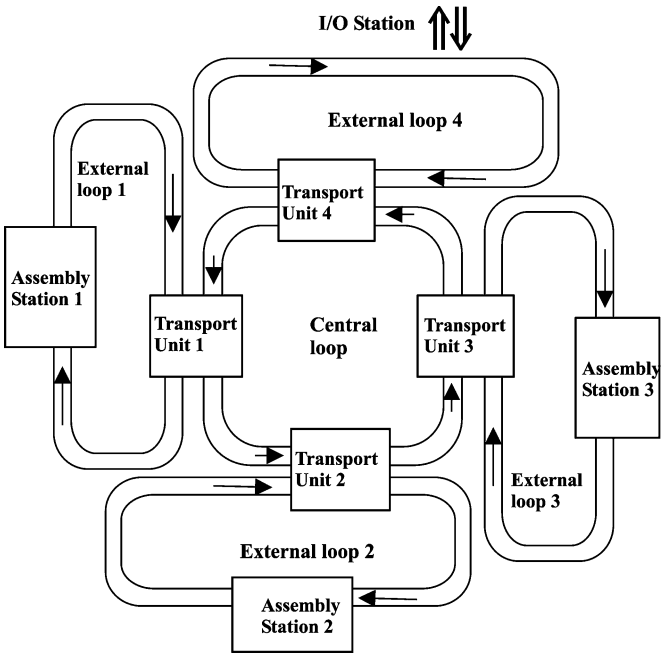


Fig. 6. AIP.

III. Overview of the AIP

To demonstrate the utility of our method, we apply it to a large manufacturing system, the AIP as described in [34] and [35]. The AIP, shown in Fig. 6, is an automated manufacturing system consisting of a central loop (CL) and four external loops (EL), three assembly stations (AS), an I/O station, and four inter-loop transfer units (TU). The I/O station is where the pallets enter and leave the system. Pallets entering the system can be of type 1 or of type 2, chosen at random.

A. Assembly Stations

The assembly stations are shown in Fig. 7. Each consists of a robot to perform assembly tasks, an extractor to transfer the pallet from the conveyor loop to the robot, sensors to determine the location of the extractor, and a raising platform to present the pallet to the robot. The station also contains a pallet sensor to detect a pallet at the pallet gate, the pallet stop, and a sensor to detect when a pallet has left the station. Finally, the assembly station contains a read/write (R/W) device to read and write to the pallet's electronic label. The pallet label contains information about the pallet type, error status, and assembly status (which tasks have been performed).

Whereas the assembly stations contain the same basic components, they differ with respect to functionality. Station 1 is capable of performing two separate tasks denoted task1A and task1B, while station 2 can perform tasks task2A and task2B. Station 3 can perform all four of these tasks as well as functioning as a repair station allowing an operator to repair a damaged pallet. The assembly stations also differ with respect to reliability. Stations 1 and 2 can break down and must be repaired, while station 3 is of higher quality and is assumed never to break down. Station 3 is used as a substitute for the other stations when they are down.

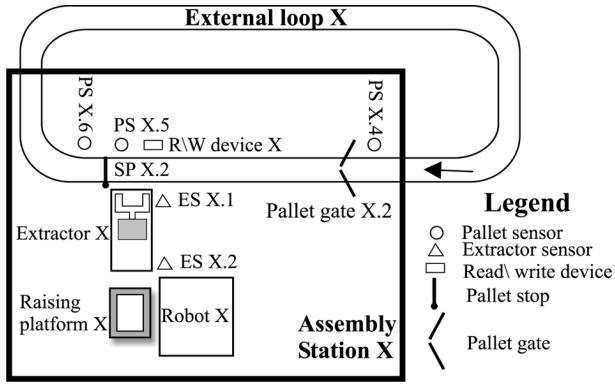


Fig. 7. Assembly station of external loop $X = 1, 2, 3$.

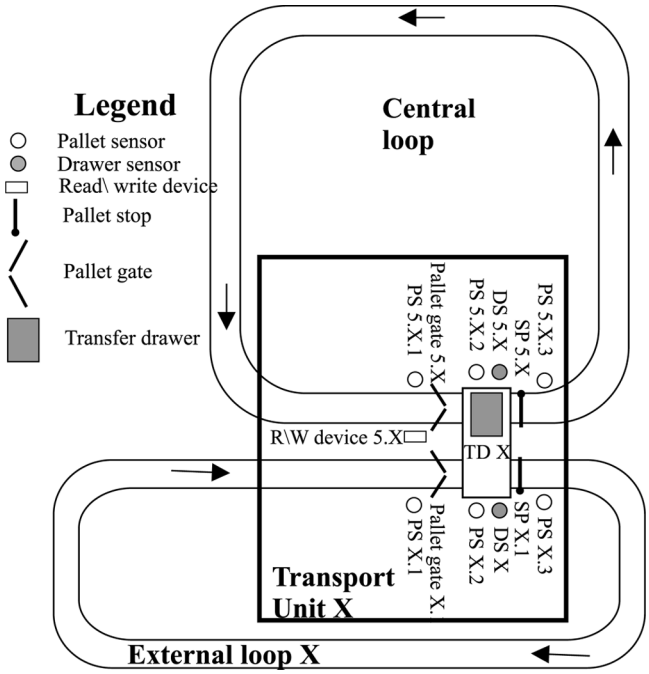


Fig. 8. Transport unit for external loop $X = 1, 2, 3, 4$.

B. Transport Units

The structure of the four identical transport units is shown in Fig. 8. The transport units are used to transfer pallets between the central loop and the external loops. Each one consists of a transport drawer which physically conveys the pallet between the two loops, plus sensors to determine the drawer's location. At each loop, the unit contains a pallet gate and a pallet stop, to control access to the unit from the given loop. The unit also contains multiple pallet sensors to detect when a pallet is at a gate, drawer, or has left the unit. Also, each unit contains a R/W device located before the central loop gate.

C. Control Specifications

For this example, we adopt the control specifications and assumptions used in [34] and [35] and restated as points (1)–(6) below. To this we add Specification 7 to make the assembly stations more interesting and complicated.

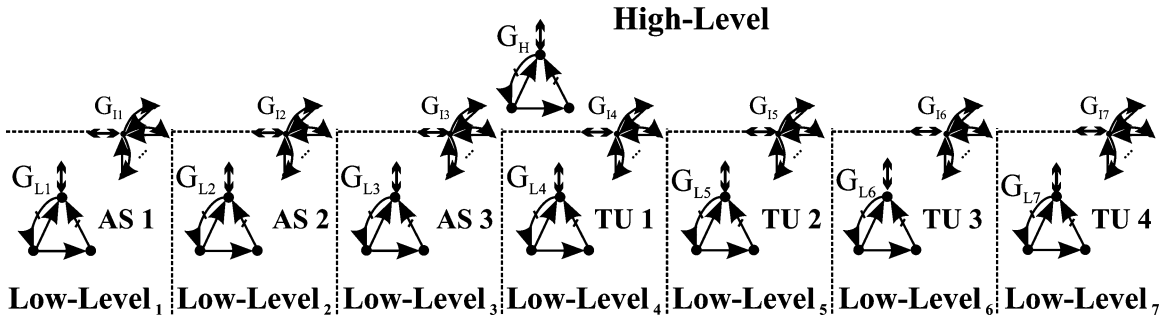


Fig. 9. Structure of parallel system.

Assumptions: We assume that: 1) the system is initially empty; 2) two types of pallets are introduced to the system, subjected to assembly operations, and then leave; and 3) pallets enter the system following the order: type 1, type 2, type 1,....

Specifications:

1) **Routing**: Pallets follow a certain route based on their type. A type 1 pallet must go first to AS1, then AS2 before leaving the system. Type 2 pallets go first to AS2, then AS1 before leaving the system. A pallet is not allowed to leave the system until all four assembly tasks have been successfully performed on it.
2) **Maximum capacity of external loops 1 and 2**: The maximum allowed number of pallets in each loop at a given time is one.
3) **Ordering of pallet exit from system**: The pallets must exit the system in the following order: type 1, type 2, type 1,...
4) **Assembly errors**: When a robot makes an assembly error, the pallet is marked damaged and routed to AS3 for maintenance. After maintenance, the pallet is returned to the original assembly station to undergo the assembly operation again.
5) **Assembly station breakdown**: The robots of external loops 1 and 2 are susceptible to breakdowns. When a station is down, pallets are routed to assembly station 3 which is capable of performing all tasks of the other two stations. When the failed station is repaired, all pallets not already in external loop 3 are rerouted to the original station.
6) **Maximum capacity of assembly stations**: To avoid collisions, only one pallet is allowed in a given station at a time.
7) **Assembly task ordering**: Assembly tasks are performed in a different order for pallets of different types. For pallets of type 1, task1A is performed before task1B, and task2A is performed before task2B. For pallets of type 2, task1B is performed before task1A, and task2B is performed before task2A.

IV. System Structure

To cast the AIP into a parallel interface system, we break the system down into a high-level (models how the low levels interact with each other), and seven low levels corresponding to the three assembly stations and four transport units, as shown in Fig. 9. We describe each of the subsystems (components) in the following sections. As this example contains more than 174 DES, we are not able to describe the design in complete detail, but refer the reader to [28] for a complete description.

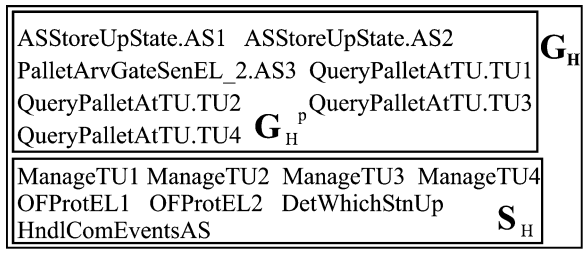


Fig. 10. AIP high-level.

The models and supervisors developed for this example are based on the automata presented in [34] and [35]. We have altered them to fit our setting, and extended them to fill in the missing details of several events that were defined, in order to simplify the model and reduce complexity, as "macro events." By adding in these missing details, we substantially increased the complexity of the example, which is immediately apparent, by noting that the estimated worst case state space size of the original plant in [34] is 2×10^{24}, while our estimated worst case state space size is 2×10^{43} (see Section V for details).

In this paper, all supervisors were designed for their level as modular supervisors. The supervisors were designed by hand to meet the given specifications and then verified that they satisfy their share of the interface, controllability, and nonblocking properties. If a component fails to satisfy its share of these properties, it is modified until it does satisfy them.

In the following diagrams, uncontrollable events are shown in italics; all other events are controllable. Initial states can be recognized by a thick outline and marker states are filled.

A. High-Level Subsystem

The high-level subsystem keeps track of the breakdown status of assembly stations 1 and 2 and enforces the maximum capacity of external loops 1 and 2. This component controls the external operation of all transport units and assembly stations, while tracking the pallet's progress around the manufacturing system. The plant components of the high-level, primarily provide a mechanism to determine information about the system, such as if a given assembly station is down. The majority of the high-level's behavior is encapsulated in its supervisors. The high-level $\mathbf{G}_H$ consists of the synchronous product of the 15

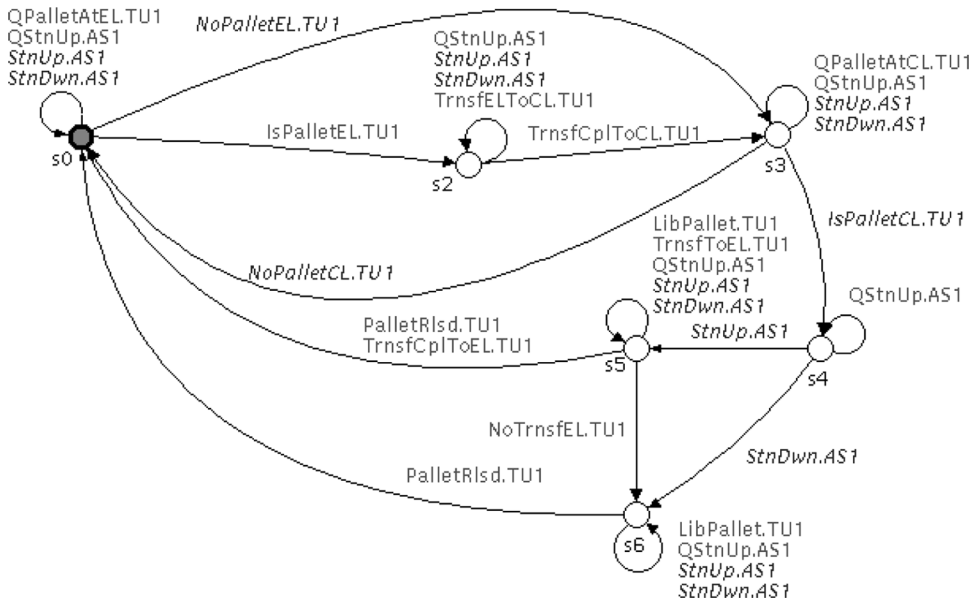


Fig. 11. Supervisor ManageTU1.

DES listed in Fig. 10. It is further subdivided into a plant component $\mathbf{G}_H^p$, and a supervisor component $\mathbf{S}_H$ as indicated in the diagram.

As an example of the high-level subsystem's behavior, we discuss supervisor ManageTU1, shown in Fig. 11. This supervisor controls the transfer of pallets between the central loop and external loop 1. It permits pallets on the central loop to pass through transport unit 1 (to be liberated) without being transferred to the external loop. Pallets are liberated if EL1 is at maximum capacity, AS1 is down, or TU1 determines that the pallet is not to be transferred.

B. Low-Level Subsystems AS1 and AS2

We now describe the low-level subsystems that represent assembly stations 1 and 2. As they are identical, we will describe them collectively as component k, where $k = \text{AS1, AS2}$. Component k provides the functionality specified in its interface, shown in Fig. 12. The assembly station accepts the pallet at its gate, and presents it to the robot for assembly. It then releases the pallet and reports on the success of the assembly operation. If the robot breaks down, this is reported through the interface and the pallet is released. Component k then waits for a repair command to return the robot to operation. Fig. 12 shows how the breakdown status of component k is reflected in its interface, and how only the appropriate request event is possible at a given marked state. Component k (low-level $w = 1, 2$) contains the 17 DES listed in Fig. 13. The diagram gives the definition of component k's subsystem $\mathbf{G}_{L_w}$, plant component $\mathbf{G}_{L_w}^p$, and supervisor component $\mathbf{S}_{L_w}$. Again, they are the synchronous product of the indicated automata.

Supervisor HndlPallet.AS1, shown in Fig. 16, provides an example of low-level subsystem AS1's behavior. HndlPallet.AS1 handles the task of processing a pallet once it reaches the extractor. It reads the pallet's label, presents the pallet to the robot, and has the robot perform the appropriate tasks on the pallet.

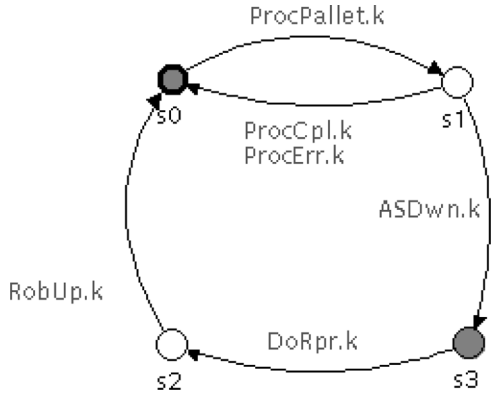


Fig. 12. Interface to low-level $k = \text{AS1, AS2}$.

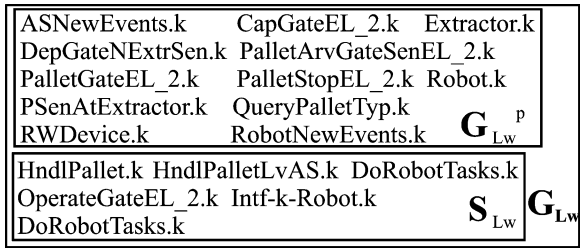


Fig. 13. AIP low-level $k = \text{AS1, AS2}$.

The supervisor then allows the pallet to leave the assembly station and reports on the success of the processing operation by updating the pallet's label and notifies the high-level subsystem through the interface.

C. Low-Level Subsystem AS3

Component AS3 provides the functionality specified in its interface, shown in Fig. 14. This subsystem describes the behavior of assembly station 3, which is very similar to stations 1 and 2.

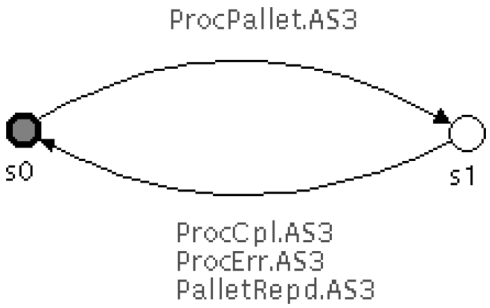


Fig. 14. Interface to low-level AS3.

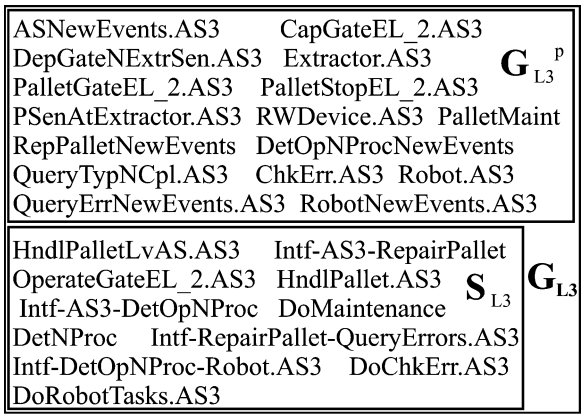


Fig. 15. AIP low-level AS3.

The main differences are that station 3 can repair damaged pallets, is assumed not to break down, and can substitute for either AS1 or AS2 when they are down. Component AS3 (low-level 3) contains the 27 DES listed in Fig. 15. The diagram gives the definition of Component AS3's subsystem $\mathbf{G}_{L_3}$, plant component $\mathbf{G}^p_{L_3}$, and supervisor component $\mathbf{S}_{L_3}$.

One of the difficulties in designing an HISC system is to ensure a clear line of dependency (when required) between events in the high-level, interfaces, and low levels. AS3 provides an excellent example. Taking $X = 3$ in Fig. 7, we see that a pallet must reach pallet sensor PS 3.4 before it can leave pallet gate 3.2. In the modeling used in [34] and [35], this was captured in a single DES. However, the "pallet has arrived at the gate" information is needed at the high-level to determine when to activate the assembly station, while the "pallet leaving the gate" information is needed internally to express the relationship that a pallet cannot arrive at pallet stop SP 3.2 (located at the extractor) until it has left the gate. The latter relationship is captured by subplant DepGateNExtraSen.AS3, shown in Fig. 17 ($j = AS3$). The dependency between these two actions can be established by making request event Proc.Pallet.AS3 dependent on the pallet arriving at the gate (subplant PalletArvGateSenEL_2.AS3 in Fig. 18) and the pallet leaving the gate dependent on event *Proc.Pallet.AS3* (subplant *CapGateEL_2.AS3* in Fig. 19). This preserves the dependency between the pallet arriving at the gate and the pallet leaving the gate, while at the same time ensuring that the internal events of AS3 do not have direct dependencies on external events.

D. Low-Level Subsystems TU1 and TU2

We now describe the low-level subsystems that represent transport units 1 and 2. As they are identical, we will describe them collectively as component r, where $r = \mathrm{TU1}, \mathrm{TU2}$. We also define the companion index $X = 1, 2$, which takes its values relative to r (e.g., $X = 1$ when $r = \mathrm{TU1}$). For diagrams in this section, we set index $i = r$. Component r (low-level $v = 4, 5$) contains the 25 DES listed in Fig. 20. The diagram gives the definition of component r's subsystem $\mathbf{G}_{L_v}$, plant component $\mathbf{G}^p_{L_v}$, and supervisor component $\mathbf{S}_{L_v}$.

Component r provides the functionality specified in its interface, shown in Fig. 21. The transport units are used to transfer pallets between the central loop, and the external loops (i.e., TU1 transfers pallets between CL and EL1). Component r has two entry points for pallets: the central loop gate and the external loop gate. If a pallet is at the EL gate, subsystem r transfers the pallet to the central loop. If a pallet is at the CL gate, component r can be requested to liberate the pallet (allow it to pass through and continue on the CL), or to transfer the pallet to the EL. When requested to transfer a pallet to the EL, component r will only transfer the pallet if the pallet is undamaged and if the next assembly task required by the pallet is performed by the external loop's assembly station.

E. Low-Level Subsystem TU3

Low-level TU3 provides the functionality specified in its interface, shown in Fig. 22. This component describes the behavior of transport unit 3, which is very similar to TU1 and TU2. TU3 differs in how it decides if a pallet should be transferred from the central loop to external loop 3. First, all damaged pallets are to be transferred to EL3 for maintenance. Second, if an assembly station is down and it performs the next pending task for the pallet, then the pallet is to be transferred. As TU3 must know the breakdown status of assembly stations 1 and 2, this information is passed in explicitly as differently labeled request events. Component TU3 (low-level 6) contains the 29 DES listed in Fig. 24. The diagram gives the definition of Component TU3's subsystem $\mathbf{G}_{L_6}$, plant component $\mathbf{G}^p_{L_6}$, and supervisor component $\mathbf{S}_{L_6}$.

TU3 differs from TU1 and TU2 primarily in its logic to transfer pallets from the central loop to its external loop. This is handled by supervisor HndlTrnsfToEL.TU3, shown in Fig. 23. HndlTrnsfToEL.TU3 will only transfer pallets to EL3 if they are damaged, or if the next assembly operation required by the pallet is performed by an assembly station that is down.

To determine if a substitute assembly operation is required, HndlTrnsfToEL.TU3 makes use of supervisor HndlSelCheck.TU3, shown in Fig. 26. HndlSelCheck.TU3 maps the request events TrnsfToEL3_Up, TrnsfToEL3_1D, TrnsfToEL3_2D, and TrnsfToEL3_BD, to the appropriate local command to check to see if substitution is required. These request events were encoded by the high-level with the breakdown status of assembly stations 1 and 2, by only allowing the event with the correct meaning to occur. This is analogous to passing a parameter to a function in a software program.

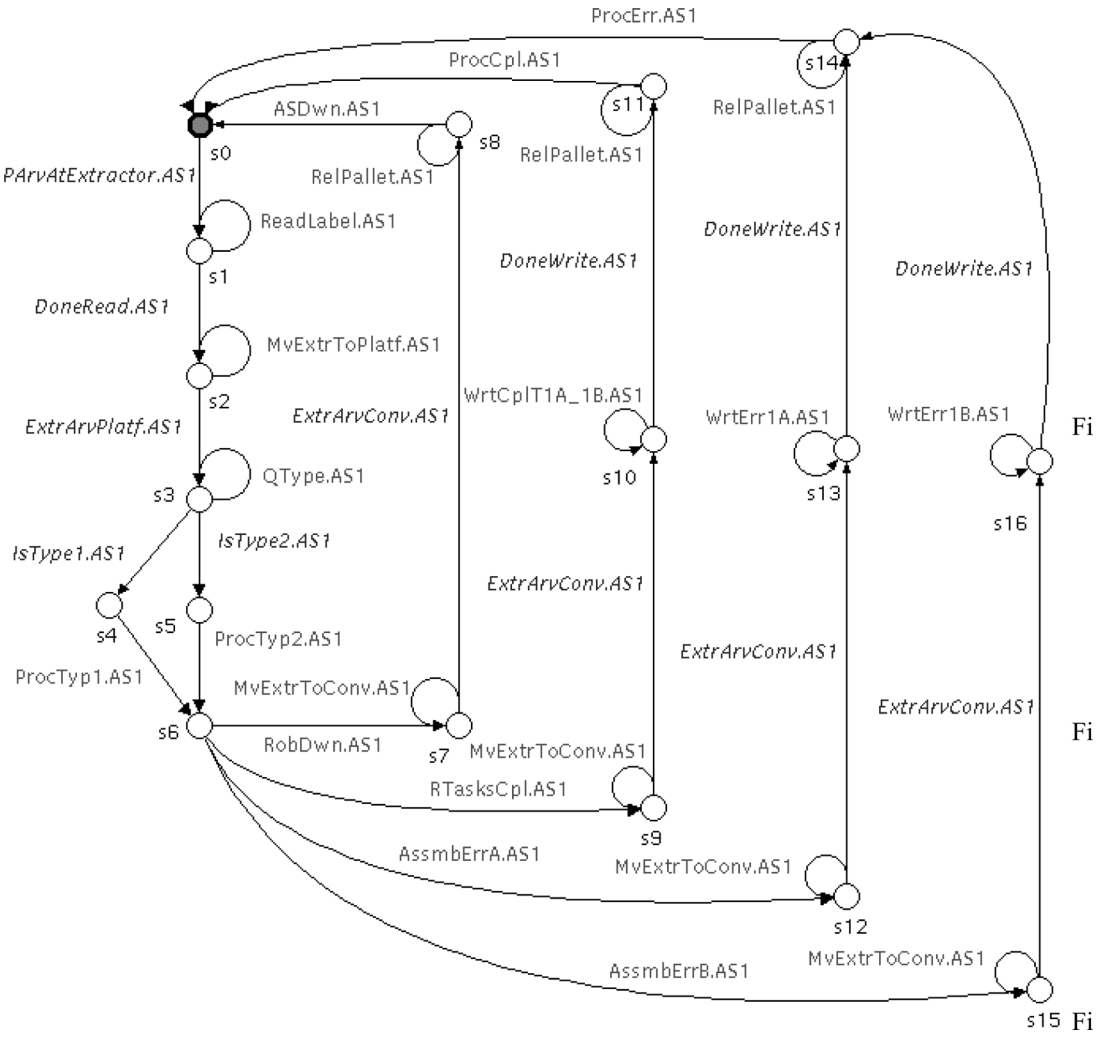


Fig. 16. Supervisor HndlPallet.AS1.

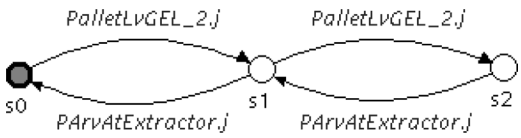


Fig. 17. DepGateNExtrSen.j.

Fig. 18. PalletArvGateSenEL_2.AS3.

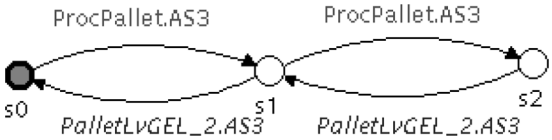


Fig. 19. CapGateEL_2.AS3.

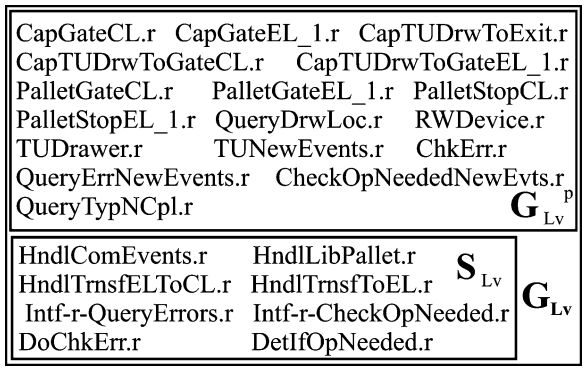


Fig. 20. AIP low-level $r = \mathrm{TU1}, \mathrm{TU2}$.

F. Low-Level Subsystem TU4

Low-level TU4 provides the functionality specified in its interface, shown in Fig. 21, and contains the 19 DES listed in Fig. 25. The diagram gives the definition of component TU4's subsystem $\mathbf{G}_{L_7}$, plant component $\mathbf{G}_{L_7}^{p}$, and supervisor component $\mathbf{S}_{L_7}$.

Component TU4 (low-level 7) describes the behavior of transport unit 4, which is very similar to TU1 and TU2. TU4

TABLE I
SIZE OF AIP SUBSYSTEM AUTOMATA MODELS

Subsystem	Standalone		$\|\|$ with $\mathbf{G}_{I_j}$		Size of $\mathbf{G}_{I_j}$	
	States	Transitions	States	Transitions	States	Transitions
$\mathbf{G}_H$	1,480,864	14,419,512	3,306,240	28,442,432	8,192	104,448
AS1	1,795	4,418	120	191	4	6
AS2	1,795	4,418	120	191	4	6
AS3	1,199	2,448	203	330	2	4
TU1	98	120	98	120	4	7
TU2	98	120	98	120	4	7
TU3	204	266	204	266	4	10
TU4	152	180	152	180	4	7

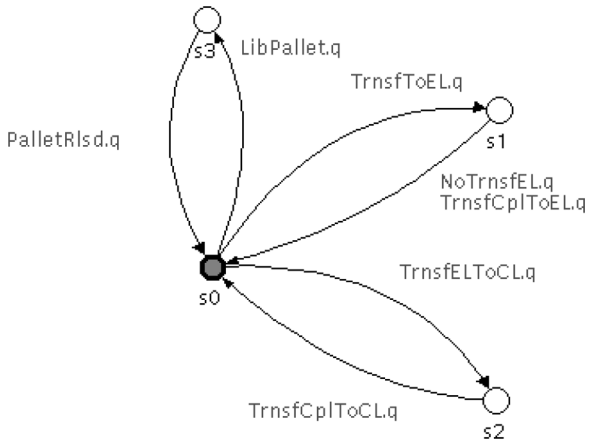


Fig. 21. Interface to low-level $q = \mathrm{TU1}$, TU2, TU4.

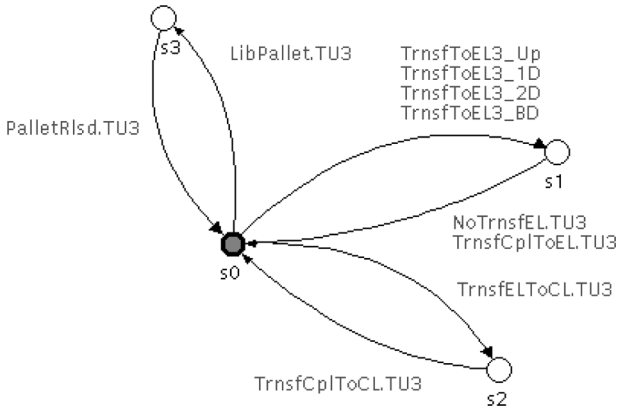


Fig. 22. Interface to low-level TU3.

differs in how it decides if a pallet should be transferred from the central loop to its external loop (EL4), which contains the I/O station. As pallets are required to leave the system in a particular order (i.e., type 1, type 2, type 1, ...), TU4 keeps track of the type of the last pallet to be transferred to EL4 and will only transfer the current pallet if it is of the type required by the sequence, and if all required assembly tasks have been successfully performed on the pallet.

V. DISCUSSION OF RESULTS

Applying our research tool to the system, we find that it is level-wise nonblocking, level-wise controllable, and interface consistent. By Theorems 1 and 2, we can conclude that the flat system is nonblocking and the flat system's supervisor is controllable for the flat plant.

This example contains more than 174 DES in total, with an estimated closed-loop state space of 2.9×10^{21}. This estimate was calculated by determining the closed-loop state space of the high-level, and each low-level and then multiplying these together to create a worst case state estimate. Similarly, we estimated the state space size of the open loop plant model to be 2×10^{43}. For both estimates, it is quite likely that the actual system will be considerably smaller. The computation ran for 25 min, using 760 MB of memory. The machine used was a 750-MHz Athlon system, with 512 MB of RAM, 2 GB of swap, running Redhat Linux 6.2. A standard nonblocking verification was also attempted on the monolithic (flat) system, but it quickly failed due to lack of memory.

Table I shows the sizes of the various subsystem automata used in the AIP calculations. First, the size of the state space of each component without being synchronized with their respective interfaces (Standalone) is given and then, state space size when synchronized with their interface DES ($\mathbf{G}_H$ is synchronized with all seven interfaces). The last two columns give the size of the interfaces for the high-level and each low-level. Letting N_H denote the size of the state space of $\mathbf{G}_H$, while N_I and N_L are upper bounds for the state space size of $\mathbf{G}_{I_j}$ and $\mathbf{G}_{L_j}$ $(j = 1, \ldots, n)$, respectively, we find that the limiting factor for a monolithic algorithm[2] would be $N_H N_L^n$ and similarly $N_H N_I^n$ for the HISC method [27]. If we substitute actual data from Table I, we get $N_L^n = (120)^2(203)(98)^2(204)(152) = 8.71 \times 10^{14}$ and $N_I^n = (4)^2(2)(4)^4 = 8192$. This is a potential savings of 11 orders of magnitude! In fact, instead of multiplying $N_H = 1\,480\,864$ by a factor of 8192, adding the interfaces only doubles the state space of the high-level. For low-level AS1, synchronizing with its interface actually causes the state space to decrease from 1795 to 120 states, an order of magnitude reduction.

We note that the prototype tool used for these calculations did not make use of IDD/BDD [36] and symbolic techniques such as those used in [37] and [38]. We conjecture that using HISC methods with tools utilizing symbolic techniques should allow the method to scale up to considerably larger systems as has been the case with the application of symbolic techniques to monolithic supervisory control calculations.

A. Localization of Changes

When modeling and designing supervisors for large systems like the AIP, one will be working with a large number of automata. As a result, an important concern is the effort involved

[2]A monolithic algorithm is performed on the composite system, which is based on the cartesian product of subsystems.

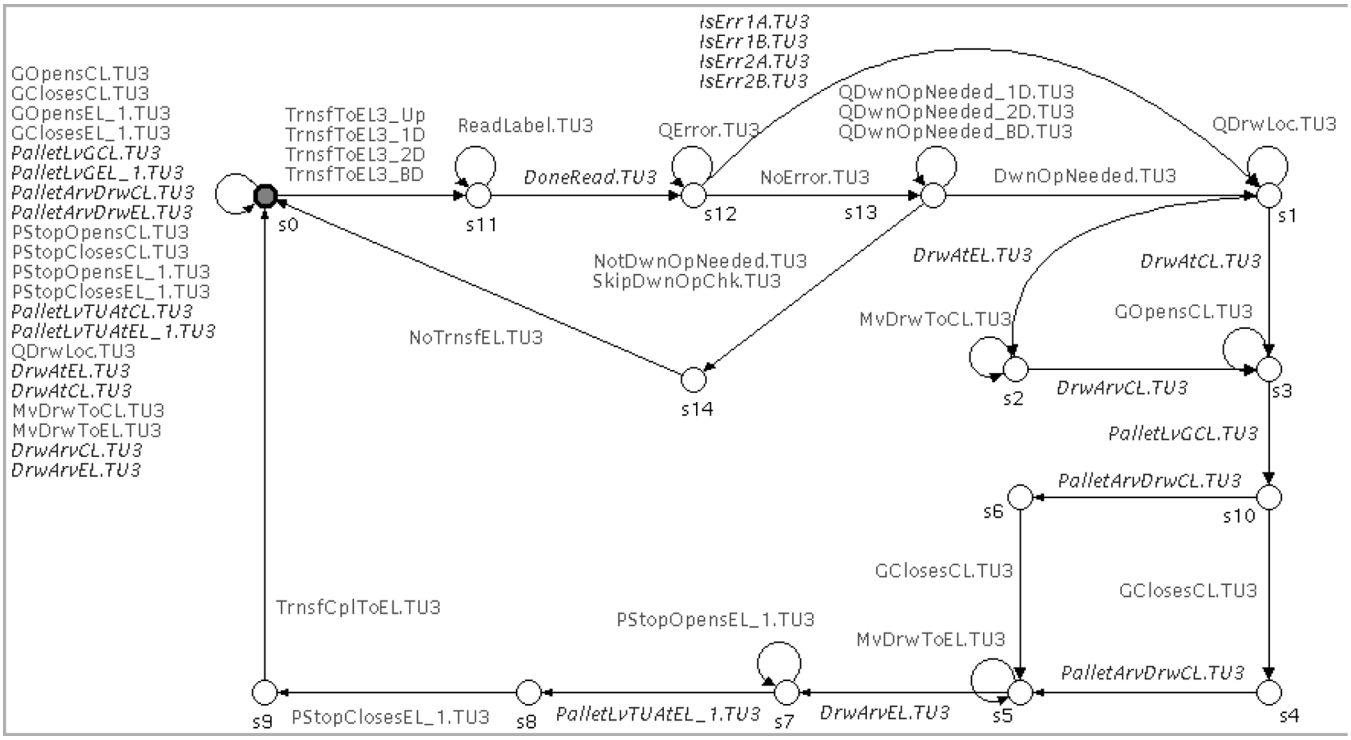


Fig. 23. Supervisor HndlTrnsfToEL.TU3.

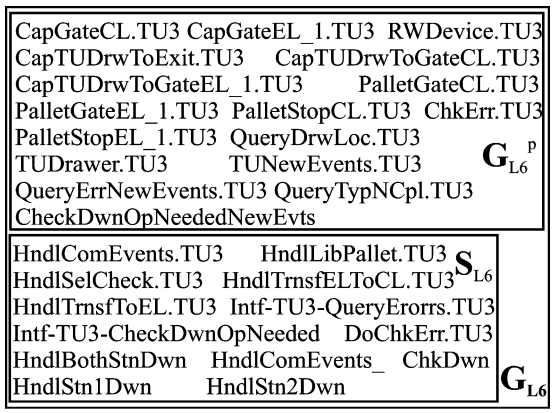


Fig. 24. AIP low-level TU3.

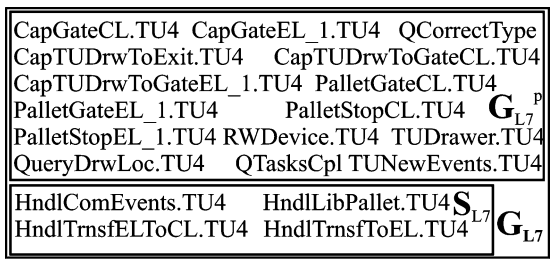


Fig. 25. AIP low-level TU4.

in making changes to your existing design. If the plant is modified or the control specifications are changed, it is desirable to restrict the effect of these changes to as few DES as possible.

In a flat system where supervisors can see and disable any event, a small change to one part of the system could require changes to a large number of DES throughout the system. There

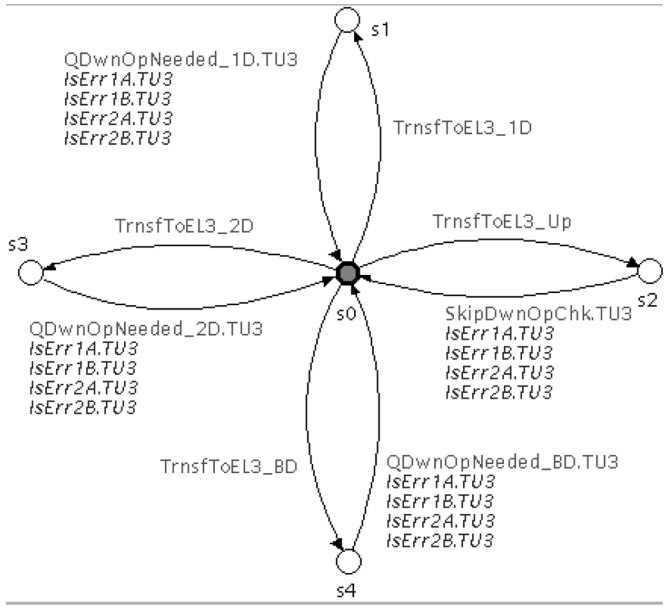


Fig. 26. HndlSelCheck.TU3.

is nothing to guarantee localization of the changes. We would also have to reverify the entire system after the changes.

For HISC, the information hiding approach creates a compartmentalization effect. As long as the interface for a low-level is unchanged, modifications to that low-level will only affect the plant and supervisor DES of that low-level. All other low levels will be unchanged and we will only have to reverify the HISC conditions for the modified low-level. If the interface for a low-level is changed as well, this will at most cause changes to the high-level. All other low levels will be unaffected. Finally, if the high-level is modified, no interfaces or low levels

would need to be changed; we would only need to reverify the high-level's HISC conditions.

The HISC conditions and structure can also be used to create reusable libraries. Once an interface is defined, the low-level can be modeled, supervisors designed, and HISC conditions verified completely separately from the rest of the system. This low-level can then be used in any system without having to recheck the low-level's HISC conditions. This would allow a company that sells a manufacturing machine, for instance, to specify an interface, and then model, design, and verify the low-level for the given machine. The company can then provide the interface and low-level as a preverified package, allowing a customer to add the machine to their system, and then just design a high-level to operate the machine's interface. This could represent a considerable saving of effort for a customer.

B. Related Work

The most significant feature which distinguishes this paper from some current work along similar lines (e.g., [22]) is the results on nonblocking, although Endsley *et al.* later extended their work to include a form of deadlock detection in [39]. Also, the focus of this paper is on how the HISC theory can be applied to provide safe, nonblocking supervision of a nontrivial industrial system.

One way to improve the scalability of modular and decentralized schemes is to exploit the existing architecture of the system. In [40], 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 [41] 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 [3] to obtain a reduction in complexity for the nonconflicting check of modular control. Similarly the standard controllability definition has been refined and localized in [42] to check on a per subplant basis only those uncontrollable events that can occur locally.

The scale of the AIP system is much larger than that of the illustrative example system given in [43]. There, the open loop system could be represented by a 14 place petri net with no more than one token per place in any state. Thus, the open-loop state space size has upper bound of $2^{14} = 16\,384$. The controller has three places which have the restriction that the sum of tokens for the three places is always 1 (i.e., there are three states for the controller) giving an upper bound on the closed loop state space size of $3{\times}2^{14} = 49\,152$. The most generous estimate of the state space size is $2^{17} = 131\,072$ for arbitrary control policies on the three control places. Application of the methods of [43] to an example of the scale of the AIP system would permit a more direct comparison of the scalability of the method. We note that while [43] also relied upon identification of specific structural properties of the system, "the class of flexible manufacturing systems called MRF1 (multiple reentrant flow lines with no self-loops, with jobs that require only one resource, with no two consequent jobs using the same resource, with no choice jobs and no assembly jobs and [sic] with shared resources)," they consider is not comparable to the class of systems to which HISC can be applied. For instance, the AIP example used to illustrate HISC violates the "no machine failures," and "no self-loops" conditions, while the MRF1 class of system does not impose that same architectural restriction upon the plant subsystems.

In [38], Ma *et al.* extended an earlier version (from [28]) of our AIP example by changing Specification 2 of Section III-C to allow up to ten pallets in each of the two external loops. Admittedly, their version is larger but it is important to note that they are using binary decision diagrams, which represent the system as compact predicates. It has been shown that symbolic techniques alone ([37], [44]) can allow significant gains in the size of systems that can be handled, irrespective of hierarchical methods, provided an appropriate variable ordering can be found to exploit the structure of the system. In our HISC method, we were able to handle the AIP example using only automata based algorithms, that extensionally represent the states and transitions of the DES. Use of symbolic techniques should allow the method to scale up to considerably larger systems.

VI. Conclusion

Hierarchical interface-based supervisory control offers an effective means to model systems with a natural master–slave structure. Using multiple $(n \geq 1)$ low-level subsystems allows the subsystems to be independently modeled and verified, while still allowing a high degree of concurrent operation. Because each of the interface conditions can be verified using a single subsystem and its interface(s), the complete system model never needs to be stored in memory or traversed, offering potentially significant savings in computational resources. This allowed us to be able to quickly verify a large system that was previously far beyond our means. An added benefit is that the independent subsystems can be more easily understood with a higher degree of reusability. Any individual subsystem can be replaced without requiring the other subsystems to be altered or reverified.

These benefits were illustrated by a large example (more than 174 DES with a plant model that has an estimated worst case open loop state space size of 2×10^{43}) based on the automated manufacturing system of the AIP. The example demonstrates how the HISC method can be applied to interesting systems of realistic complexity, even though symbolic techniques have not yet been incorporated into the approach.

A. Limitations and Future Research

In the HISC approach, the limiting factor is the state space size of the synchronous product of the high-level subsystem and the interfaces to all of the low-level subsystems. When the interfaces can be designed to have smaller state spaces than the low-level subsystems (true for the AIP example), the state space of the high-level synchronized with the interfaces will be considerably smaller than the state space of the flat system model. However, in the worst case the high-level's state space can still grow exponentially in the number of components. To address this problem, future research will focus on extending the method to a multilevel hierarchy.

Currently, we only provide a method to verify if a system is globally nonblocking and controllable but we do not provide

a means to synthesize a maximally permissive supervisor. We intend to address this problem by developing a synthesis method that will respect the interface conditions and, thus, be able to take advantage of the benefits of our HISC method. The result should be maximally permissive for the system as constrained by the interfaces, but in general, will be more restrictive than the maximally permissive behavior without interface constraints.

REFERENCES

[1] N. Alsop, "Formal techniques for the procedural control of industrial processes," Ph.D. dissertation, Dept. Chemical Eng. Chemical Technol., Imperial College Sci., Technol., Med., London, U. K., 1996.

[2] 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.

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

[4] 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.

[5] W. M. Wonham, "Supervisory control of discrete-event systems," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 2004.

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

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

[8] 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.

[9] 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.

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

[11] 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.

[12] Y. Li and W. Wonham, "Control of vector discrete-event systems: I—The base model," *IEEE Trans. Autom. Control*, vol. 38, no. 8, pp. 1214–1227, Aug. 1993.

[13] J. O. Moody and P. J. Antsaklis, *Supervisory Control of Discrete Event Systems Using Petri Nets*. Norwell, MA: Kluwer, 1998.

[14] M. Zhou and F. DiCesare, *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Norwell, MA: Kluwer, 1993.

[15] P. Caines and Y. Wei, "The hierarchical lattices of a finite machine," *Syst. Contr. Lett.*, vol. 25, no. 7, pp. 257–263, Jul. 1995.

[16] H. Chen and H.-M. Hanisch, "Model aggregation for hierarchical control synthesis of discrete event systems," in *Proc. 39th Conf. Dec. Contr.*, 2000, pp. 418–423.

[17] 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.

[18] K. Wong, "Discrete-event control architecture: An algebraic approach," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 1994.

[19] W. Wu, H. Su, J. Chu, and H. Zhai, "Hierarchical control of DES based on colored petri nets," in *Proc. IEEE Syst., Man, Cybern.*, 2001, pp. 1571–1576.

[20] 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.

[21] 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.

[22] E. W. Endsley, M. R. Lucas, and D. M. Tilbury. (2000, Oct.). *Modular design and verification of logic control for reconfigurable machining systems.* [Online] Available: http://www-personal.engin.umich.edu/\verb+ +tilbury/papers.html

[23] P. Gohari-Moghadam, "A linguistic framework for controlled hierarchical DES," M.A.Sc. thesis, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 1998.

[24] H. Liu, J. Park, and R. Miller, "On Hybrid synthesis for hierarchical structured petri nets," Dept. Comput. Sci., Univ. Maryland, College Park, MD, Rep. CS-TR-3628, 1998.

[25] B. Wang, "Top-down design for rw supervisory control theory," M.A.Sc. thesis, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 1995.

[26] R. Leduc, B. Brandin, M. Lawford, and W. M. Wonham, "Hierarchical interface-based supervisory control, Part I: Serial case," *IEEE Trans. Autom. Control*, vol. 50, no. 9, pp. 1322–1335, Sep. 2005.

[27] 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.

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

[29] 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.

[30] R. Leduc, M. Lawford, and P. Dai, "Hierarchical interface-based supervisory control of a flexible manufacturing system," Software Quality Res. Lab., Dept. Comput. Software, McMaster Univ., Hamilton, ON, Canada, Tech. Rep. 32, 2005.

[31] P. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete-event processes," *SIAM J. Contr. Optim.*, vol. 25, no. 1, pp. 206–230, Jan. 1987.

[32] W. M. Wonham and P. Ramadge, "On the supremal controllable sublanguage of a given language," *SIAM J. Contr. Optim.*, vol. 25, no. 3, pp. 637–659, May 1987.

[33] P. Dai, "Synthesis method for hierarchical interface-based supervisory control," M.A.Sc. thesis, Dept. Comput. Software, McMaster Univ., Hamilton, ON, Canada.

[34] B. Brandin and F. Charbonnier, "The supervisory control of the automated manufacturing system of the AIP," in *Proc. Rensselaer's Fourth Int. Conf. Comput. Integr. Manuf. Autom. Technol.*, 1994, pp. 319–324.

[35] F. Charbonnier, "Commande par Supervision des Systèmes à Événements Discrets: Application à un Site Expérimental L'Atelier Inter-Établissement de Productique," Lab. d'Automatique Grenoble, Grenoble, France, Tech. Rep., 1994.

[36] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, 1986.

[37] 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.

[38] C. Ma and W. M. Wonham, "Nonblocking Supervisory Control of state tree structures," in *Lecture Notes in Control and Information Sciences*. Berlin, Germany: Springer-Verlag, 2005, vol. 317.

[39] E. W. Endsley and D. M. Tilbury, "Modular verification of modular finite state machines," in *Proc. 43th Conf. Decision Contr.*, vol. 1, 2004, pp. 972–979.

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

[41] S. Abdelwahed, "Interacting discrete event systems: Modeling, verification and supervisory control," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 2002.

[42] K. Åkesson, H. Flordal, and M. Fabian, "Exploiting modularity for synthesis and verification of supervisors," in *Proc. IFAC World Congr. Autom. Control*, 2002, CDROM.

[43] S. Bogdan, F. L. Lewis, Z. Kovačić, A. Gürel, and M. Štajdohar, "An implementation of the matrix-based supervisory controller of flexible manufacturing systems," *IEEE Trans. Contr. Syst. Technol.*, vol. 10, no. 5, pp. 709–716, Sep. 2002.

[44] J. Burch, E. M. Clarke, and K. McMillan, "Symbolic model checking: 10^{20} states and beyond," *Inform. Computat.*, vol. 98, no. 2, pp. 142–170, Jun. 1992.

Ryan J. 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.

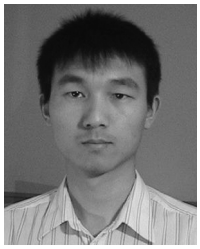
In 1997 and 1998, he was a Guest Scientist at Siemens Corporate Technology, Munich, Germany. In 2001, he joined McMaster University, Hamilton, ON, Canada, where he is currently an Assistant Professor in the Department of Computing and Software. His research interests include supervisory control of discrete-event 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. He is a licensed Professional Engineer in the province of Ontario, Canada.

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

From 1997 to 1998, he worked at Ontario Hydro, Toronto, ON, Canada, as a consultant on the Darlington Nuclear Generating Station Shutdown Systems Redesign Project, where he was a corecipient of an Ontario Hydro New Technology Award. Currently, he is an Associate Professor in the Department of Computing and Software at McMaster University, Hamiltion, ON, Canada, where he has helped to develop software engineering programs. His research interests include discrete event systems, formal methods for real-time systems, and computer-aided inspection of safety critical software.

Dr. Lawford is a Licensed Professional Engineer in the Province of Ontario, Canada.



Pengcheng Dai received the B.Eng. degree in computer science and engineering from Tianjin University, Tianjin, China, in 1998. He is currently pursuing the M.A.Sc. degree in software engineering at McMaster University, Hamilton, ON, Canada.

He worked for Motorola, Tianjin, China, as a software developer. He is currently a software consultant with Morgan Stanley, Mantra, QC, Canada. His research interests include the discrete-event system area, especially in supervisory control of DES, and creating software tools for modeling DES systems.