Hierarchical Interface-based Supervisory Control: Command-pair Interfaces

R.J. Leduc

Dept. of Computing and Software, McMaster University
email: leduc@mcmaster.ca

Version 1.1,
June 15, 2004

Abstract

In this paper we extend our previous results in which we presented a hierarchical method that decomposed a system into a *high level subsystem* which communicated with $n \geq 1$ parallel *low level subsystems* through separate interfaces. This method offered significant computational savings as the complete system model never needed to be constructed.

We introduce a new interface structure called *command-pair interfaces* which extends the previous approach (*star interfaces)* to now be able to represent state information about the *low levels.* We show that *star interfaces* are a special case, and extend the results for *star interfaces* to include the more general *command-pair interfaces.*

We illustrate the new approach by re-visiting a large manufacturing example that was designed using only *star interfaces.* Finally, we present a complexity analysis showing that the algorithm's time complexity is $\mathbf{O}(m^2)$, where $m = n + 1$ is the total number of subsystems.

1 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 statespace. Although many methods have been developed to deal with this problem (modular control [1, 17, 31, 46, 52, 56], decentralized control [5, 34, 49, 50, 55, 58], vector DES/Petri Nets [11, 12, 33, 39, 62, 63], model aggregation methods [2, 4, 9, 10, 13, 19, 22, 24, 44, 45, 51, 54, 57, 61], and multi-level hierarchy [6, 20, 35, 36, 53]), large-scale systems are still problematic, particularly for verification of nonblocking.

One exception is the recent work of Zhang et al. [59, 60] who have recently developed algorithms that use Integer Decision Diagrams (an extension of Binary Decision Diagrams (BDD:[7])) to verify centralized DES systems on the order of 10^{23} states. This builds upon the work by the model checking/temporal logic community [3, 16, 14, 15, 8, 25, 37, 38, 40, 47] who have successfully used BDDs to handle systems of similar size. However, Zhang's work doesn't represent a hierarchical approach, but a more efficient way to represent DES and verify properties. This means it should be possible to use it to compliment a hierarchical method for even better scalability.

To deal with the complexity of large scale systems, the software engineering community has long advocated the decomposition of software into modules (components) that interact via well defined interfaces (e.g., [23, 42, 41, 43]). Recently the supervisory control community has begun to advocate a similar approach [18, 27, 28, 30]. These approaches develop well defined interfaces between components to provide the structure to allow local checks to guarantee global properties such as controllability [18, 28, 30] or nonblocking [27, 28, 30].

In this paper, we extend the work of [27, 28, 30] which introduced a hierarchical method, called *hierarchical interface-based supervisory control* (HISC), that decomposes a system into a *high level subsystem* which communicates with $n \geq 1$ parallel *low level subsystems* through separate interfaces. This method presented a set of interface consistency properties that can be used to verify if a discrete-event system (DES) is nonblocking and controllable. As each clause of the definition can be verified using a single subsystem, the complete system model never needed to be constructed, offering significant savings in computational effort.

One limitation of the method was the definition of its interface structure, which we will refer to as *star interfaces.* This structure was too restrictive as it was not able to represent state information about the *low levels.* We introduce a new type of interface called *command-pair interfaces* that is similar, but can contain state information. We show that *star interfaces* are a special case, and extend the results for *star interfaces* to include the more general *command-pair interfaces.*

We illustrate the use of *command-pair interfaces* by re-visiting a large manufacturing example [26] with an estimated closed-loop statespace size of 7×10^{21} that was designed using only *star interfaces.* Finally, we present a complexity analysis for the method and show that the algorithm's time complexity for evaluating a system is $\mathbf{O}(m^2)$, where $m = n + 1$ is the total number of subsystems.

2 Serial Case and Command-pair Interfaces

Before we discuss *star interfaces* and introduce *command-pair interfaces,* we must first introduce the setting that they are defined in. We will do this by discussing the serial case of HISC. In the serial case, we are restricting ourselves to only one *low level* ($n = 1$). In this setting, we have 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 and information flow of the system. We call this the serial case as communication occurs in a serial fashion between the two subsystems.

To capture the restriction of the flow of information imposed by the *interface,* the alphabet of the plant (Σ) is split into four disjoint alphabets: Σ_H, Σ_L, Σ_R, and Σ_A. The events in Σ_H are called *high level events* and the events in Σ_L *low level events* as these events appear only in the high level and low level models, respectively.

The alphabets Σ_R and Σ_A are called collectively *interface events.* These events are common to both levels of the hierarchy and represent communication between the two subsystems. The events in Σ_R, called *request events*, represent commands sent from the

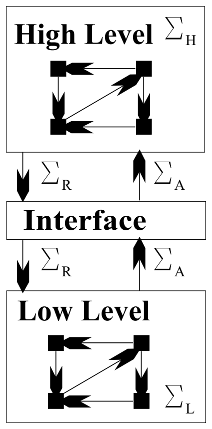

Figure 1: Interface Block Diagram.

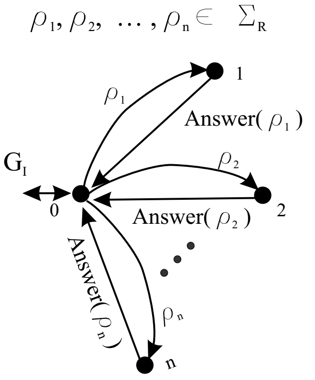

Figure 2: Interface Specification.

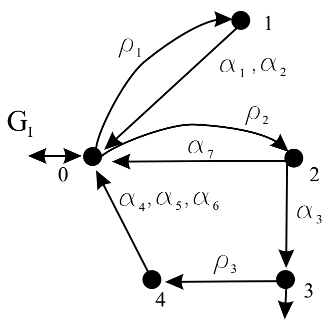

Figure 3: Example *Command-pair Interface.*

high level subsystem to the *low level subsystem.* The events in Σ_A are *answer events* and represent the *low level's* responses to the *request events.*

In [27], Leduc et al. introduced the concept of *star interfaces.*[1] This interface structure was useful 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** : $\Sigma_R \rightarrow \mathrm{Pwr}(\Sigma_A)$. For $\rho \in \Sigma_R$, **Answer**(ρ) is the set of possible answers (referred to as the *answer set*) the *low level subsystem* could provide after receiving request ρ. For consistency, the constraints that the *low level subsystem* must provide at least one response for each request it receives, and that Σ_A does not contain any unused events are added. Figure 2, shows how a *star interface,* with $n = |\Sigma_R|$ $(n \geq 0)$, is expressed as a DES. The required structure for a *star interface* is given by DES G_I. It is also required that the event set of G_I be set to $\Sigma_R \dot\cup \Sigma_A$ but no restrictions on whether a *request* or *answer event* is controllable or uncontrollable are made.

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

Definition: A DES $G_I = (X, \Sigma_I, \xi, x_o, X_m)$ is a *command-pair interface* if the following conditions are satisfied:

(A) $\Sigma_I = \Sigma_R \dot\cup \Sigma_A$

(B) $(\forall s \in L(G_I))(\forall \rho \in \Sigma_R)\ s\rho \in L(G_I) \Rightarrow s \in L_m(G_I)$

[1]Leduc et al. referred to *star interfaces* as "interfaces." We are introducing the term *star interfaces* to make it easier to refer to this type of interface structure.

(C) $(\forall s \in L_m(G_I))(\forall \sigma \in \Sigma_I)\ s\sigma \in L(G_I) \Rightarrow \sigma \notin \Sigma_A$

(D) $L_m(G_I) = \{\epsilon\} \cup (\Sigma_I^*.\Sigma_A \cap L(G_I))$

(E) $L(G_I) \subseteq \overline{(\Sigma_R.\Sigma_A)^*}$

The first point says that G_I's event set is restricted to *request* and *answer events* and that the two sets are disjoint. **Point B** states that *request event* transitions are only defined at marked states. **Point C** states that there are no *answer events* defined at marked states. **Point D** says that the marked language of G_I consists of the empty string, and strings that end in an *answer event.* Finally, **Point E** says that in the language of G_I, a *request event* always occurs first and then *request* and *answer events* alternate.

Finally, we show that *star interfaces* are a special case of *command-pair interfaces.*

Proposition 1 *If DES* $G_I = (X, \Sigma_I, \xi, x_o, X_m)$ *is a* star interface, *then* G_I *is a* command-pair interface.

Proof: See proof in [32].

2.1 Definitions and Notation

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 (defined over $\Sigma_R \cup \Sigma_A$). Also, the *high level* will mean $\textbf{sync}(G_H, G_I)$, and the *low level* $\textbf{sync}(G_L, G_I)$.[2] The overall structure of the system is displayed in Figure 4.

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

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

Whereas the representation of the system as given in Figure 4 (called the *serial subsystem based form*) is useful for verifying nonblocking as it simplifies the notation, it ignores the distinctions 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 to be $\mathcal{G}_H$, and the *high level supervisor* to be $\mathcal{S}_H$ (both defined over event set Σ_{IH}). Similarly, the *low level plant* and *supervisor* are

[2]The operation **sync** is the synchronous product operation from CTCT [56].

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

$$G_H := \textbf{sync}(\mathcal{G}_H, \mathcal{S}_H) \qquad\qquad G_L := \textbf{sync}(\mathcal{G}_L, \mathcal{S}_L)$$

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

$$\begin{array}{rlrll}
\textbf{Plant} := \textbf{sync}(\mathcal{G}_H,\, \mathcal{G}_L) & & \textbf{Sup} := \textbf{sync}(\mathcal{S}_H,\, \mathcal{S}_L,\, G_I) & & \\
\textbf{H} := P_{IH}^{-1}L(\mathcal{G}_H), & & \textbf{H}_{\mathcal{S}} := P_{IH}^{-1}L(\mathcal{S}_H), & & \subseteq \Sigma^* \\
\textbf{L} := P_{IL}^{-1}L(\mathcal{G}_L), & & \textbf{L}_{\mathcal{S}} := P_{IL}^{-1}L(\mathcal{S}_L), & & \subseteq \Sigma^*
\end{array}$$

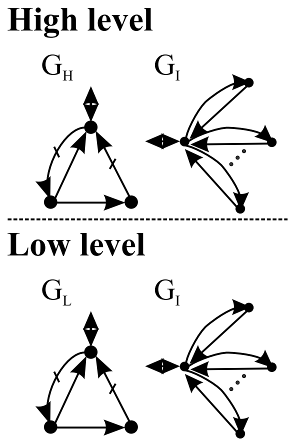

Figure 4: Two Tiered Structure of the System.

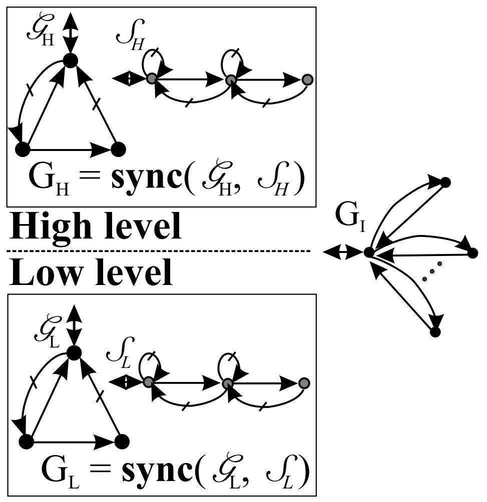
$\mathcal{G}_H$
$\mathcal{S}_H$
$G_H = \textbf{sync}(\mathcal{G}_H, \mathcal{S}_H)$
High level
G_I
Low level
$\mathcal{G}_L$
$\mathcal{S}_L$
$G_L = \textbf{sync}(\mathcal{G}_L, \mathcal{S}_L)$

Figure 5: Plant and Supervisor Subplant Decomposition

Finally, we will be using the eligibility operator in our definitions. For a language $L \subseteq \Sigma^*$ and a string $s \in \Sigma^*$, the operator $\text{Elig}_L : \Sigma^* \rightarrow \text{Pwr}(\Sigma)$ is defined as follows:

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

2.2 Serial Interface Properties and Theorems

We now present the interface requirements that the system must satisfy to ensure that it interacts with the *interface* correctly. We then define the nonblocking and controllability requirements each level must satisfy. Refer to [32] for a more detailed explanation of the requirements. The only difference between these definitions and the ones in [27, 28] is that the definitions now all refer to *command-pair interfaces* instead of assuming *star interfaces.* We are abusing notation by not changing the definition's names to reflect this since a *command-pair interface* is also a *star interface* and would thus satisfy the previous version of the definitions.

Serial Interface Consistent: 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 Σ_{IH}, and the event set of G_L is Σ_{IL}.
2. G_I is a *command-pair interface* for the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$

High Level Properties

3. $\mathcal{H}\Sigma_A \cap \mathcal{I} \subseteq \mathcal{H}$

Low Level Properties

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

Serial Level-wise Nonblocking: 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) $\overline{\mathcal{H}_m \cap \mathcal{I}_m} = \mathcal{H} \cap \mathcal{I}$ *nonblocking at the high level*

(II) $\overline{\mathcal{L}_m \cap \mathcal{I}_m} = \mathcal{L} \cap \mathcal{I}$ *nonblocking at the low level*

We now define the controllability requirements for each level. We adopt the standard partition $\Sigma = \Sigma_u \dot\cup \Sigma_c$, splitting our alphabet into *uncontrollable* and *controllable events.*

Serial Level-wise Controllable: 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 Σ_{IH}, the alphabet of $\mathcal{G}_L$ and $\mathcal{S}_L$ is Σ_{IL}, and the alphabet of G_I is Σ_I

(II) $(\mathbf{L}_{\mathcal{S}} \cap \mathcal{I})\Sigma_u \cap \mathbf{L} \subseteq \mathbf{L}_{\mathcal{S}} \cap \mathcal{I}$

(III) $\mathbf{H}_{\mathcal{S}}\Sigma_u \cap (\mathbf{H} \cap \mathcal{I}) \subseteq \mathbf{H}_{\mathcal{S}}$.

We now present our main results for this chapter, the *serial interface nonblocking theorem* and the *serial controllability theorem.* As the *serial level-wise nonblocking, serial level-wise Controllable,* and *serial interface consistent* definitions can be evaluated by examining only one level of our system at a time, we now have a means to verify nonblocking of our system using local checks. The difference between these theorems and the ones in [27, 28] is that they now use the *command-pair interface* definitions.[3]

[3]Note, the controllability results here, and later for the parallel case, are automatic from [27, 28] as they don't rely on the specific structure of the *star interface*, just its event set. They are included for completeness.

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*

$$L(G) = \overline{L}_m(G), \text{ where } \quad G = \textbf{sync}(G_H, G_L, G_I)$$

Proof: See proof in [32].

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:*

$$(\forall s \in L(\textbf{Plant}) \cap L(\textbf{Sup})) \quad Elig_{L(\textbf{Plant})}(s) \cap \Sigma_u \subseteq Elig_{L(\textbf{Sup})}(s)$$

where $\textbf{Plant} = \textbf{sync}(\mathcal{G}_H, \mathcal{G}_L)$ *and* $\textbf{Sup} = \textbf{sync}(\mathcal{S}_H, \mathcal{S}_L, G_I)$.

Proof: See proof in [32].

3 Parallel Case

In Section 2, we described the serial case for the HISC method where the number of *low levels* (n) is restricted to one. We now describe the more general setting where we have $n \geq 1$ *low levels.* Figure 6 shows conceptually the structure and flow of information of such a system. In this new setting, we still have a single *high level,* but this time it is interacting with $n \geq 1$ independent low levels, communicating with each *low level* in parallel through a separate *interface.* We will refer to the number of *low levels*, n, as the *degree* of the system.

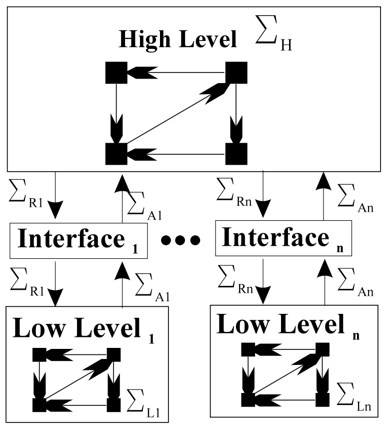

Figure 6: Parallel Interface Block Diagram.

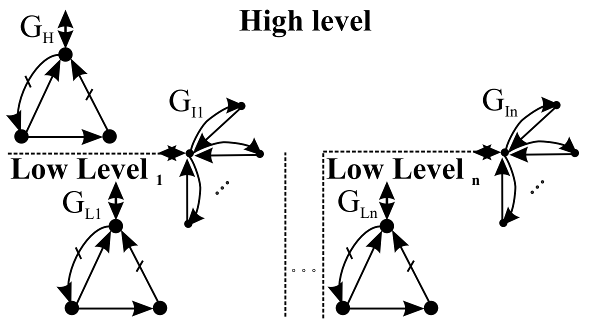

Figure 7: Two Tiered Structure of Parallel System

As in the serial case, in order to capture the restriction of the flow of information imposed by the *interface,* we partition the alphabet of the system into the following analogous pairwise disjoint alphabets: Σ_H, Σ_{R_j}, Σ_{A_j}, and Σ_{L_j}, with $j = 1, \ldots, n$.

For an n^{th} degree parallel system, we assume the *high level subsystem* is modelled by DES G_H (defined over event set $\dot\cup_{j\in\{1,\ldots,n\}}[\Sigma_{R_j} \dot\cup \Sigma_{A_j}] \dot\cup \Sigma_H$). For $j \in \{1, \ldots, n\}$, the j^{th}

low level subsystem is modelled by DES G_{L_j} (defined over event set $\Sigma_{L_j} \dot\cup \Sigma_{R_j} \dot\cup \Sigma_{A_j}$), the j^{th} *interface* by DES G_{I_j} (defined over event set $\Sigma_{R_j} \dot\cup \Sigma_{A_j}$), and that the overall system has the structure shown in Figure 7. Furthermore, we will refer to the j^{th} *low level* to mean $\mathbf{sync}(G_{L_j}, G_{I_j})$ and we will assume that the alphabet partition is specified by $\Sigma := \dot\cup_{j\in\{1,\ldots,n\}}[\Sigma_{L_j} \dot\cup \Sigma_{R_j} \dot\cup \Sigma_{A_j}] \dot\cup \Sigma_H$ and that the *flat system* is taken to be:

$$G = \mathbf{sync}(G_H, G_{L_1}, \ldots, G_{L_n}, G_{I_1}, \ldots, G_{I_n})$$

In order to simplify the notation in proofs, we now introduce the following event sets, natural projections, and useful languages. For the remainder of this section, the index j is defined to have range $\{1, \ldots, n\}$.

$$\begin{aligned}
\Sigma_{I_j} &:= \Sigma_{R_j} \cup \Sigma_{A_j} \\
\Sigma_{IH} &:= \cup_{j\in\{1,\ldots,n\}} \Sigma_{I_j} \cup \Sigma_H \\
\Sigma_{IL_j} &:= \Sigma_{L_j} \cup \Sigma_{I_j} \\
P_{IH} : \Sigma^* &\to \Sigma_{IH}^* \\
P_{IL_j} : \Sigma^* &\to \Sigma_{IL_j}^* \\
P_{I_j} : \Sigma^* &\to \Sigma_{I_j}^*
\end{aligned}$$

$$\begin{array}{rll}
\mathcal{H} := P_{IH}^{-1}(L(G_H)), & \mathcal{H}_m := P_{IH}^{-1}(L_m(G_H)) & \subseteq \Sigma^* \\
\mathcal{L}_j := P_{IL_j}^{-1}(L(G_{L_j})), & \mathcal{L}_{m_j} := P_{IL_j}^{-1}(L_m(G_{L_j})) & \subseteq \Sigma^* \\
\mathcal{I}_j := P_{I_j}^{-1}(L(G_{I_j})), & \mathcal{I}_{m_j} := P_{I_j}^{-1}(L_m(G_{I_j})) & \subseteq \Sigma^*
\end{array}$$

3.1 General Form

As in the serial case, we need to be able to decompose the n^{th} degree ($n \geq 1$) *parallel interface system* into its plant and supervisor components.

We now define the *high level plant* to to be $\mathcal{G}_H$, and the *high level supervisor* to be $\mathcal{S}_H$ (both defined over Σ_{IH}). Similarly, the j^{th} *low level plant* and *supervisor* are $\mathcal{G}_{L_j}$ and $\mathcal{S}_{L_j}$(defined over Σ_{IL_j}). We now define the *high level subsystem* and *the* j^{th} *low level subsystem* as follows:

$$G_H := \mathbf{sync}(\mathcal{G}_H, \mathcal{S}_H) \qquad\qquad G_{L_j} := \mathbf{sync}(\mathcal{G}_{L_j}, \mathcal{S}_{L_j})$$

The reader should note that the definition of a *parallel interface system* that we present here in terms of plant and supervisor components, is the *general form* of such systems. We will refer to the original form shown in Figure 7, used to simplify nonblocking definitions and proofs, as the *parallel subsystem based form.*

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

$$\begin{array}{ll}
\mathbf{Plant} := \mathbf{sync}(\mathcal{G}_H, \mathcal{G}_{L_1}, \ldots, \mathcal{G}_{L_n}) & \mathbf{Sup} := \mathbf{sync}(\mathcal{S}_H, \mathcal{S}_{L_1}, \ldots, \mathcal{S}_{L_n}, G_{I_1}, \ldots, G_{I_n}) \\
\mathbf{H} := P_{IH}^{-1} L(\mathcal{G}_H), & \mathbf{H}_{\mathcal{S}} := P_{IH}^{-1} L(\mathcal{S}_H), \quad \subseteq \Sigma^* \\
\mathbf{L}_j := P_{IL_j}^{-1} L(\mathcal{G}_{L_j}), & \mathbf{L}_{\mathcal{S}_j} := P_{IL_j}^{-1} L(\mathcal{S}_{L_j}), \quad \subseteq \Sigma^*
\end{array}$$

3.2 Serial System Extraction

As the event set of each *low level* is mutually exclusive from the event sets of the other *low levels,* we can consider the *parallel interface system* as n *serial interface systems* by choosing one *low level* and ignoring the others. This will allow us to reuse our existing definitions and results for *serial interface systems.*

In this section, we introduce the concept of *serial system extractions* for an n^{th} degree ($n \geq 1$) *parallel interface system,* shown conceptually in Figure 8 in terms of subsystems. Below we give the general form of the definition. The parallel subsystem form of the definition can be obtained by using the identities $G_H = \textbf{sync}(\mathcal{G}_H, \mathcal{S}_H)$, $G_L = \textbf{sync}(\mathcal{G}_L, \mathcal{S}_L)$, and $G_{L_j} = \textbf{sync}(\mathcal{G}_{L_j}, \mathcal{S}_{L_j})$.

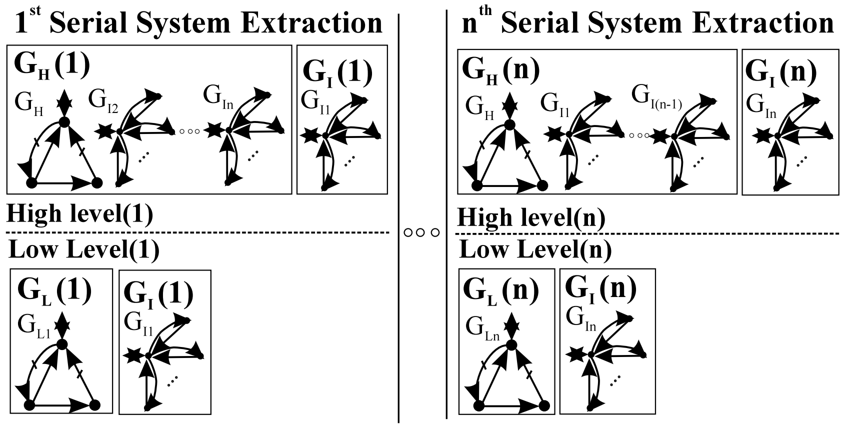

Figure 8: The Serial System Extraction

j^{th} Serial System Extraction: For the n^{th} degree ($n \geq 1$) *parallel interface system* composed of DES $\mathcal{G}_H, \mathcal{G}_{L_1}, \ldots, \mathcal{G}_{L_n}, \mathcal{S}_H, \mathcal{S}_{L_1}, \ldots, \mathcal{S}_{L_n}, G_{I_1}, \ldots, G_{I_n}$, with alphabet partition $\Sigma := \dot\cup_{k\in\{1,\ldots,n\}}[\Sigma_{L_k}\dot\cup\Sigma_{R_k}\dot\cup\Sigma_{A_k}] \,\dot\cup\, \Sigma_H$, the j^{th} *serial system extraction,* denoted by *system*(j), is composed of the following elements:

$$\begin{aligned}
\mathcal{G}_H(j) &:= \textbf{sync}(\mathcal{G}_H, G_{I_1}, \ldots, G_{I_{(j-1)}}, G_{I_{(j+1)}}, \ldots, G_{I_n}) \\
\mathcal{S}_H(j) &:= \mathcal{S}_H \ , \quad \mathcal{G}_L(j) := \mathcal{G}_{L_j} \ , \quad \mathcal{S}_L(j) := \mathcal{S}_{L_j} \ , \quad G_I(j) := G_{I_j} \\
\Sigma_H(j) &:= \dot\cup_{k\in\{1,\ldots,(j-1),(j+1),\ldots,n\}}\Sigma_{I_k} \,\dot\cup\, \Sigma_H \\
\Sigma_L(j) &:= \Sigma_{L_j} \ , \quad \Sigma_R(j) := \Sigma_{R_j} \ , \quad \Sigma_A(j) := \Sigma_{A_j} \\
\Sigma(j) &:= \Sigma_H(j) \,\dot\cup\, \Sigma_L(j) \,\dot\cup\, \Sigma_R(j) \,\dot\cup\, \Sigma_A(j) \\
&= \Sigma - \dot\cup_{k\in\{1,\ldots,(j-1),(j+1),\ldots,n\}}\Sigma_{L_k}
\end{aligned}$$

3.3 Parallel Case Definitions and Theorems

In this section we present a set of properties that are equivalent to their serial interface counterparts. The only difference between these definitions and the ones in [27, 28] is that they now use *command-pair interfaces* instead of assuming *star interfaces.*

Interface Consistent: The n^{th} degree ($n \geq 1$) *parallel interface system* composed of DES $G_H, G_{L_1}, \ldots, G_{L_n}, G_{I_1}, \ldots, G_{I_n}$, is *interface consistent* with respect to alphabet partition $\Sigma := \dot\cup_{k\in\{1,\ldots,n\}}[\Sigma_{L_k}\dot\cup\Sigma_{R_k}\dot\cup\Sigma_{A_k}] \,\dot\cup\, \Sigma_H$, if:

$(\forall j \in \{1,\ldots,n\})$ The j^{th} *serial system extraction* of the system is *serial interface consistent.*

Level-wise Nonblocking: The n^{th} degree $(n \geq 1)$*parallel interface system* composed of DES $G_H, G_{L_1}, \ldots, G_{L_n}, G_{I_1}, \ldots, G_{I_n}$, is *level-wise nonblocking* with respect to the alphabet partition $\Sigma := \dot\cup_{k\in\{1,\ldots,n\}}[\Sigma_{L_k}\dot\cup\Sigma_{R_k}\dot\cup\Sigma_{A_k}] \;\dot\cup\; \Sigma_H$, if:

$(\forall j \in \{1,\ldots,n\})$ The j^{th} *serial system extraction* of the system is *serial level-wise nonblocking.*

We now extend *serial level-wise controllability* to the parallel case. We adopt the standard partition $\Sigma = \Sigma_u \,\dot\cup\, \Sigma_c$, splitting our alphabet into *uncontrollable* and *controllable events.*

Level-wise Controllable: The n^{th} degree $(n \geq 1)$ *parallel interface system* composed of DES $\mathcal{G}_H$, $\mathcal{G}_{L_1}, \ldots, \mathcal{G}_{L_n}$, $\mathcal{S}_H$, $\mathcal{S}_{L_1}, \ldots, \mathcal{S}_{L_n}$, $G_{I_1}, \ldots, G_{I_n}$, is *level-wise controllable* with respect to alphabet partition $\Sigma := \dot\cup_{k\in\{1,\ldots,n\}}[\Sigma_{L_k}\dot\cup\Sigma_{R_k}\dot\cup\Sigma_{A_k}] \;\dot\cup\; \Sigma_H$, if:

$(\forall j \in \{1,\ldots,n\})$ The j^{th} *serial system extraction* of the system is *serial level-wise controllable.*

We now present our nonblocking theorem for parallel interface systems. It states that, to verify if a parallel system is nonblocking, it is sufficient to check that each of its *serial system extractions* is *serial level-wise nonblocking* and *serial interface consistent.* The difference between these theorems and the ones in [27, 28] is that they now use the *command-pair interface* definitions.

Theorem 3 *If the n^{th} degree ($n \geq 1$)* parallel interface system *composed of DES G_H, $G_{L_1}, \ldots, G_{L_n}, G_{I_1}, \ldots, G_{I_n}$, is* level-wise nonblocking *and* interface consistent *with respect to the alphabet partition $\Sigma := \dot\cup_{k\in\{1,\ldots,n\}}[\Sigma_{L_k}\dot\cup\Sigma_{R_k}\dot\cup\Sigma_{A_k}] \;\dot\cup\; \Sigma_H$, then*

$L(G) = \overline{L}_m(G)$, *where* $\quad G = \mathbf{sync}(G_H,\, G_{L1}, \ldots,\, G_{Ln}, G_{I1}, \ldots, G_{In})$

Proof: See proof in [32].

Next, we present our controllability theorem for *parallel interface systems.* It states that, to verify if a parallel system is controllable, it is sufficient to check that each of its *serial system extractions* is *serial level-wise controllable.*

Theorem 4 *If the n^{th} degree ($n \geq 1$)* parallel interface system *composed of plant components $\mathcal{G}_H$, $\mathcal{G}_{L_1}, \ldots, \mathcal{G}_{L_n}$, supervisors $\mathcal{S}_H$, $\mathcal{S}_{L_1}, \ldots, \mathcal{S}_{L_n}$, and interfaces $G_{I_1}, \ldots, G_{I_n}$, is* level-wise controllable *with respect to the alphabet partition $\Sigma := \dot\cup_{k\in\{1,\ldots,n\}}[\Sigma_{L_k}\dot\cup\Sigma_{R_k}\dot\cup\Sigma_{A_k}]$ $\dot\cup\; \Sigma_H$, then*

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

where **Plant** $:= \mathbf{sync}(\mathcal{G}_H,\, \mathcal{G}_{L_1}, \ldots, \mathcal{G}_{L_n})$ *is the system's* flat plant, *and*
Sup $:= \mathbf{sync}(\mathcal{S}_H,\, \mathcal{S}_{L_1}, \ldots, \mathcal{S}_{L_n}, G_{I_1}, \ldots, G_{I_n})$ *is the system's* flat supervisor.

Proof: See proof in [32].

4 Application to the AIP

We now revisit an application to a large manufacturing system, the Atelier Inter-établissement de Productique (AIP) discussed in [26], to illustrate the use of *command-pair interfaces.* The AIP, shown in Figure 9, is a highly automated manufacturing system consisting of a central loop (CL) and four external loops (EL), three assembly stations (AS), an input/output (I/O) station, and four inter-loop transfer units (TU). The I/O station is where the pallets enter and leave the system. Pallets can be of type 1 or of type 2, and it is assumed that the type of the pallet entering is random.

4.1 Assembly Stations

The structure of the assembly stations is shown in Figure 10. Each station 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 pallet sensors to detect a pallet at the pallet gate, the pallet stop, and 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 to substitute for the other stations when they are down.

4.2 Transport Units

The structure of the four identical transport units is shown in Figure 11. 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.

4.3 Using Command-pair Interfaces

In [26], the AIP was modelled using only *star interfaces.* The system was designed as a 7^{th} degree *parallel interface system* as shown in Figure 12, with the *low levels* representing the three assembly stations, and four transfer units. For full design details, refer to [32].

The design of the *low level* for assembly station 1 was poorly suited to being modelled by *star interfaces.* This can be seen by examining its *star interface,* shown in Figure 13. We see that the AS1 has two *request events, ProcPallet.AS1* and *DoRpr.AS1.* Clearly, it only makes sense to do a repair, after the *answer event ASDwn.k* has occurred. Also, it

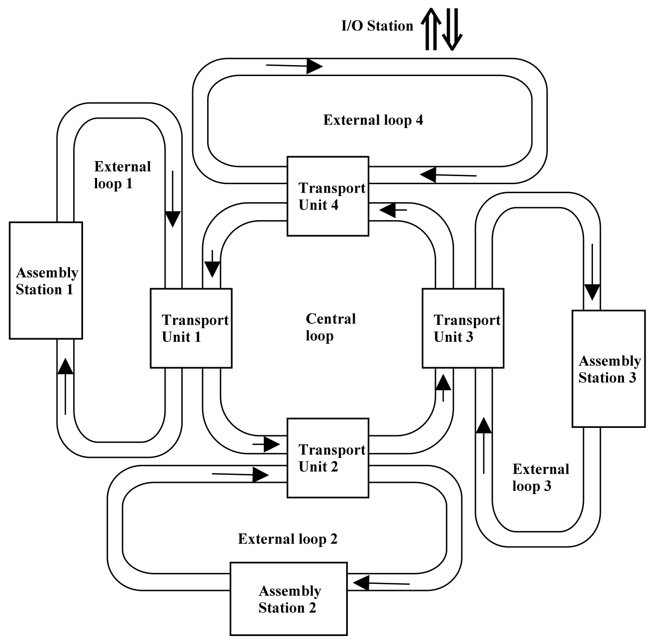

Figure 9: The Atelier Inter-établissement de Productique

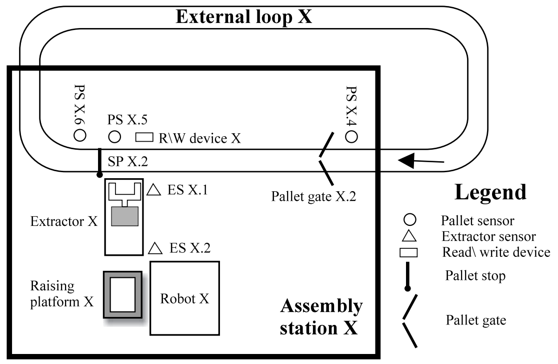

Figure 10: Assembly Station of External Loop $X = 1, 2, 3$.

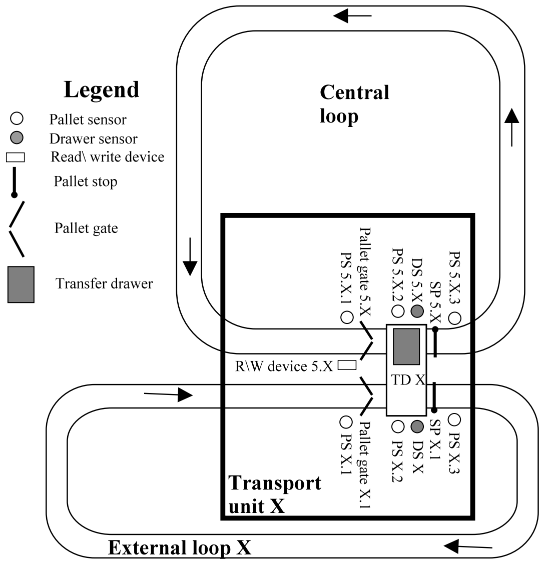

Figure 11: Transport Unit for External Loop $X = 1, 2, 3, 4$

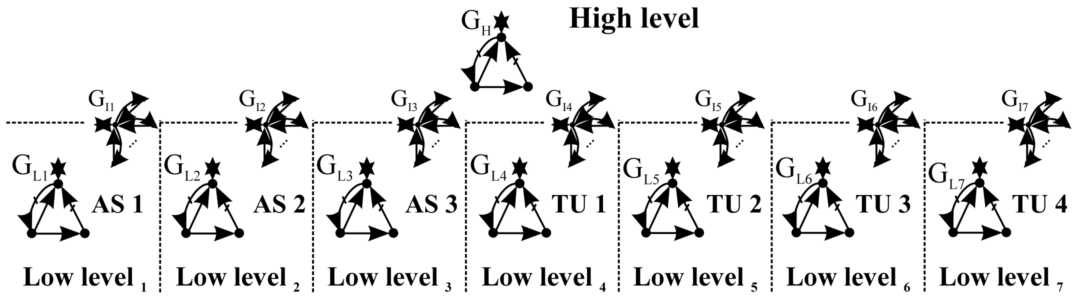

Figure 12: Structure of Parallel System

doesn't make sense to try to process a pallet while the AS is down. With *star interfaces,* all *request events* are always possible because of the "star" structure, so we couldn't model this. With *command-pair interfaces,* this is easily accomplished as in Figure 13.

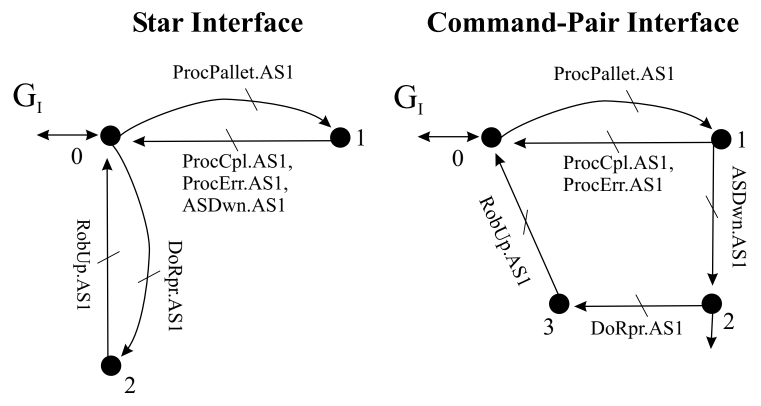

Figure 13: Interfaces for Assembly Station 1

In particular, representing the assembly station as a *star interface* is difficult because of **Point 5** of the *serial interface consistent* definition. This point says that after a *request event* has occurred (such as *ProcPallet.AS1*), then all answer events that the *interface* says can follow the event (ie. for *star interface*, events at *state 1*) must be possible after at most a sequence of low level events. However once the station is down, the physical events *ProcCplAS1* and *ProcErrAS1* can't occur until it has been repaired. How this was resolved in [26] can be seen in Figure 14 where events *RobDwn.AS1, RtasksCpl.AS1, RobUp.AS1,* and *AssmbErrA/B.AS1* roughly correspond to the station's *answer events* and *ProcType1/2.AS1* and *DoRpr.AS1* roughly correspond to the *request events.* We see that the *answer events* have the correct physical meaning most of the time, but if the robot is down (*state s11*) and a request to process a pallet is made, all *answer events* are possible and which one occurs is randomly selected and has no physical meaning. This allows **Point 5** to be satisfied and as long as the *high level* is careful to never try to process a pallet while AS1 is down, this situation will never be encountered. This ugly kludge is unnecessary if *command-pair interfaces* are used.

To finish converting the AIP example to use *command-pair interfaces,* we need to make some additional changes. As assembly station 2 is identical upto relabelling, we need to modify it as well. To present the changes to the two assembly stations, we will describe them collectively as *low level subsystem* w, where $w = 1, 2$. We also define the companion index $k = \text{AS1}, \text{AS2}$, which takes its values relative to w (eg. $k = \text{AS1}$ when $w = 1$).

The first changes are to the interface for *low level* w, shown in Figure 15. This is identical to the interface shown in Figure 13, but generalized to include the interface for station 2. The next required change is to supervisor **Intf-k-Robot.k**, shown in Figure 16, who defines the tasks that the robot can perform. It is changed to the DES in Figure 18.

The last required change is to supervisors **DoRobotTasks.AS1** and **DoRobot-Tasks.AS2,** shown in Figures 14 and 20, who control the operation of the robots. They make sure that the assembly tasks are performed in the correct order for a given type of pallet, they report on the success of the assembly operation, and they handle repairs

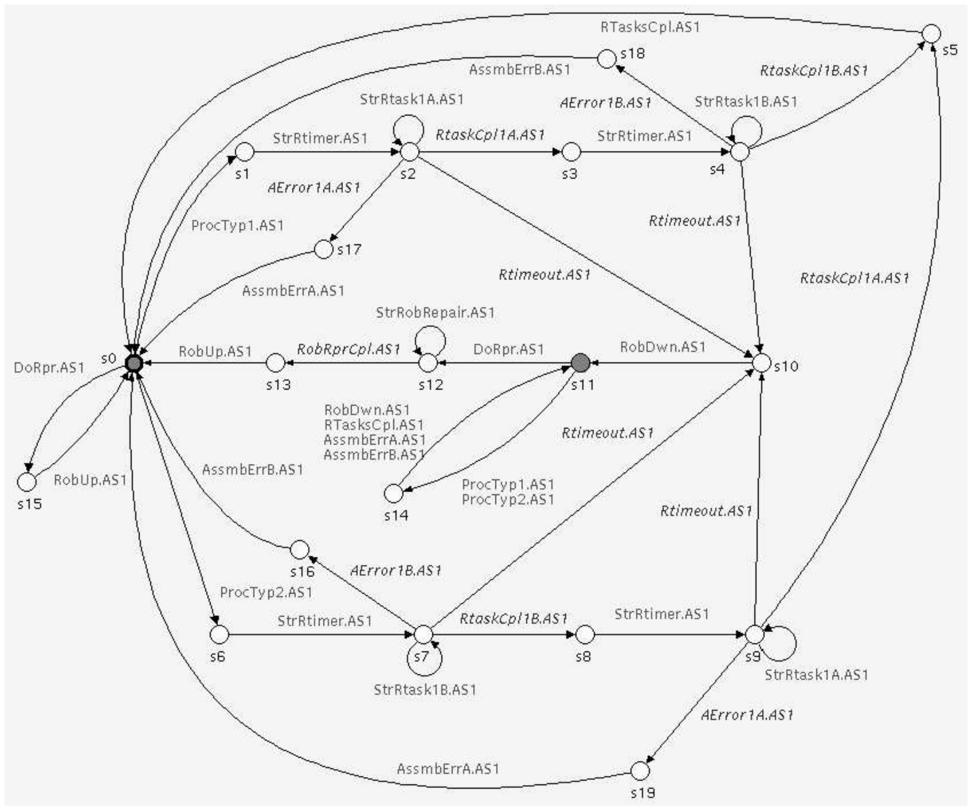

Figure 14: DoRobotTasks.AS1

when the robot breaks down. They are changed to the DES in Figures 19, and 21, where states 14 and 15 have been deleted.

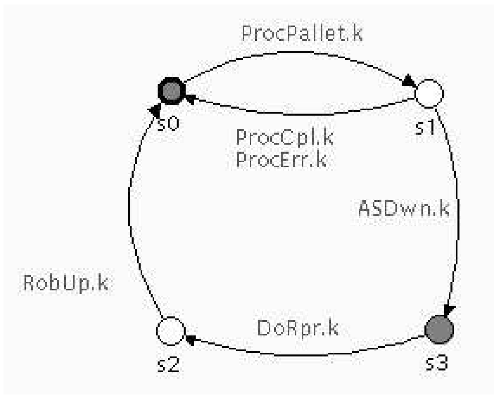

Figure 15: Interface to *Low Level* w-cp.

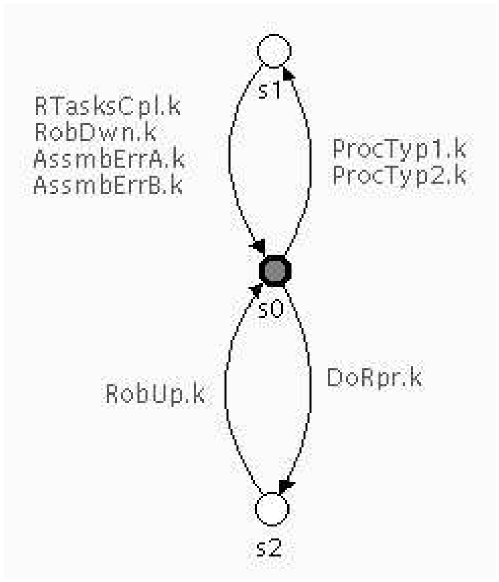

Figure 16: Intf-k-Robot.k

We now apply our research tool to the seven *serial extraction systems* (the two we modified and the five original systems from [26]) and we find that they are all *serial level-wise non-blocking, serial level-wise controllable,* and *serial interface consistent.* We

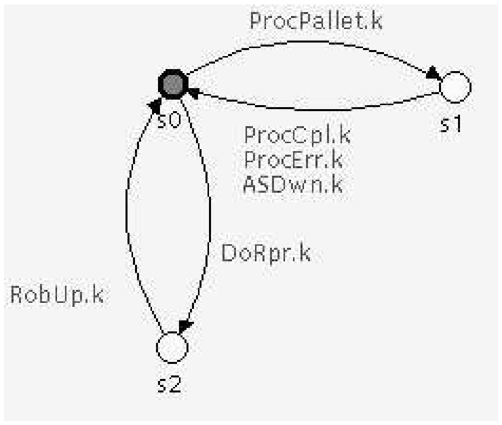

Figure 17: Interface to *Low Level w*.

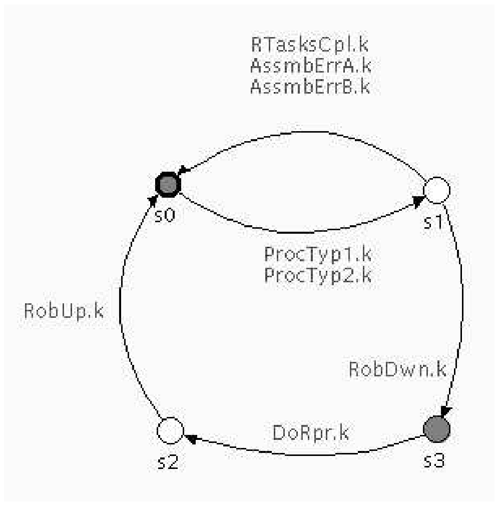

Figure 18: Intf-k-Robot.k-cp

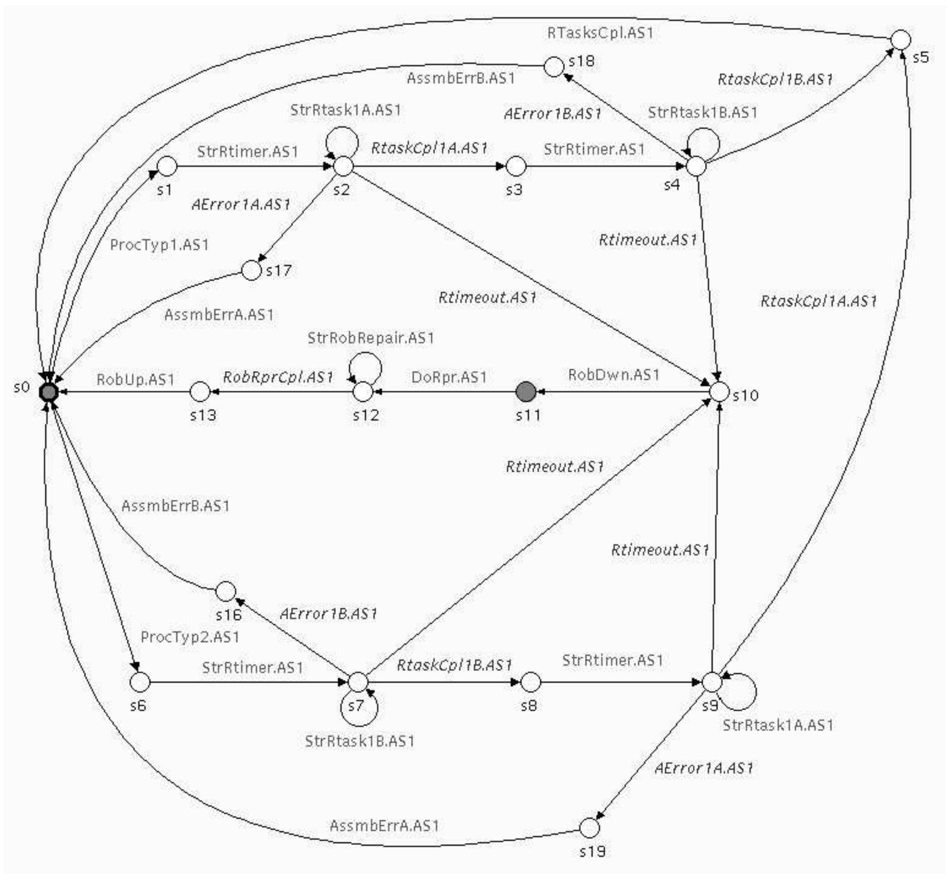

Figure 19: DoRobotTasks.AS1-cp

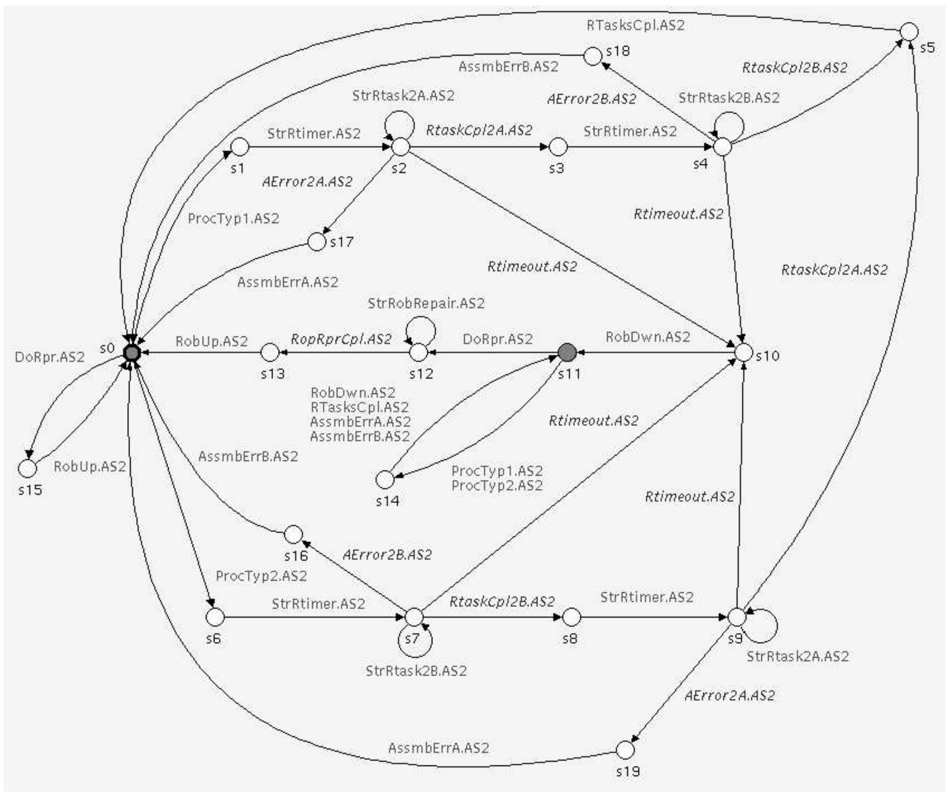

Figure 20: DoRobotTasks.AS2

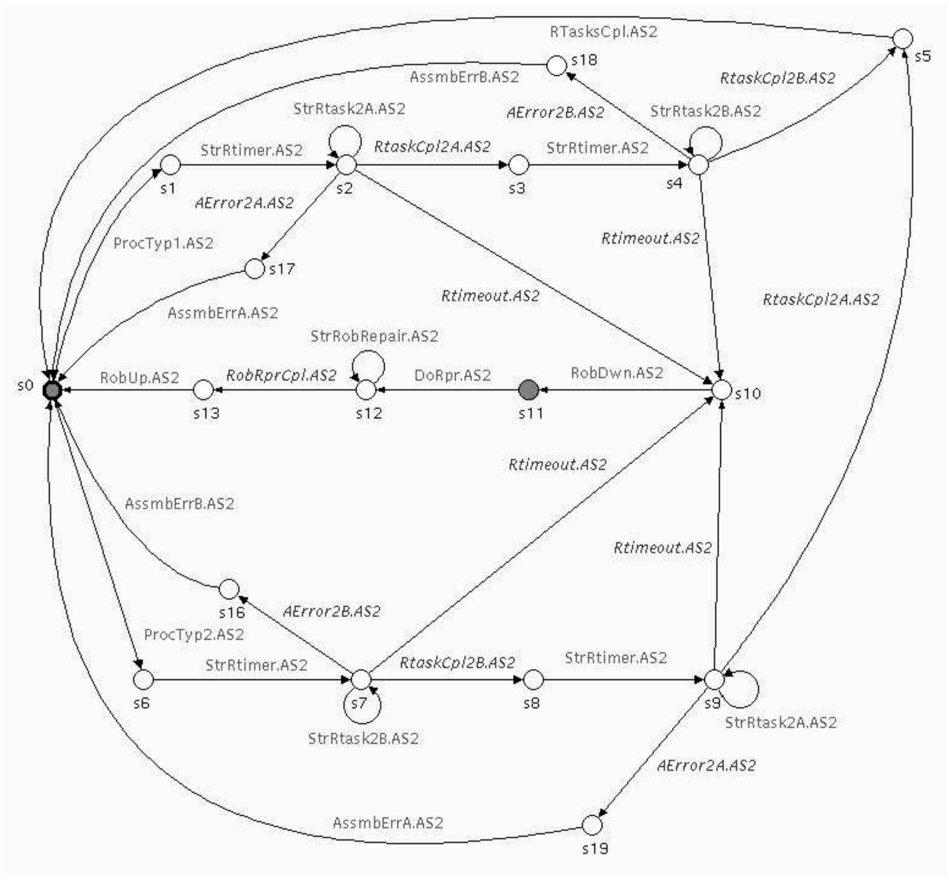

Figure 21: DoRobotTasks.AS2-cp

can thus conclude that the system is *level-wise non-blocking, level-wise controllable,* and *interface consistent.* This allows us to conclude by **Theorems 3** and **4**, that the *flat system* is nonblocking and that the system's *flat supervisor* is controllable for the *flat plant.*

5 Complexity Analysis

To aid in investigating *hierarchical interface-based supervisory control,* we have developed software routines to verify that a system satisfies the conditions *serial level-wise nonblocking and controllable,* and *serial interface consistent.* The routines were developed by Leduc during his collaboration with Siemens Corporate Research and they use the algorithms described in [32].

Analyzing the steps required to verify the above conditions, we see that they consist of verifying system properties (ie. is G_I a *command-pair interfaces*), high level properties (ie. does G_H satisfy **Point 3** of the *serial interface consistent* definition) and low level properties (ie. does G_L satisfy **Point 4** of the *serial interface consistent* definition). We next note that all the properties to be verified for the *high level* have an equivalent property for the *low level* and thus we can treat the *high level* as a *low level* for our analysis. If we group the verification of the system properties with the low level properties, verifying two "low levels" (components) can be used as an upper bound for verifying the system.

The next logical step would be to perform an analytic analysis of the worst case time complexity for evaluating a "low level." Unfortunately, to do this we would require program sourcecode to provide details of the data structures used and how they are accessed. The only sourcecode available is copyrighted by Siemens and cannot be released. Instead, we follow the advice of Goodrich et al. [21] and use experimental algorithm analysis to estimate the worse case time complexity for per component analysis. As we will see, this is sufficient as the per component complexity only contributes a constant term to the overall complexity of evaluating a system.

To perform this analysis, we will assume that the running time for one component is of the form $t(x) = bx^c$ with x the state size of our component and for some constants $b > 0$ and $c > 0$. We then use the *power test* discussed in [21] to experimentally determine the worst case running time to be $t(x) = (8.56 \times 10^{-9})x^3$ which makes the algorithm $\mathbf{O}(x^3)$. See Chapter 6 of [32] for details on this process.

We next consider verifying an n^{th} degree *parallel interface system.* To do this, we must check that $3n+1$ event sets are pairwise disjoint and check that n *serial extraction systems* are *serial level-wise nonblocking and controllable,* and *serial interface consistent.* We let $m = n + 1$ be the number of components to be verified. We also assume that the statespace (x) of each component and the cardinality of the system's event set (Σ) are bounded with upper bounds $N \geq 0$ and $N_\Sigma \geq 0$, respectively. We further assume that the cardinality of event sets Σ_H, $\Sigma_{L_1}, \ldots, \Sigma_{L_n}$, $\Sigma_{R_1}, \ldots, \Sigma_{R_n}$ and $\Sigma_{A_1}, \ldots, \Sigma_{A_n}$ is each bounded by $N_{\Sigma'} \geq 0$ (ie. $|\Sigma_H| \leq N_{\Sigma'}$).

It can be shown that verifying $3n+1$ event sets are pairwise disjoint can be performed by $\frac{9}{2}m^2 - \frac{15}{2}m + 3$ empty intersection tests which are each (from [48]) $\mathbf{O}(N_{\Sigma'}^2)$. The whole process is thus $\mathbf{O}(\frac{9}{2}m^2 N_{\Sigma'}^2 - \frac{15}{2}mN_{\Sigma'}^2 + 3N_{\Sigma'}^2) = \mathbf{O}(m^2)$.

To verify the n *serial extraction systems*, we must perform the per component analysis $2n$ times. As the per component analysis is $\mathbf{O}(x^3) = \mathbf{O}(N^3)$, the system analysis is thus $\mathbf{O}(2n \cdot N^3) = \mathbf{O}(2mN^3 - 2N^3) = \mathbf{O}(m)$ as N is a constant. Combining the two steps, we find that verifying a n^{th} degree parallel interface system is $\mathbf{O}(m^2 + m) = \mathbf{O}(m^2)$. In practice, the $2mN^3$ is much larger than the $\frac{9}{2}m^2N_{\Sigma'}^2$ term and thus the algorithm behaves as if it is $\mathbf{O}(m)$. Of course, this only remains practical as long as N isn't so large that it contributes a prohibitively large constant term.

We next compare the HIS method to verifying non-blocking of the the synchronous product of our m components. Based on the work of Rudie [48], it can be shown (see [32] for details) that the monolithic algorithm is $\mathbf{O}(N^{2m})$ and thus our algorithm scales significantly better. To illustrate this, let's examine the two algorithms for a few values of N, $N_{\Sigma'}$, and m. Table 1 shows the results for terms $T_1 = N^{2m}$, and $T_2 = 2mN^3 - 2N^3 + \frac{9}{2}m^2N_{\Sigma'}^2 - \frac{15}{2}mN_{\Sigma'}^2 + 3N_{\Sigma'}^2$. We see that even for $m = 2$ (serial system) and $N = 10^6$, our approach is six orders of magnitude better. To put this into perspective, if our algorithm ran for one hour, the monolithic algorithm would require 114 years!

		$m = 2$		$m = 9$	
N	$N_{\Sigma'}$	T_1	T_2	T_1	T_2
10^3	10^2	10^{12}	2×10^9	10^{54}	1.60×10^{10}
10^6	10^2	10^{24}	2×10^{18}	10^{108}	1.60×10^{19}

Table 1: Parallel Algorithm Comparison

Of course, there is a cost for this increase in computational efficiency. The trade-off is a more restrictive architecture. The interface approach restricts knowledge about internal details of components, and only allows supervisors to disable local events and *interface events.* As similar interface-based approaches are common in both hardware and software, we are confident that our method will be widely applicable.

UPDATE: The analysis presented here relies on the assumption that the states-pace of each component is bounded by the constant N. As long as this assumption is reasonable, the analysis is correct. For the DES $G_H, G_{L_1}, \ldots, G_{L_n}, G_{I_1}, \ldots, G_{I_n}$, this assumption is reasonable.

However, when analyzing the conditions *interface consistent, level-wise nonblocking,* and *level-wise controllable,* we must construct *serial extraction systems* (see Section 3.2) to analyze the corresponding serial conditions. For example, to verify that the *parallel interface system* is *interface consistent,* we must verify that all n *serial system extractions (subsystem form)* are *serial interface consistent.* To verify the latter condition, we must use the component $G_H(j) := G_H||_sG_{I_1}||_s \ldots ||_sG_{I_{(j-1)}}||_sG_{I_{(j+1)}}||_s \ldots ||_sG_{I_n}$ with the serial algorithms we developed in [32]. Unlike the DES G_H, component $G_H(j)$ grows proportionally to n, thus the assumption that $G_H(j)$ is bounded by N is questionable. In this view, the above analysis is a bit too optimistic and is thus more in line with an average or best case analysis. This does not mean that the approach does not have great potential to scale. For a good scalability discussion, see [29].

6 Conclusions

Hierarchical interface-based supervisory control offers an effective method to model systems with a natural client-server architecture. By introducing *command-pair interfaces,* we have extended the modelling flexibility for *interfaces* by allowing the representation of low level state information, enabling many new systems to be easily cast as *low levels.*

From examining the definitions in Sections 2, and 3, it's clear that each requirement can be verified using only one subsystem. This means that the entire plant model never needs to be constructed or traversed (in computer memory), offering potentially significant savings in computation. We have shown this concretely by proving that the time complexity for analyzing a system by our method is $\mathbf{O}(m^2)$ ($m = n + 1$ is the total number of subsystems), as compared to a monolithic analysis which is $\mathbf{O}(N^{2m})$ ($N \geq 0$ is an upper bound for the statespace of the subsystems).

References

[1] N. Alsop. *Formal Techniques for the Procedural Control of Industrial Processes.* PhD thesis, Department of Chemical Engineering and Chemical Technology, Imperial College of Science, Technology and Medicine, London, 1996.

[2] Rajeev Alur and Thomas A. 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*, pages 166–179, 1995.

[3] A. Arnold. *Finite Transition Systems.* Prentice Hall, 1994.

[4] Adnan Aziz, Vigyan Singhal, and Gitanjali M. 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*, pages 255–261, Cambridge, Massachusetts, Oct 1994.

[5] George Barrett and Stephane Lafortune. Decentralized supervisory control with communicating controllers. *IEEE Trans. Automatic Control*, 45(9):1620–1638, 2000.

[6] Y. Brave and M. Heymann. Control of discrete event systems modeled as hierarchical state machines. *IEEE Trans. on Automatic Control*, 38(12):1803–1819, Dec 1993.

[7] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, C-35(8), 1986.

[8] J.R. Burch, Edmund M. Clarke, and K.L. McMillan. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.

[9] P.E. Caines and Y.J. Wei. The hierarchical lattices of a finite machine. *Systems Control Letters*, 25:257–263, July 1995.

[10] Haoxun Chen and Hans-Michael Hanisch. Model aggregation for hierarchical control synthesis of discrete event systems. In *Proc. 39th Conf. Decision Contr.*, pages 418–423, Sydney, Australia, December 2000.

[11] S.-L. Chen. Existence and design of supervisors for vector discrete event systems. Master's thesis, Department of Electrical Engineering, University of Toronto, Toronto, Ont, 1992.

[12] S.-L. Chen. *Control of Discrete-Event Systems of Vector and Mixed Structural Type.* PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1996.

[13] Yi-Liang Chen and Feng Lin. Hierarchical modeling and abstraction of discrete event systems using finite state machines with parameters. In *Proc. 40th Conf. Decision Contr.*, pages 4110–4115, Orlando, USA, December 2001.

[14] Edmund M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Programming Languages and Systems*, 8(2):244–263, April 1986.

[15] Edmund M. Clarke, O. Grümberg, and K. Hamaguchi. Another look at LTL model checking. In *Proc. of 6th Conf. on Computer Aided Verification*, number 818 in LNCS, pages 415–427. Springer-Verlag, 1994.

[16] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking.* MIT Press, 2001.

[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*, pages 571–576, Budapest, 1993.

[18] 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.

[19] Jose M. Eyzell and Jose E.R. Cury. Exploiting symmetry in the synthesis of supervisors for discrete event systems. In *Proc. of American Control Conference*, pages 244–248, Philadelphia, USA, June 1998.

[20] Peyman Gohari-Moghadam. A linguistic framework for controlled hierarchical DES. Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1998.

[21] Michael T. Goodrich and Roberto Tamassia. *Algorithm Design.* Wiley, 2001.

[22] O. Grümberg and D.E. Long. Model checking and modular verification. In *Proc. of CONCOUR'91*, number 527 in LNCS, pages 361–375. Springer-Verlag, 1991.

[23] Daniel M. Hoffman and David M. Weiss, editors. *Software Fundamentals. Collected Papers by David L. Parnas.* Addison Wesley, 2001.

[24] Paul Hubbard and Peter E. Caines. Trace-DC hierarchical supervisory control with applications to transfer-lines. In *Proc. 37th Conf. Decision Contr.*, pages 3293–3298, Tampa, Florida USA, December 1998.

[25] Mark Lawford. *Model Reduction of Discrete Real-Time Systems.* PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1997.

[26] R. Leduc, M. Lawford, and W. Murray Wonham. Hierarchical interface-based supervisory control: AIP example. In *Proc. of 39th Annual Allerton Conference on Comm., Contr., and Comp.*, pages 396–405, Oct 2001.

[27] R.J. Leduc, B.A. Brandin, and W. Murray Wonham. Hierarchical interface-based nonblocking verification. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, pages 1–6, May 2000.

[28] R.J. Leduc, B.A. Brandin, W. Murray Wonham, and M. Lawford. Hierarchical interface-based supervisory control: Serial case. In *Proc. of 40th Conf. Decision Contr.*, pages 4116–4121, Orlando, USA, December 2001.

[29] R.J. Leduc, M. Lawford, and W. Murray Wonham. Hierarchical interface-based supervisory control, part II: Parallel case. Submitted to IEEE Trans. Automatic Control, Aug, 2003. An earlier version is available as Software Quality Research Laboratory Report No. 13, Dept.

of Computing and Software, McMaster University, Hamilton, ON. [ONLINE] Available: http://www.cas.mcmaster.ca/sqrl/sqrl_reports.html.

[30] R.J. Leduc, W. Murray Wonham, and M. Lawford. Hierarchical interface-based supervisory control: Parallel case. In *Proc. of 39th Annual Allerton Conference on Comm., Contr., and Comp.*, pages 386–395, Oct 2001.

[31] Ryan 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.

[32] Ryan Leduc. *Hierarchical Interface-based Supervisory Control.* PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 2002.

[33] Y. Li. *Control of Vector Discrete-Event Systems.* PhD thesis, Department of Electrical Engineering, University of Toronto, Toronto, Ont, 1991.

[34] F. Lin and W. Murray Wonham. Decentralized control and coordination of discrete-event systems with partial observations. In *Proc. 27th IEEE Conf. Decision Contr.*, pages 1125–1130, Dec 1988.

[35] Hong Liu, Jun-Cheol Park, and Raymond E. Miller. On hybrid synthesis for hierarchical structured petri nets. Technical report, Department of Computer Science, University of Maryland, College Park, MD, 1996.

[36] Chuan 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.

[37] E.M. Clarke M.C. Brown and O. Grümberg. Characterizing kripke structures in temporal logic. In G. Levi H. Erhig, R. Kowalski and U. Montanari, editors, *TAPSOFT'87, vol. I*, number 249 in LNCS, pages 256–270. Springer-Verlag, 1987.

[38] K.L. McMillan. *Symbolic Model Checking.* Kluwer, 1992.

[39] John O. Moody and Panos J. Antsaklis. *Supervisory Control of Discrete Event Systems using Petri Nets.* Kluwer Academic Publishers, 1998.

[40] J.S. Ostroff. *Temporal Logic for Real-Time Systems.* Research Studies Press/ Wiley, Taunton, UK, 1989.

[41] D. L. Parnas. Use of abstract interfaces in the development of software for embedded computer systems. NRL Report 8047, Naval Research Laboratory, 1977.

[42] David L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, December:1053–1058, December 1972.

[43] David Lorge Parnas, Paul C. Clements, and David M. Weiss. The modular structure of complex systems. *IEEE Transactions on Software Engineering*, SE-11(3):259–66, March 1985.

[44] Ken Qian 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.

[45] Robin G. Qiu and Sanjay 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*, 29(6):573–586, 1999.

[46] M.H. de Queiroz and J.E.R. Cury. Modular supervisory control of large scale discrete event systems. In *Proceedings of WODES 2000*, pages 103–110, Ghent, Belgium, Aug 2000.

[47] C. Costas R. Alur and D. Dill. Model-checking for real-time systems. In *Proc. of 5th IEEE Symp. Logic in Computer Science*, pages 414–425, 1990.

[48] K. Rudie. Software for the control of discrete-event systems: A complexity study. Master's thesis, Dept. of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1988.

[49] Karen Rudie and Jan C. Willems. The computational complexity of decentralized discrete-event control problems. *IEEE Trans. Automatic Control*, 440(7):1313–1319, 1995.

[50] Karen Rudie and W. Murray Wonham. Think globally, act locally: decentralized supervisory control. *IEEE Trans. on Automatic Control*, 37(11):1692–1708, Nov 1992. Reprinted in F.A. Sadjadi (Ed.), Selected Papers on Sensor and Data Fusion, 1996; ISBN 0-8194-2265-7.

[51] Gang Shen and Peter E. Caines. Hierarchically accelerated dynamic programming for finite-state machines. *IEEE Trans. Automatic Control*, 47(2):271–283, 2002.

[52] G. Stremersch and R.K. Boel. Decomposition of the supervisory control problem for Petri nets under preservation of maximal permissiveness. *IEEE Trans. Automatic Control*, 46(9):1490–1496, 2001.

[53] Bing Wang. Top-down design for RW supervisory control theory. Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1995.

[54] K.C. Wong. *Discrete-Event Control Architecture: An Algebraic Approach*. PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1994.

[55] K.C. Wong and J.H. van Schuppen. Decentralized supervisory control of discrete event systems with communication. In *Proc. of WODES 1996*, pages 284–289, Edinburgh, UK, Aug 1996.

[56] W. Murray Wonham. *Notes on Control of Discrete-Event Systems*. Department of Electrical and Computer Engineering, University of Toronto, 1999. Notes and CTCT software can be downloaded at http://odin.control.toronto.edu/DES/.

[57] Weimin Wu, Hongye Su, Jian Chu, and Haifeng Zhai. Hierarchical control of DES based on colored petri nets. In *Proc. of IEEE Systems, Man, and Cybernetics*, volume 3, pages 1571–1576, 2001.

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

[59] Z.H. 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.

[60] Z.H. Zhang and W. Murray Wonham. STCT: an efficient algorithm for supervisory control design. In *Proc. of SCODES 2001*, INRIA, Paris, July 2001.

[61] H. Zhong and W. Murray Wonham. On the consistency of hierarchical supervision in discrete-event systems. *IEEE Trans. on Automatic Control*, 35(10):1125–1134, Oct 1990.

[62] Meng Chu Zhou, David T. Wang, and Israel Mayk. Using petri nets for object-oriented design of command and control systems. *International Journal of Intelligent Control and Systems*, 2(2):287–300, 1998.

[63] MengChu Zhou and Frank DiCesare. *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Kluwer Academic Publishers, 1993.