Hierarchical Interface-Based Supervisory Control with Data Events

Ryan J. Leduc

***Abstract*— Hierarchical Interface-based Supervisory Control (HISC) decomposes a discrete-event system (DES) into a high-level subsystem which communicates with $n \geq 1$ low-level subsystems, through separate interfaces which restrict the interaction of the subsystems. It provides a set of local conditions that can be used to verify global conditions such as nonblocking and controllability. As each clause of the definition can be verified using a single subsystem, the complete system model never needs to be stored in memory, offering potentially significant savings in computational resources.**

In this paper, we extend the range of the behavior of low-levels that interfaces can model by adding a new type of event, low data events, and by relaxing some restrictions in the HISC definitions. This allows us to have (i) request events that don't need to be followed by an answer event, (ii) to start a low-level on a task and then poll it for completion, (iii) to be able to send additional commands while a low-level is already processing a command (iv) to model low-levels that behave as buffers, and (v) to allow unsolicited information (status etc.) to be sent up from a low-level.

Besides greatly enriching the behavior that can be modelled as interfaces and thus expanding the systems that HISC can effectively be applied to, the changes can enable behavior to be moved from the high-level to the low-levels. We demonstrate this when we discuss the application of our method to a large manufacturing system example based upon the AIP example, where we saw a 4.8 times reduction in computation time and a 6.5 times reduction in memory use. This helps prevent the high-level from growing too large, allowing the HISC method to apply to larger systems.

I. Introduction

In the area of Discrete-Event Systems (DES), two common tasks are to verify that a composite system, based on a Cartesian product of subsystems, is (i) nonblocking and (ii) controllable. The main obstacle to performing these tasks is the combinatorial explosion of the product state space.

The Hierarchical Interface-based Supervisory Control (HISC) framework was proposed by Leduc et al. ([1], [2], [3], [4]) to alleviate the state explosion problem. The HISC approach 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. It provides a set of local conditions that can be used to verify global conditions such as nonblocking and controllability. As each clause of the definition can be verified using a single subsystem, the complete system model never needs to be stored in memory, offering potentially significant savings in computational resources.

There has been some very promising recent work by Pena et al. [5], Flordal et al. [6], and Hill et al. [7] in using different forms of equivalence abstractions to verify flat systems in a modular fashion. These methods promise to increase the size of unstructured systems that we can verify. As individual levels (components) of an HISC system are treated as flat systems, these new results have the potential of being able to increase the individual component size that we can handle, if they can be adapted to verify the per component HISC conditions.

In this paper, we set out to extend the range of the behavior of low-levels that interfaces can model. This will have the affect of increasing the systems that HISC can be applied to as well as, in some cases, allow us to model certain low-levels in a more flexible, and intuitive way.

As we will see, extending the range of behavior we can model can have the side effect of allowing us to move behavior from the high-level to one or more low-levels, decreasing the size of the high-level. As the high-level (which includes the interfaces) typically increases in size as we add more low-levels, it is often the bottleneck limiting the size of systems we can handle with the HISC method. The new changes we are introducing can thus result in more natural HISC system design, as well as a decrease in computation time and memory usage.

The most important type of behavior we wish to be able to add, is the ability to start a low-level on a task and then poll it for completion. Currently, we can start the low-level on a task by sending it a command (*request event*), but then we must wait for it to complete the task and send us a response (*answer event*). Between the request event and answer event, we can not send any more commands to the low-level such as to add another task to its queue, or even to send it an abort command. Once the low-level has sent an answer event, it can't currently provide the high-level with any additional status information until after the next request event. That means the interface can't change state at this point in response to changes in the low-level state that would effect which request events are currently available, as well as which answer events can follow these request events.

The current inability of the low-level to provide status information between the occurrence of an answer event and a request event is the key limiting factor. If we started a machine on a task, and then polled it to see if the task is complete, we might expect to get two possible responses: *taskDone* or *notDone*. However, if the task completed at the low-level between the reception of the *notDone*, and before the request event to check completion, the interface would still be saying that the answer event *notDone* is possible, even though it is not. This would violate the current HISC

Ryan J. Leduc is with the Dept. of Computing and Software, McMaster University, Hamilton, ON, Canada, L8S 4K1 `leduc@mcmaster.ca`

interface consistency condition.

What we need is a new type of interface events, called *low data events*, that can occur in between a request and answer event, allowing the low-level to send status information as needed. This will allow the interface automata to stay in a consistent state. Low data events will actually be allowed to occur at any state in the interface, providing the additional ability of allowing the low-level to provide information to the high-level at will.

These new events will not only allow us to model polling behavior, but more importantly allow us to model low-levels that behave, from an input-output perspective, as buffers. This is important as buffers are a very common structure in systems. Being able to represent such low-levels using interfaces allows us to move the behavior of such subsystems to a low-level, while still interacting with the system as a buffer (i.e. adding tasks to the subsystems queue individually, and removing them individually once processed).

In addition to these new abilities, we would also like to have request events that don't need to be followed by an answer event. For instance, starting a task when using polling doesn't require a response, thus simplifying the interface automata. Finally, we would like to be able to send additional commands (additional information, an abort signal etc.) while a low-level is already processing a command.

In this paper, we first discuss DES preliminaries and then introduce the HISC approach in Section III. As the HISC method has already been explained and justified in detail in [2], [3] , and [4], we will present it quickly, focussing on our changes. We will discuss the new low data events, and the corresponding modifications to the HISC definitions to allow us to model the new types of behavior. We will then present our nonblocking and controllability results. Next we will discuss how to use the new method to model buffers, followed by an application to the AIP example from [1], [4].

II. DES Preliminaries

Supervisory control theory [8], [9], [10] 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 [10]. 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 $\overline{L} = \{t \in \Sigma^* \mid t \leq s \text{ for some } s \in L\}$. Let $\mathrm{Pwr}(\Sigma)$ denote the power set of Σ. For language L, the eligibility operator $\mathrm{Elig}_L : \Sigma^* \rightarrow \mathrm{Pwr}(\Sigma)$ is given by $\mathrm{Elig}_L(s) := \{\sigma \in \Sigma \mid s\sigma \in L\}$ 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 $\delta : Y \times \Sigma \rightarrow Y$ 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 $\delta : Y \times \Sigma^* \rightarrow Y$ 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. The coreachable state subset, denoted by Y_{cr}, is $Y_{cr} := \{y \in Y | \ (\exists s \in \Sigma^*)\, \delta(y, s) \in Y_m\}$. A DES is *coreachable* if $Y_{cr} = Y$.

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

$$\begin{aligned} P_i(\epsilon) &= \epsilon, \quad P_i(\sigma) = \left\{ \begin{array}{l} \epsilon \text{ if } \sigma \notin \Sigma_i \\ \sigma \text{ if } \sigma \in \Sigma_i \end{array} \right. \\ P_i(s\sigma) &= P_i(s)P_i(\sigma) \end{aligned}$$

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

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

where $P_i^{-1} : \mathrm{Pwr}(\Sigma_i^*) \rightarrow \mathrm{Pwr}(\Sigma^*)$ is the inverse image function of P_i.

The *synchronous product of DES* $\mathbf{G}_i = (Y_i, \Sigma_i, \delta_i, y_{o_i}, Y_{m_i})$ $(i = 1, 2)$, denoted $\mathbf{G}_1||\mathbf{G}_2$, is defined to be a reachable DES $\mathbf{G}$ with event set $\Sigma = \Sigma_1 \cup \Sigma_2$ and properties:

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

For DES, the two main properties we want to check are *nonblocking* and *controllability.* A DES $\mathbf{G}$ is said to be *nonblocking* if $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$. This is equivalent to saying that every reachable state is also coreachable.

For controllability, we assume the standard event partition $\Sigma = \Sigma_u \dot{\cup} \Sigma_c$, splitting our alphabet into *uncontrollable* and *controllable events*. To control a given plant $\mathbf{G}_1$, we define a *supervisor* represented as an automaton $\mathbf{S} = (X, \Sigma_S, \xi, x_o, X_m)$.

Definition 1: Let $\Sigma := \Sigma_1 \cup \Sigma_S, P_1 : \Sigma^* \rightarrow \Sigma_1^*$ and $P_S : \Sigma^* \rightarrow \Sigma_S^*$. 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, $(\forall s \in \mathcal{L}_{\mathbf{G}_1} \cap \mathcal{L}_{\mathbf{S}})\ \mathrm{Elig}_{\mathcal{L}_{\mathbf{G}_1}}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\mathcal{L}_{\mathbf{S}}}(s)$.

III. HISC with Low Data Events

A HISC system currently is a two-level system which includes one *high-level subsystem* and $n \geq 1$ *low-level subsystems*. The high-level subsystem communicates with each low-level subsystem through a separate *interface.*

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. This style of interaction is enforced by an interface that mediates communication between the two subsystems. All system components, including the interfaces, are modeled as automata.

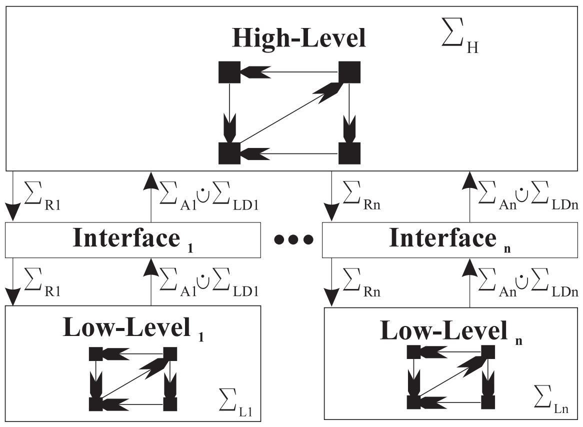


Fig. 1. Interface Block Diagram.

In order to restrict information flow and decouple the subsystems, the system alphabet is partitioned into pairwise disjoint alphabets:

$$\Sigma := \Sigma_H \,\dot{\cup} \dot{\bigcup_{j=1,\ldots,n}} [\Sigma_{L_j} \dot{\cup} \Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}] \qquad (1)$$

The events in Σ_H are called *high-level events* and the events in Σ_{L_j} are the j^{th} *low-level events* ($j = 1, \ldots, n$) as these events appear only in the high-level and j^{th} low-level subsystem models, $\mathbf{G}_H$ and $\mathbf{G}_{L_j}$ respectively. We then have $\mathbf{G}_H$ defined over event set $\Sigma_H \dot{\cup} (\dot{\cup}_{j \in \{1,\ldots,n\}} [\Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}])$ and $\mathbf{G}_{L_j}$ defined over event set $\Sigma_{L_j} \dot{\cup} \Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}$. We model the j^{th} interface by DES $\mathbf{G}_{I_j}$, which is defined over event set $\Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}$. We define our *flat system* to be $\mathbf{G} = \mathbf{G}_H || \mathbf{G}_{I_1} || \mathbf{G}_{L_1} || \ldots || \mathbf{G}_{I_n} || \mathbf{G}_{L_n}$. By flat system we mean the equivalent DES if we ignored the interface structure. For the remainder of this paper, the index j has range $\{1, \ldots, n\}$.

As the j^{th} *interface*, $\mathbf{G}_{I_j}$, is only concerned with communication between the two subsystems, it is defined over the events that are common to both levels of the hierarchy. The events in Σ_{R_j}, called *request events*, represent commands sent from the high-level subsystem to the j^{th} low-level subsystem. The events in Σ_{A_j} are *answer events* and represent the low-level subsystem's responses to the request events.

To these two event types present in the HISC formulation of [1], [2], [3], [4], we add a new event type called *low data events*. These events provide a means for a low-level to send information (data) through the interface, independent of the standard command-answer structure offered by the request and answer events. These new events are the key to keeping the state of the interfaces consistent with the state of their corresponding low-levels. Otherwise, the state of an interface can only be updated after the high-level sends a command down and the low-level responds with an answer event. This is insufficient to allow interfaces to represent polling (and thus buffer) behavior. Request, answer, and low data events are collectively known as the set of *interface events*, defined as $\Sigma_I := \dot{\cup}_{k \in \{1,\ldots,n\}} [\Sigma_{R_k} \dot{\cup} \Sigma_{A_k} \dot{\cup} \Sigma_{LD_k}]$.

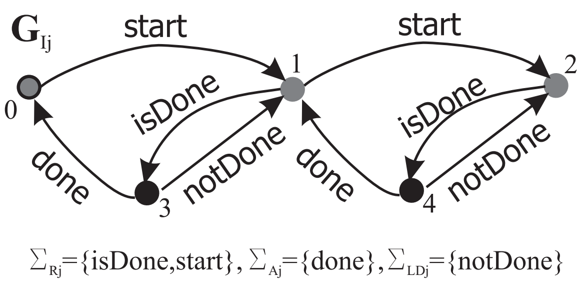
In order to enforce the serialization of requests and answers, we restrict the interfaces to the subclass of LD interfaces defined below. Our definition is based upon the state based command-pair interface definition of Leduc et al. [11], [12], which is equivalent to the definitions in [2], [3], [4]. Fig. 2 shows an example of a LD interface. It could correspond to a machine at the low-level with an effective internal buffer of two.

Definition 2: The j^{th} interface DES $\mathbf{G}_{\mathbf{I_j}} = (X_j, \Sigma_{I_j}, \xi_j, x_{o_j}, X_{m_j})$ is a *LD interface* if the following properties are satisfied:

1) $x_{o_j} \in X_{m_j}$
2) $(\forall x \in X_{m_j})(\forall \sigma \in \Sigma_{I_j})\, \xi_j(x, \sigma)! \Rightarrow [\sigma \in \Sigma_{R_j}] \vee [\sigma \in \Sigma_{LD_j} \wedge \xi_j(x, \sigma) \in X_{m_j}]$
3) $(\forall x \in X_j - X_{m_j})(\forall \sigma \in \Sigma_{I_j})\, \xi_j(x, \sigma)! \Rightarrow [\sigma \in \Sigma_{A_j} \wedge \xi_j(x, \sigma) \in X_{m_j}] \vee [\sigma \in \Sigma_{LD_j}]$

In the command-pair interface definition, *Point 2* would have only allowed request events to go to an unmarked state, requiring that they must be followed by an answer event. We allow request events to also go directly to a marked state and thus not requiring a succeeding answer event. Allowing this type of behavior can lead to smaller, more compact interfaces. This is possible because our proofs require that request events be only possible at marked states (allows us to force a low-level to a marked state even if the desired request event is disabled) and every answer event be preceded by a request event (so we can apply *Point 5* of Definition 3), not that every request event be followed by an answer event.

The other main change to the interface definition is the addition of the low-level data events. *Point 2* says low data events can occur at marked states, but they may only lead to another marked state. *Point 3* states that low data events may occur at unmarked states, and take us to either marked or unmarked states. These conditions are important because they ensure that the only way to get to a state where answer events are possible (an unmarked state), is via at least one request event.



Σ_{Rj}={isDone,start}, Σ_{Aj}={done},Σ_{LDj}={notDone}

Fig. 2. Example LD Interface 1.

To simplify notation in our exposition, we bring in the following event sets, natural projections, and languages. In particular, languages such as $\mathcal{H}$ represent the behavior of a given DES extended over Σ^*, as needed for the synchronous product.

$$\begin{array}{ll} \Sigma_{I_j} := \Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}, & P_{I_j} : \Sigma^* \to \Sigma^*_{I_j} \\ \Sigma_{IL_j} := \Sigma_{L_j} \cup \Sigma_{I_j}, & P_{IL_j} : \Sigma^* \to \Sigma^*_{IL_j} \\ \Sigma_{IH} := \Sigma_H \cup \bigcup_{k \in \{1,\ldots,n\}} \Sigma_{I_k}, & P_{IH} : \Sigma^* \to \Sigma^*_{IH} \\ \mathcal{H} := P^{-1}_{IH}(L(\mathbf{G}_H)), & \mathcal{H}_m := P^{-1}_{IH}(L_m(\mathbf{G}_H)) \subseteq \Sigma^* \\ \mathcal{L}_j := P^{-1}_{IL_j}(L(\mathbf{G}_{L_j})), & \mathcal{L}_{m_j} := P^{-1}_{IL_j}(L_m(\mathbf{G}_{L_j})) \subseteq \Sigma^* \\ \mathcal{I}_j := P^{-1}_{I_j}(L(\mathbf{G}_{I_j})), & \mathcal{I}_{m_j} := P^{-1}_{I_j}(L_m(\mathbf{G}_{I_j})) \subseteq \Sigma^* \\ \mathcal{I} := \cap_{k \in \{1,\ldots,n\}} \mathcal{I}_k, & \mathcal{I}_m := \cap_{k \in \{1,\ldots,n\}} \mathcal{I}_{m_k} \\ \Sigma_{LD} := \bigcup_{k \in \{1,\ldots,n\}} \Sigma_{LD_k} & \Sigma_{IHc} := \Sigma_{IH} - \Sigma_{LD} \end{array}$$

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

Definition 3: The n^{th} degree $(n \geq 1)$ 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 *LD interface consistent* with respect to the alphabet partition given by (1), if for all $j \in \{1, \ldots, n\}$, the following conditions are satisfied:

Multi-level 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 LD interface.

High-Level Property

3) $(\forall s \in \mathcal{H} \cap \mathcal{I})\ \text{Elig}_{\mathcal{I}_j}(s) \cap (\Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}) \subseteq \text{Elig}_{\mathcal{H}}(s)$

Low-Level Properties

4) $(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j)\ \text{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{R_j} \subseteq \text{Elig}_{\mathcal{L}_j}(s)$
5) $(\forall s \in \Sigma^*.\Sigma_{R_j} \cap \mathcal{L}_j \cap \mathcal{I}_j)$
$\text{Elig}_{\mathcal{L}_j \cap \mathcal{I}_j}(s\Sigma^*_{L_j}) \cap \Sigma_{A_j} = \text{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{A_j}$ where
$$\text{Elig}_{\mathcal{L}_j \cap \mathcal{I}_j}(s\Sigma^*_{L_j}) := \bigcup_{l \in \Sigma^*_{L_j}} \text{Elig}_{\mathcal{L}_j \cap \mathcal{I}_j}(sl)$$
6) $(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j)$
$s \in \mathcal{I}_{m_j} \Rightarrow (\exists l \in \Sigma^*_{L_j})\ sl \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$.

The only change from the interface consistency definition of [4], is in *Point 3*. Previously, it only referred to answer events. This addition ensures that if a low-level (which includes an interface) needs a low data event to occur to reach a marked state, the high-level subsystem must allow it. However, if the high-level requires a low data event to occur, there is no such guarantee. For a discussion of the remaining points, please see [1], [4].

A. 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 flat system $\mathbf{G}$ is to be nonblocking.

Definition 4: The n^{th} degree $(n \geq 1)$ 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 *LD level-wise nonblocking* if the following conditions are satisfied:

(I) *LD nonblocking at the high-level*:
$(\forall s \in \mathcal{H} \cap \mathcal{I})(\exists s' \in (\Sigma - \Sigma_{LD})^*)$
$ss' \in \mathcal{H}_m \cap \mathcal{I}_m$

(II) *nonblocking at the low-level*:
$(\forall j \in \{1, \ldots, n\})\ \overline{\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}} = \mathcal{L}_j \cap \mathcal{I}_j$

Like the level-wise nonblocking definition of [4], Definition 4 requires that the high-level and each low-level be individually nonblocking. However, we add the further restriction that each string in the high-level's closed behavior must be extendable to a marked string without using low data events. The reason for this is that although the high-level can react to the occurrence of low data events, it has no guarantee that a low-level will ever allow these events to occur. As a result, it must ensure that it can always return to a marked state without them. A useful rule of thumb is to make sure that for each reachable, unmarked state in each DES at the high-level (including the interfaces), there is at least one non low data event transition defined leading to a different state. For example, if we removed answer event *done* at state 3 in Fig. 2, then the only way to reach a marked state from state 3 would be via low data event *notDone*, thus violating *Point I* of Definition 4. This doesn't guarantee the system will be LD level-wise nonblocking, but will avoid many problems.

We have now presented all the changes to the HISC definitions of [4], required to model the new types of behavior discussed in Section I. It can be easily shown that if a n^{th} degree interface system satisfies the level-wise nonblocking and interface consistency definitions of [4], and contains no low data events (i.e. $\Sigma_{LD} = \emptyset$), then it follows that it is also LD level-wise nonblocking and LD interface consistent. The new definitions were difficult to develop since we required that they be backwards compatible, make as few changes as possible to the previous HISC definitions, allow for compact interfaces, and allow for significant reuse of existing proofs and software, while still allowing us to be able to model the desired new behaviors.

We now present our new results from [13]. Our first proposition says that for every string accepted by our system, we can always bring all the low-levels to a marked state via a string containing only low-level and interface events. This is useful as it ensures that after string sl, all interfaces are in a marked states. This means that any future extensions that contain answer events also contain a preceding request event, making the application of *Point 5* of our LD interface consistency definition straight forward.

Proposition 1: If the n^{th} degree $(n \geq 1)$ 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 LD level-wise nonblocking and LD interface consistent with respect to the alphabet partition given by (1), then

$(\forall s \in \mathcal{H} \cap [\cap_{j \in \{1,\ldots,n\}} (\mathcal{L}_j \cap \mathcal{I}_j)])$
$(\exists l \in \Sigma^*_{IL})\ (sl \in \mathcal{H} \cap [\cap_{j \in \{1,\ldots,n\}} (\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})])$

Proof: See proof in [13]. ■

Our next propositions says that for every string accepted by our system, we can always find a string containing only high-level and interface events but no low data events, that will bring the high-level to a marked state. However, string h may not be accepted by the low-levels.

Proposition 2: If the n^{th} degree $(n \geq 1)$ 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 LD level-wise nonblocking and LD interface consistent with respect to the alphabet partition given by (1), then

$(\forall s \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_j \cap \mathcal{I}_j)])$
$(\exists h \in \Sigma_{IHc}^*)\ sh \in \mathcal{H}_m \cap \mathcal{I}_m$

Proof: See proof in [13]. ∎

The next proposition says that given appropriate strings s and h, we can modify string h by adding appropriate low-level events (which are ignored by the high-level) into a string u, such that the flat system is in a marked state. It is key that h does not contain any low data events as there is no guarantee that we could get the low-levels to accept them.

Proposition 3: If the n^{th} degree $(n \geq 1)$ 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 LD interface consistent with respect to the alphabet partition given by (1), then

$(\forall s \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})])(\forall h \in \Sigma_{IHc}^*)$
$sh \in \mathcal{H}_m \cap \mathcal{I}_m \Rightarrow (\exists u \in \Sigma^*)\,(su \in \mathcal{H}_m \cap$
$[\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]) \wedge (P_{IH}(u) = h)$

Proof: See proof in [13]. ∎

This brings us to our main nonblocking result.

Theorem 1: If the n^{th} degree $(n \geq 1)$ 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 LD level-wise nonblocking and LD interface consistent with respect to the alphabet partition given by (1), then

$L(G) = \overline{L_m(G)}$ where $G = G_H || G_{L_1} || G_{I_1} || \ldots || G_{L_n} || G_{I_n}$

Proof:

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

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

Let $s \in L(G) = \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_j \cap \mathcal{I}_j)]$ **(2)**

Must show: $(\exists u \in \Sigma^*)\ su \in L_m(G) = \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]$

Applying *Proposition 1*, we can conclude:

$(\exists l \in \Sigma_{IL}^*)\ sl \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]$ **(3)**

Applying *Proposition 2*, taking sl to be string s in said proposition, we can conclude:

$(\exists h \in \Sigma_{IHc}^*)\ (slh \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}\mathcal{I}_{m_j}])$ **(4)**

Combining with **(3)**, we can now apply *Proposition 3*, taking sl to be string s in said proposition, and conclude:

$(\exists u' \in \Sigma^*)\,(slu' \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]$

We take string $u = lu'$ and we have $su \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})] = L_m(G)$, as required. ∎

As verifying the LD level-wise nonblocking and LD interface consistent conditions only require a single level at a time, we can evaluate each level independently. We thus do not need to construct the entire system model, potentially significantly reducing the computational resources required.

B. Local Conditions for Global Controllability of the System

For controllability, we need to split the subsystems into their plant 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 j^{th} *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 j^{th} 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:

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

We now provide the controllability requirements that each level must satisfy.

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

(I) 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}

(II) $(\forall s \in \mathcal{L}_j^p \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j)\quad \text{Elig}_{\mathcal{L}_j^p}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s)$

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

The above definition is identical to the level-wise controllability definition of [4], except that the system is defined over an event set that can also contain low data events. For a discussion of the individual points, please see [1], [4].

We now present our new controllability results from [13]. Our first proposition says that the j^{th} low-level supervisor, in conjunction with the j^{th} interface, is controllable for the flat plant.

Proposition 4: If n^{th} degree $(n \geq 1)$ interface system composed of DES $\mathbf{G}_H^p, \mathbf{S}_H, \mathbf{G}_{L_1}^p, \mathbf{S}_{L_1}, \mathbf{G}_{I_1}, \ldots, \mathbf{G}_{L_n}^p, \mathbf{S}_{L_n}, \mathbf{G}_{I_n}$ is LD level-wise controllable with respect to the alphabet partition given by (1), then

$(\forall j \in \{1, \ldots, n\})\,(\forall s \in L(\textbf{Plant}) \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j)$
$\text{Elig}_{L(\textbf{Plant})}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s)$

Proof: See proof in [13]. ∎

Our next proposition says that the high-level supervisor is controllable for the flat plant, when the plant is already under the control of the system's interfaces.

Proposition 5: If n^{th} degree $(n \geq 1)$ interface system composed of DES $\mathbf{G}_H^p, \mathbf{S}_H, \mathbf{G}_{L_1}^p, \mathbf{S}_{L_1}, \mathbf{G}_{I_1}, \ldots, \mathbf{G}_{L_n}^p, \mathbf{S}_{L_n}, \mathbf{G}_{I_n}$ is LD level-wise controllable with respect to the alphabet partition given by (1), then

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

Proof: See proof in [13]. ∎

This brings us to our main controllability result.

Theorem 2: If n^{th} degree $(n \geq 1)$ interface system composed of DES $\mathbf{G}_H^p, \mathbf{S}_H, \mathbf{G}_{L_1}^p, \mathbf{S}_{L_1}, \mathbf{G}_{I_1}, \ldots, \mathbf{G}_{L_n}^p, \mathbf{S}_{L_n}, \mathbf{G}_{I_n}$

is LD level-wise controllable with respect to the alphabet partition given by (1), then
$(\forall s \in L(\textbf{Plant}) \cap L(\textbf{Sup}))$
$\quad \text{Elig}_{L(\textbf{Plant})}(s) \cap \Sigma_u \subseteq \text{Elig}_{L(\textbf{Sup})}(s)$

Proof:

Assume system is LD level-wise controllable. **(1)**

Let $s \in L(\textbf{Plant}) \cap L(\textbf{Sup})$, $\sigma \in \text{Elig}_{L(\textbf{Plant})}(s) \cap \Sigma_u$ **(2)**

Must show $s\sigma \in L(\textbf{Sup}) = \mathcal{S}_H \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{S}_{L_j} \cap \mathcal{I}_j)]$

We first apply *Proposition 4* and conclude:
$\quad s\sigma \in \cap_{j\in\{1,\ldots,n\}}(\mathcal{S}_{L_j} \cap \mathcal{I}_j)$ **(3)**

Combining with **(2)**, we have $\sigma \in \text{Elig}_{L(\textbf{Plant})\cap\mathcal{I}}(s) \cap \Sigma_u$

Applying *Proposition 5*, we can conclude $s\sigma \in \mathcal{S}_H$

Combining with **(3)**, gives $s\sigma \in \mathcal{S}_H \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{S}_{L_j} \cap \mathcal{I}_j)]$, as required.

■

As verifying the LD level-wise controllable condition only requires a single level at a time, we can evaluate each level independently. We again do not need to construct the entire system model.

IV. Representing Buffers as Interfaces

Now that we have defined the new low data HISC conditions, we will discuss how to use interfaces to represent low-levels that behave as buffers. It proved to be tricky to figure out how to do this in a way that produced compact interfaces, yet did not violate *Point 5* of the LD interface consistency definition. *Point 5* says that immediately after a request event occurs, there must exist a path containing only low-level events to each answer event that can follow said request event.

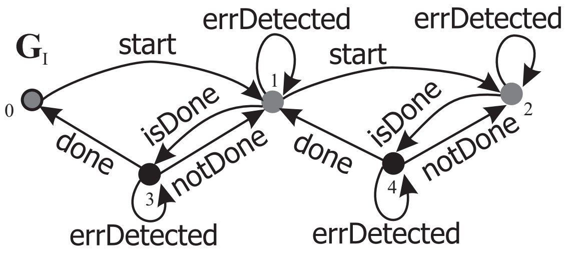
If we are only interested in starting tasks and determining when the task is complete, we can use the approach shown in Fig. 2 which represents a two slot buffer. Request event *start* is used to add a task to the low-level's buffer. As we will start the task and then poll to determine completion, we don't need a matching answer event; thus event *start* takes us directly to a marked state (gray circle). We can then either add another task or use request event *isDone* to check to see if the task is complete.

Done must be an answer event so that we can be assured that it will eventually occur, and so that the high-level can require that it occur in order to reach a marked state. If we made *done* a low data event, we would have problems with *Point I* of the LD level-wise nonblocking definition.

Response events that represent behavior that is not required to complete a task (i.e. error events, notDone etc.) should be made low data events as they typically don't need to occur to reach a marked state. Accordingly, we make *notDone* a low data event. This is key to passing *Point 5* of the LD interface consistency definition. If we made *notDone* an answer event and the low-level finished its task while the interface was in state 1 or 2, then *notDone* would no longer be possible at state 3. This would cause *Point 5* of the LD interface consistency definition to fail. However, if *notDone* is a low data event, it is not subject to *Point 5* and need never occur as the high-level can always return to a marked state without it.

The approach shown in Fig. 2 works well if we only wish to report that the task is complete. If we wish to also report that an error occurred, for instance, things are more complicated. If we added a low data event, *error*, from state 3 to 0, this would cause event *done* to fail *Point 5* of the LD interface consistency definition if the low-level determined an error occurred while the interface was at state 1. The reason is that event *done* would no longer be possible at state 3.

We can use the approach shown in Fig. 3 to handle this situation. Here we have added a new low data event, *errDetected*, and selflooped it at states 1–4. As the occurrence of *errDetected* does not preempt the occurrence of answer event *done*, all is well. The high-level can detect the occurrence of *errDetected* and change state so that it will take the appropriate action once event *done* finally occurs. There is one caveat: event *errDetected* must be the first point at which the low-level knows that an error has occurred (i.e. up until *errDetected* occurs, it must have been still possible that the task could have completed without error). If this is not the case, then it is possible that once the interface reaches state 3 or 4, all paths to answer event *done* contain the low data event *errDetected*, violating *Point 5* of the LD interface consistency definition. As can be seen from Fig. 3, the new low data HISC conditions allow buffers to be modelled in a very compact manner, minimizing the increase in complexity of the high-level.



Σ_R={isDone,start}, Σ_A={done}, Σ_{LD}={notDone,errDetected}

Fig. 3. Example LD Interface With Error Event.

V. Verifying Properties

The LD interface definition can be verified using the command-pair interface algorithm from [11] with straight forward modifications. The LD level-wise controllability definition as well as *Point II* of the LD level-wise nonblocking definition can be verified using existing supervisory control algorithms after suitable definitions have been made. For *Point I* of the LD level-wise nonblocking definition, we can use a standard nonblocking algorithm modified to ignore low data events when it checks coreachability.

For *Point 3* of the LD interface consistency definition, we can use standard controllability algorithms such as TCT's **condat** function [10]. We simply define $\textbf{ThePlant} = \mathbf{G}_{I_j}$, $\textbf{TheSpec} = \mathbf{G}_H$, $\Sigma_u = \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}$, and $\Sigma_c = \Sigma - \Sigma_u$. The remaining conditions of the LD interface consistency definition can be verified using the interface consistency algorithm in [11].

VI. AIP Example

To demonstrate the utility of our method, we apply it to a large manufacturing system, the Atelier Inter-établissement de Productique (AIP). The AIP was originally investigated by Charbonnier et al. in [14], [15], by Leduc et al. in [1], [4] using the HISC method, and finally by Ma et al. in [16] using state tree structures and binary decision diagrams. The AIP, shown in Fig 4, is an 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 transport 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.

In the original HISC design from [1], [4], the system consisted of a high-level and seven low-levels; one for each transport unit and assembly station. Also, only one pallet at a time was allowed on external loops 1 and 2. For the full control specifications, see [1], [4].

In our new extended version, we wanted to allow two pallets at a time in external loops 1 and 2 (which should significantly increase the size of the original design's high-level), and move the details of these loops (including the assembly stations) out of the high-level. To achieve this, we merged low-levels AS1 and TU1 into a new low-level called EL1. Similarly, AS2 and TU2 became EL2. We now have only five low-levels.

Fig. 5 shows the interface for an EL low-level ($(k, r) = (\mathrm{AS1}, \mathrm{TU1})$ for EL1, and $(k, r) = (\mathrm{AS2}, \mathrm{TU2})$ for EL2). Here, uncontrollable events are shown in italics; all other events are controllable. Initial states can be recognized by a thick outline, and marked states are filled.

In essence, the interface represents a two slot buffer (states s0-s5, s9, s10) where pallets are moved from the central loop to the external loop via request-answer event pair *TrnsfToEL-TrnsfCplToEL*, processed internally by the assembly station, and then returned to the central loop via request-answer event pair *TrnsfELToCL-TrnsfCplToCL*. Request event *SkipTrnsfCL*, selflooped at states s0, s6, s11, is used by the high-level to bypass attempts to execute event *TrnsfELToCL* as it would cause blocking at that point. At state s5, the EL is full so request-answer event pair *LibPallet-palletRlsd* allows pallets on the central loop to pass through the TU and continue moving around the CL.

At states s2 and s5, we are polling to see if a pallet on the external loop has been processed by the assembly station and is now waiting to be transferred back to the central loop. If not, low data event *noTrnsfCL* occurs and we keep waiting. Otherwise, the pallet is transferred to the CL.

States s6-s8 and s11-s14 handle the case when the assembly station is down. Low data event, *Rtimeout*, signals that the AS is down, and causes the interface to change to a new behavior where transfers to and from the external loop are suspended until the AS is repaired. Request-answer event pair *LibPallet-palletRlsd* allows pallets on the central loop to pass through the TU and continue moving around the CL. We use request event *ASUpYet* to poll if the AS is up yet, where answer event *RobUp* and low data event *NotASUp* provide the responses. For the remainder of the design details, including the more than 170 other automata, please refer to [13].

After the design was completed, we applied our software to the system and determined that it was LD level-wise nonblocking and controllable, and LD interface consistent. We can thus apply *Theorems 1* and *2* and conclude that the flat system is nonblocking and that the flat supervisor is controllable for the flat plant. The computation took 104 seconds, and required 117MB. Detailed results are shown in Table I, including the statespace of each level. This example

TABLE I
Low Data AIP Results

Subsystem	States	
	$\|\|$ with $\mathbf{G}_{I_j}$	Size of $\mathbf{G}_{I_j}$
$\mathbf{G}_H$	518,400	7,200
EL1, **EL2**	30,185	15
AS3	203	2
TU3	204	4
TU4	152	4

has an estimated closed-loop statespace of 2.97×10^{21}. This estimate was calculated by determining the closed-loop statespace of the high-level and each low-level, and then multiplying these together to create a worst case state estimate. It's quite likely that the actual system will be considerably smaller.

On the same computer, we also verified that the AIP example from [1], [4] was level-wise nonblocking and controllable, and interface consistent. The computation took 356.76 seconds, and required 760MB. Table II shows the

TABLE II
AIP Results

Subsystem	States	
	$\|\|$ with $\mathbf{G}_{I_j}$	Size of $\mathbf{G}_{I_j}$
$\mathbf{G}_H$	3,306,240	8,192
AS1, **AS2**	120	4
AS3	203	2
TU1, **TU2**	98	4
TU3	204	4
TU4	152	4

detailed results. As we can see, because the new low Data HISC method allowed us to shift behavior to the low-levels, we were able to decrease the statespace of the high-level by a factor of 6.4. This resulted in a 3.4 times computation speedup, and a 6.5 times reduction in memory. We believe that applying this approach to much larger examples will produce even more dramatic savings.

VII. Conclusions

Hierarchical interface-based supervisory control (HISC) offers an effective method to model systems with a natural

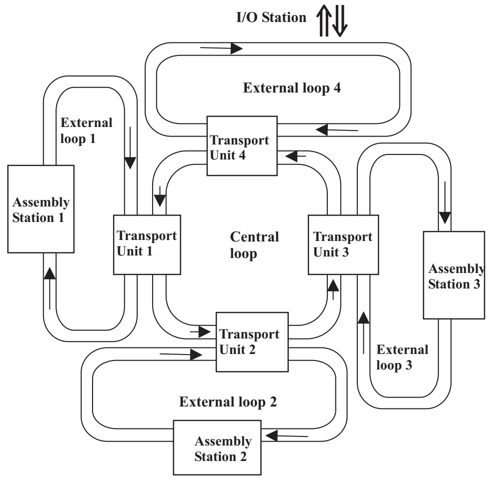


Fig. 4. The Atelier Inter-établissement de Productique

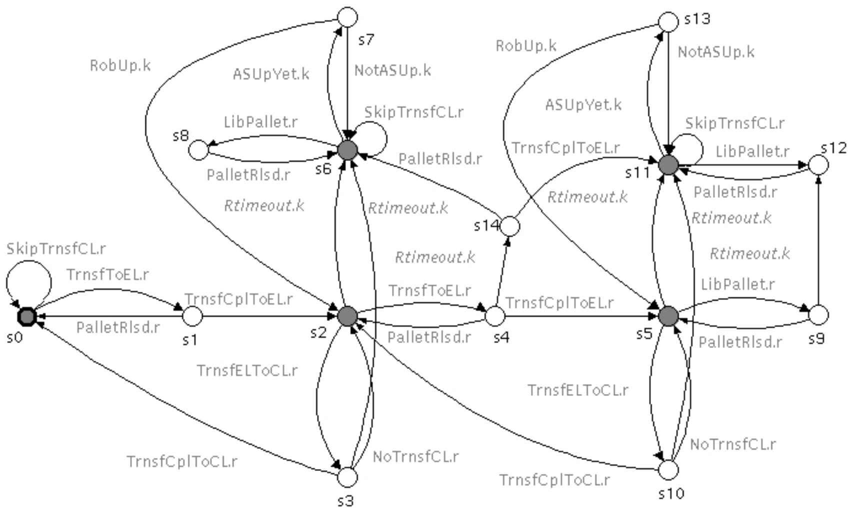


Fig. 5. EL Interface

client-server architecture. As each requirement can be verified using only one subsystem, the entire plant model never needs to be constructed or traversed, offering potentially significant savings in computation.

In this paper, we extend the range of the behavior of low-levels that interfaces can model by adding a new type of event, low data events, and by relaxing some restrictions in the HISC definitions. These new changes are backwards compatible, allow for compact interfaces, and permit significant reuse of existing proofs, and software.

These changes allow us to give low-levels more freedom in sending information to the high-level, allow us to model polling behavior, and most importantly, allow us to model low-levels that behave as buffers in a natural, intuitive manner. This not only lets us greatly enrich the behaviors we can model, but enables us to move more behavior from the high-level to low-levels. Reducing the complexity of the high-level increases the size of the systems we can apply HISC to since the statesize of the high-level grows as we add more low-levels, becoming a limiting factor.

This paper also offers a tutorial on how to model buffers using interfaces. As determining how to do this in an intuitive, compact manner was nontrivial, this provides an invaluable resource for using the new approach.

Finally, an application of our method to a large manufacturing system (estimated worst case statespace on order of 10^{21}) based on the AIP example shows how we can apply the method to a real system and reduce the state size of the high-level by a factor of 6.4. This resulted in a 3.4 computation speedup, and a 6.5 times reduction in memory just from moving behavior out of the high-level.

References

[1] R. Leduc, "Hierarchical interface-based supervisory control," Ph.D. dissertation, Department of Elec. and Comp. Engineering, University of Toronto, Toronto, Ont, 2002, [ONLINE] Available: http://www.cas.mcmaster.ca/~leduc.

[2] R. J. Leduc, B. A. Brandin, M. Lawford, and W. M. Wonham, "Hierarchical interface-based supervisory control, part I: Serial case," *IEEE Trans. Automatic Control*, vol. 50, no. 9, pp. 1322–1335, Sept. 2005.

[3] R. J. Leduc, M. Lawford, and W. M. Wonham, "Hierarchical interface-based supervisory control, part II: Parallel case," *IEEE Trans. Automatic Control*, vol. 50, no. 9, pp. 1336–1348, Sept. 2005.

[4] R. J. Leduc, M. Lawford, and P. Dai, "Hierarchical interface-based supervisory control of a flexible manufacturing system," *IEEE Trans. on Control Systems Technology*, vol. 14, no. 4, pp. 654–668, July 2006.

[5] P. Pena, J. Cury, and S. Lafortune, "Testing modularity of local supervisors: An approach based on abstractions," in *Proc. of WODES 2006*, Ann Arbor, USA, Jul. 2006, pp. 107–112.

[6] H. Flordal and R. Malik, "Modular nonblocking verification using conflict equivalence," in *Proc. of WODES 2006*, Ann Arbor, USA, Jul. 2006, pp. 100–106.

[7] R. Hill and D. Tilbury, "Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction," in *Proc. of WODES 2006*, Ann Arbor, USA, Jul. 2006, pp. 399–406.

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

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

[10] W. M. Wonham, *Supervisory Control of Discrete-Event Systems*, Department of Electrical and Computer Engineering, University of Toronto, July 2006, Monograph and TCT software can be downloaded at http://www.control.toronto.edu/DES/.

[11] P. Dai, "Synthesis method for hierarchical interface-based supervisory control," Master's thesis, Dept. of Computing and Software, McMaster University, Hamilton, Ont, 2006, [ONLINE] http://www.cas.mcmaster.ca/~leduc/#studtheses.

[12] R. J. Leduc and P. Dai, "Synthesis method for hierarchical interface-based supervisory control," in *Proc. of 26th American Control Conference*, New York City, USA, July 2007, pp. 4260–4267.

[13] R. J. Leduc, "Hierarchical interface-based supervisory control with data events," Software Quality Research Laboratory, Dept. of Computing and Software, McMaster University, Hamilton, ON, Canada, Tech. Rep. No. 44, 2007, [ONLINE] Available: http://www.cas.mcmaster.ca/sqrl/sqrl_reports.html.

[14] B. Brandin and F. Charbonnier, "The supervisory control of the automated manufacturing system of the AIP," in *Proc. Rensselaer's 1994 4th International Conf. on Comput. Integr. Mfg and Automation Technol.*, Troy, Oct 1994, pp. 319–324.

[15] F. Charbonnier, "Commande par supervision des systèmes à événements discrets: application à un site expérimental l'Atelier Inter-établissement de Productique," Laboratoire d'Automatique de Grenoble, Grenoble, France, Tech. Rep., 1994.

[16] C. Ma and W. M. Wonham, *Nonblocking Supervisory Control of State Tree Structures*, ser. Lecture Notes in Control and Information Sciences. Berlin, Germany: Springer-Verlag, 2005, vol. 317, a1: 20010101.