Hierarchical Interface-Based Supervisory Control Using the Conflict Preorder

Robi Malik [*] **Ryan Leduc** [**]

[*] *Department of Computer Science, University of Waikato, Hamilton, New Zealand, (e-mail: robi@waikato.ac.nz)*
[**] *Department of Computing and Software, McMaster University, Hamilton, Canada, (e-mail: leduc@mcmaster.ca)*

Abstract: Hierarchical Interface-Based Supervisory Control decomposes a large discrete event system into subsystems linked to each other by interfaces, facilitating the design of complex systems and the re-use of components. By ensuring that each subsystem satisfies its interface consistency conditions locally, it can be ensured that the complete system is controllable and nonblocking. The interface consistency conditions proposed in this paper are based on the conflict preorder, providing increased flexibility over previous approaches. The framework requires only a small number of interface consistency conditions, and allows for the design of multi-level hierarchies that are provably controllable and nonblocking.

Keywords: Discrete event systems, large-scale systems, hierarchical control, verification.

1. INTRODUCTION

The framework of *Hierarchical Interface-based Supervisory Control (HISC)* (Leduc, 2002; Leduc *et al.*, 2005) facilitates the design of large discrete event systems, by decomposing them into *subsystems* communicating with each other through *interfaces.* This structure improves re-use and documentation of subsystems. In addition, global conditions such as controllability and nonblocking can be verified by only checking local *interface consistency* conditions for each subsystem. The complete system model never needs to be constructed, offering potentially significant savings in computational effort.

Most current HISC frameworks are based on a master-slave relationship between subsystems, with *high-level* subsystems sending *requests* to *low-level* subsystems and waiting for *answers* to come back (Leduc, 2002; Leduc *et al.*, 2005). Expressiveness is increased by the addition of *low data* events (Leduc, 2009). The framework has also been extended to support synthesis of least restrictive subsystem controllers (Leduc *et al.*, 2009) and multi-level hierarchies (Hill *et al.*, 2010).

While the master-slave structure is common in software, it is not always appropriate for reactive and control systems. Other methods use *natural projection* to obtain abstractions of subsystems (Lin and Wonham, 1990). This allows for more flexible structure, but projection-based methods are limited by the fact that natural projection does not preserve the often crucial nonblocking property, unless additional constraints are imposed (Wong and Wonham, 1996).

This paper proposes *Hierarchical Interface-based Supervisory Control using the Conflict Preorder (HISC-CP)*, which does not assume a strict master-slave relationship. Based on results about the *conflict preorder* (Malik *et al.*, 2006), interfaces are conflict-preserving abstractions of the subsystem they represent. The limitations of natural projection are avoided by the use of nondeterministic interfaces. The resulting interface consistency conditions facilitate the design of better hierarchies by allowing more flexibility in the construction of interfaces and subsystems.

This paper is organised as follows. Sect. 2 introduces the background of discrete event systems and the conflict preorder. This is followed in Sect. 3 by the description of HISC-CP. Afterwards, Sect. 4 presents some example interfaces for a large manufacturing system, and Sect. 5 concludes by comparing the approach to previous work.

2. PRELIMINARIES

2.1 Events and Traces

Event sequences and languages are a simple means to describe the behaviour of discrete event systems. Their building blocks are *events*, which are taken from a finite *alphabet* Σ. For supervisory control, Σ is partitioned into the set Σ_c of *controllable* events and the set Σ_u of *uncontrollable* events. In addition, the *silent event* $\tau \notin \Sigma$ is used, with the notation $\Sigma'_\tau = \Sigma' \cup \{\tau\}$ for any alphabet $\Sigma' \subseteq \Sigma$. Σ^* denotes the set of all finite *traces* of the form $\sigma_1\sigma_2 \cdots \sigma_n$ of events from Σ, including the *empty trace* ε. A subset $\mathcal{L} \subseteq \Sigma^*$ is called a *language*. The *concatenation* of two traces $s, t \in \Sigma^*$ is written as st. Trace $s \in \Sigma^*$ is a *prefix* of $t \in \Sigma^*$, written $s \sqsubseteq t$, if $t = su$ for some $u \in \Sigma^*$. For $\Omega \subseteq \Sigma$, the *natural projection* $P_\Omega \colon \Sigma^* \to \Omega^*$ removes from traces $s \in \Sigma^*$ all events not in Ω.

2.2 Nondeterministic Automata

System behaviours are modelled using finite-state automata. Supervisors are usually deterministic, but plant models and interfaces may be nondeterministic.

Definition 1. A (nondeterministic) *finite-state automaton* is a 5-tuple $G = \langle \Sigma_G, Q, \to, Q^\circ, Q^\omega \rangle$ where $\Sigma_G \subseteq \Sigma$ is the *automaton alphabet*, Q is a finite set of *states*, $\to\ \subseteq Q \times \Sigma_{G,\tau} \times Q$ is the *transition relation*, $Q^\circ \subseteq Q$ is the set of *initial states*, and $Q^\omega \subseteq Q$ is the set of *marked* or *terminal states*.

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$. It is also defined for $\upsilon \notin \Sigma_{G,\tau}$ by letting $x \xrightarrow{\upsilon} x$ for all states $x \in Q$. The transition relation is further extended to traces in Σ_τ^* by letting $x \xrightarrow{\varepsilon} x$ for all $x \in Q$, and $x \xrightarrow{s\sigma} y$ if $x \xrightarrow{s} z \xrightarrow{\sigma} y$ for some $z \in Q$. For brevity, $x \overset{s}{\Rightarrow} y$, with $s \in \Sigma^*$, denotes the existence of a trace $t \in \Sigma_\tau^*$ such that $x \xrightarrow{t} y$ and $P_\Sigma(t) = s$. That is, $\xrightarrow{s}$ denotes a path with *exactly* the events in s, while $\overset{s}{\Rightarrow}$ denotes a path with an arbitrary number of τ shuffled with the events in s.

For state sets $X, Y \subseteq Q$, the expression $X \xrightarrow{s} Y$ denotes the existence of $x \in X$ and $y \in Y$ such that $x \xrightarrow{s} y$. Furthermore, $x \rightarrow y$ denotes the existence of $s \in \Sigma^*$ such that $x \xrightarrow{s} y$, and $x \xrightarrow{s}$ denotes the existence of $y \in Q$ such that $x \xrightarrow{s} y$, and $G \xrightarrow{s} x$ stands for $Q^\circ \xrightarrow{s} x$, etc. The same notations are introduced for $\Rightarrow$.

The prefix-closed *language* of the automaton G is $\mathcal{L}(G) = \{\, s \in \Sigma^* \mid G \overset{s}{\Rightarrow} \,\}$. Note that this is defined over the complete alphabet Σ, not just the automaton alphabet Σ_G.

When two automata are running in parallel, lock-step synchronisation in the style of (Hoare, 1985) is used.

Definition 2. Let $G = \langle \Sigma_G, Q_G, \rightarrow_G, Q_G^\circ, Q_G^\omega \rangle$ and $H = \langle \Sigma_H, Q_H, \rightarrow_H, Q_H^\circ, Q_H^\omega \rangle$ be two automata. Then the *synchronous composition* of G and H is

$$G \,\|\, H = \langle Q_G \times Q_H, \Sigma_G \cup \Sigma_H, \rightarrow, Q_G^\circ \times Q_H^\circ, Q_G^\omega \times Q_H^\omega \rangle \quad (1)$$

where

$$\begin{aligned}
(x,y) &\xrightarrow{\sigma} (x',y') && \text{if } \sigma \in (\Sigma_G \cap \Sigma_H),\ x \xrightarrow{\sigma}_G x',\ y \xrightarrow{\sigma}_H y'\,;\\
(x,y) &\xrightarrow{\sigma} (x',y) && \text{if } \sigma \in (\Sigma_G \setminus \Sigma_H) \cup \{\tau\},\ x \xrightarrow{\sigma}_G x'\,;\\
(x,y) &\xrightarrow{\sigma} (x,y') && \text{if } \sigma \in (\Sigma_H \setminus \Sigma_G) \cup \{\tau\},\ y \xrightarrow{\sigma}_H y'\,.
\end{aligned}$$

In synchronous composition, shared events must be executed by all automata synchronously, while other events (including τ) are executed independently.

Hiding is the act of replacing certain events by the silent event τ. This is a simple form of abstraction that in general introduces nondeterminism.

Definition 3. Let $G = \langle \Sigma_G, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ and $\Upsilon \subseteq \Sigma$. The result of *hiding* Υ from G, written $G \setminus \Upsilon$, is the automaton obtained from G by replacing each transition $x \xrightarrow{\upsilon} y$ with $\upsilon \in \Upsilon$ by $x \xrightarrow{\tau} y$, and removing all events in Υ from Σ_G.

2.3 Supervisory Control

Given *plant* and *specification* automata, *supervisory control theory* (Ramadge and Wonham, 1989) allows one to design a *supervisor* that restricts the plant behaviour such that the specification is fulfilled. The key requirements for such supervisors are *controllability* and *nonblocking*.

Definition 4. Specification $K = \langle \Sigma_K, Q_K, \rightarrow_K, Q_K^\circ, Q_K^\omega \rangle$ is *controllable* with respect to plant $G = \langle \Sigma_G, Q_G, \rightarrow_G, Q_G^\circ, Q_G^\omega \rangle$ if, for every trace $s \in \Sigma^*$, every state $x \in Q_K$, and every uncontrollable event $\upsilon \in \Sigma_u$ such that $K \overset{s}{\Rightarrow}_K x$ and $G \overset{s\upsilon}{\Rightarrow}_G$, it holds that $x \xrightarrow{\upsilon}_K$.

Definition 5. An automaton $G = \langle \Sigma_G, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ is *nonblocking* if, for every state $x \in Q$ such that $G \rightarrow x$, it holds that $x \rightarrow Q^\omega$; otherwise G is *blocking*. Two automata G and H are *nonconflicting* if $G \,\|\, H$ is nonblocking.

Controllability essentially represents safety properties, while nonblocking or nonconflicting is the weak liveness property underlying supervisory control theory. A major challenge in supervisory control is to ensure that large systems remain nonconflicting.

2.4 The Conflict Preorder

Conflict equivalence (Malik *et al.*, 2006) provides a means to reason about conflicts in a compositional way. According to process-algebraic testing theory, two automata are considered as equivalent if they both respond in the same way to all tests of a certain type (De Nicola and Hennessy, 1984). Here, a *test* is an arbitrary automaton, and the *response* is the observation whether or not the test and the automaton in question are nonconflicting.

Definition 6. (Malik *et al.*, 2006) Automaton G is *less conflicting* than automaton H, written $G \lesssim_{\text{conf}} H$, if for any automaton T such that $H \,\|\, T$ is nonblocking, $G \,\|\, T$ also is nonblocking. G and H are *conflict equivalent*, written $G \simeq_{\text{conf}} H$, if $G \lesssim_{\text{conf}} H$ and $H \lesssim_{\text{conf}} G$.

The *conflict preorder* $\lesssim_{\text{conf}}$ is a *congruence* with respect to synchronous composition and hiding, i.e., it is always preserved under these operations.

Proposition 1. (Malik *et al.*, 2006) The following conditions hold for all automata $G \lesssim_{\text{conf}} H$.

(i) $G \,\|\, T \lesssim_{\text{conf}} H \,\|\, T$ for every automaton T.
(ii) $G \setminus \Upsilon \lesssim_{\text{conf}} H \setminus \Upsilon$ for every $\Upsilon \subseteq \Sigma$.

In fact, the conflict preorder is the coarsest nonblocking-preserving preorder with these congruence properties. It is the best possible preorder for compositional reasoning about nonblocking (Malik *et al.*, 2006). Therefore, conflict equivalence is used for efficient compositional verification of the nonblocking property (Flordal and Malik, 2009).

Every automaton can be associated with a language of *certain conflicts*, which contains all traces that, when possible in the environment, necessarily cause blocking.

Definition 7. (Malik, 2010) The *set of certain conflicts* of automaton G is

$$\begin{aligned}\textsc{Conf}(G) = \{\, s \in \Sigma^* \mid{}& \text{for every automaton } T,\\ & \text{if } T \overset{s}{\Rightarrow} \text{ then } G \,\|\, T \text{ is blocking} \,\}\,.\end{aligned} \quad (2)$$

If G is nonblocking, then clearly $\textsc{Conf}(G) = \emptyset$. However, the set of certain conflicts is not necessarily a subset of the language of the automaton (Malik *et al.*, 2006). Certain conflicts are closed under extension, because whenever the possibility to execute a trace s leads to blocking, then this also holds when an extension st is possible. Conversely, every trace s of certain conflicts has an *explanation* in the language of its automaton G, i.e., a prefix $r \sqsubseteq s$ accepted by G that is a certain conflict.

Lemma 2. (Malik, 2010) Let G be an automaton. Then $\textsc{Conf}(G) = \textsc{Conf}(G)\Sigma^*$.

Lemma 3. (Malik, 2010) Let G be an automaton, and let $s \in \textsc{Conf}(G)$. Then there exists a prefix $r \sqsubseteq s$ such that $r \in \mathcal{L}(G) \cap \textsc{Conf}(G)$.

Certain conflicts are closely related to the conflict preorder. The conflict preorder does not imply language inclusion, i.e., $G \lesssim_{\text{conf}} H$ does not imply $\mathcal{L}(G) \subseteq \mathcal{L}(H)$. A relationship between the languages of two automata related through the conflict preorder can only be established when certain conflicts are taken into account.

Lemma 4. (Malik *et al.*, 2006) Let G and H be arbitrary automata. If $G \lesssim_{\text{conf}} H$ then

(i) $\textsc{Conf}(G) \subseteq \textsc{Conf}(H)$;
(ii) $\mathcal{L}(G) \cup \textsc{Conf}(G) \subseteq \mathcal{L}(H) \cup \textsc{Conf}(H)$.

The following lemma is needed below to prove Prop. 8. It follows from the results cited above and shows how certain conflicts can be preserved under synchronous composition.

Lemma 5. Let G and H be two automata. If $s \in \textsc{Conf}(G)$ and $s \in \mathcal{L}(H) \cup \textsc{Conf}(H)$, it follows that $s \in \textsc{Conf}(G \| H)$.

Proof. Let $s \in \textsc{Conf}(G)$ and $s \in \mathcal{L}(H) \cup \textsc{Conf}(H)$, and let T be an arbitrary automaton such that $T \overset{s}{\Rightarrow}$. It is to be shown that $G \parallel H \parallel T$ is blocking. Consider two cases.

If $s \in \mathcal{L}(H)$, then clearly $H \| T \overset{s}{\Rightarrow}$, and since $s \in \textsc{Conf}(G)$, it follows that $G \parallel H \parallel T$ is blocking.

Otherwise $s \in \textsc{Conf}(H)$, and also $s \in \textsc{Conf}(G)$ by assumption. By Lemma 3 there exist prefixes $r_G, r_H \sqsubseteq s$ such that $r_I \in \mathcal{L}(I) \cap \textsc{Conf}(I)$ for $I \in \{G, H\}$. Choose $r = r_I$ to be the shorter of these prefixes. Then $G \parallel T \overset{r}{\Rightarrow}$ and $H \parallel T \overset{r}{\Rightarrow}$, and since $r = r_I \in \textsc{Conf}(I)$, this implies that $G \parallel H \parallel T$ is blocking. □

Exponential complexity algorithms are known to compute the set of certain conflicts of a given automaton (Malik, 2010) and to test whether two given automata are related through the conflict preorder (Ware and Malik, 2011).

3. HISC USING THE CONFLICT PREORDER

Large discrete event systems become intractable when modelled as a flat synchronous product of many automata without any structure. One solution is to organise the system into a hierarchy of *subsystems*, each containing only a few automata. *Hierarchical Interface-Based Supervisory Control using the Conflict Preorder (HISC-CP)* is based on a tree-like structure, defined recursively as follows.

Definition 8. A *Hierarchical Interface Structure (HIS)* is a 5-tuple
$$\mathbf{H} = \langle r, I, G, S, LL \rangle \tag{3}$$
where

- $r \in \mathbb{N}_0$ is the *rank* of $\mathbf{H}$, also denoted by $\text{rank}(\mathbf{H})$.
- I, G, and S are automata, called the *interface*, *plant*, and *supervisor* of $\mathbf{H}$, respectively.
- LL is a finite set of HIS, called the *lower levels* of $\mathbf{H}$, such that $\text{rank}(\mathbf{L}) < r$ for each $\mathbf{L} \in LL$.

Given an HIS $\mathbf{H} = \langle r, I, G, S, LL \rangle$ with $I = \langle \Sigma_I, Q_I, \rightarrow_I, Q_I^\circ, Q_I^\omega \rangle$, $G = \langle \Sigma_G, Q_G, \rightarrow_G, Q_G^\circ, Q_G^\omega \rangle$, and $S = \langle \Sigma_S, Q_S, \rightarrow_S, Q_S^\circ, Q_S^\omega \rangle$, the following additional notations are used.

- $I(\mathbf{H}) = I$ is the interface of $\mathbf{H}$.
- $LL(\mathbf{H}) = LL$ denotes the lower levels of $\mathbf{H}$.
- $\Sigma_I(\mathbf{H}) = \Sigma_I$ is the *interface alphabet* of $\mathbf{H}$.
- $\Sigma_H(\mathbf{H}) = \Sigma_G \cup \Sigma_S$ is the *local* or *high-level alphabet* of $\mathbf{H}$.
- $\Sigma(\mathbf{H}) = \Sigma_G \cup \Sigma_S \cup \bigcup_{\mathbf{L} \in LL} \Sigma(\mathbf{L})$ is the *global alphabet* of $\mathbf{H}$.
- $G(\mathbf{H}) = G \parallel \|_{\mathbf{L} \in LL} G(\mathbf{L})$ is the *flat plant* of $\mathbf{H}$.
- $S(\mathbf{H}) = S \parallel \|_{\mathbf{L} \in LL} S(\mathbf{L})$ is the *flat supervisor* of $\mathbf{H}$.
- $F(\mathbf{H}) = G(\mathbf{H}) \parallel S(\mathbf{H})$ is the *flat system* of $\mathbf{H}$.

The above definitions describe a structure similar to (Leduc *et al.*, 2005; Hill *et al.*, 2010), with each subsystem consisting of local plant, specification, and interface automata. The flat system describes the composition of all the automata in the hierarchy and thus the behaviour of the complete system. Unlike the above works, interfaces are not part of the flat system. The interface of an HIS $\mathbf{H}$ is understood as an *abstraction* that can replace $\mathbf{H}$ when analysing a larger system containing $\mathbf{H}$.

For the subsystems to be considered in isolation, they may not share any events except through the interface alphabets. Unlike previous approaches, HISC-CP allows events to be shared by different lower levels, if these events are in both interfaces. Strict event locality is thus sacrificed for increased expressiveness.

Definition 9. An HIS $\mathbf{H}$ is *well-formed* if it satisfies the following conditions.

- $\Sigma_I(\mathbf{H}) \subseteq \Sigma_H(\mathbf{H})$.
- $\Sigma(\mathbf{L}) \cap \Sigma_H(\mathbf{H}) \subseteq \Sigma_I(\mathbf{L})$ for all $\mathbf{L} \in LL(\mathbf{H})$.
- $\Sigma(\mathbf{L}_1) \cap \Sigma(\mathbf{L}_2) \subseteq \Sigma_I(\mathbf{L}_1)$ for all $\mathbf{L}_1, \mathbf{L}_2 \in LL(\mathbf{H})$ with $\mathbf{L}_1 \neq \mathbf{L}_2$.
- Every $\mathbf{L} \in LL(\mathbf{H})$ is well-formed.

A crucial question in supervisory control is whether a system is controllable and nonblocking. This amounts to checking whether the flat system of an HIS satisfies these properties.

Definition 10. An HIS $\mathbf{H}$ is *globally controllable* if $S(\mathbf{H})$ is controllable with respect to $G(\mathbf{H})$.

Definition 11. An HIS $\mathbf{H}$ is *globally nonblocking* if $F(\mathbf{H})$ is nonblocking.

HISC abandons these global conditions in favour of sufficient conditions checked *locally* for each subsystem. HISC-CP uses the conflict preorder to define these so-called *interface consistency* conditions.

Definition 12. An HIS $\mathbf{H}$ is *interface consistent* if it satisfies the following conditions.

- $(G \parallel S \parallel \|_{\mathbf{L} \in LL} I(\mathbf{L})) \setminus (\Sigma \setminus \Sigma_I(\mathbf{H})) \lesssim_{\text{conf}} I(\mathbf{H})$.
- Every $\mathbf{L} \in LL(\mathbf{H})$ is interface consistent.

An HIS $\mathbf{H}$ is interface consistent if its interface is a conflict-preserving abstraction of the automata in $\mathbf{H}$ and the interfaces at the next level. All events not used in the interface can be hidden as well-formedness ensures that they cannot be used by any other subsystem. The requirement for the interface to be *more conflicting* ensures that the nonblocking property is preserved if the subsystem is replaced by its interface when analysing another system that uses the subsystem. The following result lifts the interface consistency condition to the complete flat system containing all lower levels of the hierarchy.

Proposition 6. Let HIS $\mathbf{H}$ be well-formed and interface consistent. Then $F(\mathbf{H}) \setminus (\Sigma \setminus \Sigma_I(\mathbf{H})) \lesssim_{\text{conf}} I(\mathbf{H})$.

Proof. Let $\mathbf{H} = \langle r, I, G, S, LL \rangle$ and $I = \langle \Sigma_I, Q_I, \rightarrow_I, Q_I^\circ, Q_I^\omega \rangle$. The claim is shown by induction on the rank r of $\mathbf{H}$.

If $r = 0$ then $LL = \emptyset$, and it follows directly from Def. 12 that $F(\mathbf{H}) \setminus (\Sigma \setminus \Sigma_I) = (G \parallel S \parallel \|_{\mathbf{L} \in LL} F(\mathbf{L})) \setminus (\Sigma \setminus \Sigma_I) = (G \parallel S) \setminus (\Sigma \setminus \Sigma_I) = (G \parallel S \parallel \|_{\mathbf{L} \in LL} I(\mathbf{L})) \setminus (\Sigma \setminus \Sigma_I) \lesssim_{\text{conf}} I$.

Now consider $r = n + 1$, and assume the claim holds for all $\mathbf{L} \in LL$. It follows that,

$$\begin{aligned}
&F(\mathbf{H}) \setminus (\Sigma \setminus \Sigma_I) \\
&= \big(G \parallel S \parallel \mathop{\|}_{\mathbf{L}\in LL} F(\mathbf{L})\big) \setminus (\Sigma \setminus \Sigma_I) \\
&= \big(G \parallel S \parallel \mathop{\|}_{\mathbf{L}\in LL} [F(\mathbf{L}) \setminus (\Sigma \setminus \Sigma_I(\mathbf{L}))]\big) \setminus (\Sigma \setminus \Sigma_I) \\
&\qquad\qquad \text{(by Def. 9 as } \mathbf{H} \text{ is well-formed)} \\
&\lesssim_{\text{conf}} (G \parallel S \parallel \mathop{\|}_{\mathbf{L}\in LL} I(\mathbf{L})) \setminus (\Sigma \setminus \Sigma_I) \\
&\qquad\qquad \text{(by inductive assumption and Prop. 1)} \\
&\lesssim_{\text{conf}} I \quad \text{(by Def. 12 as } \mathbf{H} \text{ is interface consistent)} \quad \square
\end{aligned}$$

By Prop. 6, the local property of interface consistency ensures that the interface of an HIS $\mathbf{H}$ is a more conflicting abstraction of the complete flat system of $\mathbf{H}$. If the interface is nonblocking, this is enough to ensure that the complete system is nonblocking.

Proposition 7. Let HIS $\mathbf{H} = \langle r, I, G, S, LL\rangle$ be well-formed and interface consistent. If I is nonblocking, then $\mathbf{H}$ is globally nonblocking.

Proof. Note that $F(\mathbf{H}) \setminus (\Sigma \setminus \Sigma_I(\mathbf{H})) \lesssim_{\text{conf}} I$ by Prop. 6. Since I is nonblocking, it follows from Lemma 4(i) that $\textsc{Conf}(F(\mathbf{H}) \setminus (\Sigma \setminus \Sigma_I(\mathbf{H}))) \subseteq \textsc{Conf}(I) = \emptyset$, i.e., $\textsc{Conf}(F(\mathbf{H})) = \emptyset$. This means that $F(\mathbf{H})$ is nonblocking, i.e., $\mathbf{H}$ is globally nonblocking. $\square$

If an HIS $\mathbf{H}$ has a nonblocking interface, then interface consistency guarantees that the complete hierarchy below $\mathbf{H}$ is nonblocking. Usually it is desired for subsystems to be nonblocking on their own, and such subsystems can be represented by nonblocking interfaces. If the topmost system in a hierarchy is not intended for use as part of another system, a one-state nonblocking interface $I_{\text{top}} = \langle \emptyset, \{q^\circ\}, \emptyset, \{q^\circ\}, \{q^\circ\}\rangle$ can capture the requirement that the global system must be nonblocking.

Global controllability is more difficult to prove. It is known that controllability of subsystems implies controllability of the global system (Brandin *et al.*, 2004). This suggests that global controllability can be ensured if each subsystem is controllable on its own. Unfortunately, these results do not take interfaces into account. Sometimes, the following stronger condition is needed to prove global controllability.

Definition 13. An HIS $\mathbf{H}$ is *locally controllable* if it satisfies the following conditions.

- S is controllable with respect to $G \parallel \|_{\mathbf{L}\in LL} I(\mathbf{L})$.
- Every $\mathbf{L} \in LL(\mathbf{H})$ is locally controllable.

The problem with taking the interfaces into account is that the conflict preorder is not directly linked to language inclusion when certain conflicts are present. Yet, the additional assumption of the system being nonblocking ensures that interface consistency in combination with local controllability implies global controllability.

Proposition 8. Let HIS $\mathbf{H} = \langle r, I, G, S, LL\rangle$ be well-formed, interface consistent, and locally controllable. If I is nonblocking, then $\mathbf{H}$ is globally controllable.

Proof. It is shown by induction on the rank r that for all $s \in \Sigma^*$ and for all $v \in \Sigma_u$ such that $S(\mathbf{H}) \overset{s}{\Rightarrow} x_{S(\mathbf{H})}$ and $G(\mathbf{H}) \overset{s}{\Rightarrow} x_{G(\mathbf{H})} \overset{v}{\to}$, it holds that $S(\mathbf{H}) \overset{s}{\Rightarrow} x_{S(\mathbf{H})} \overset{v}{\to}$ or $s \in \textsc{Conf}(I)$. As $\textsc{Conf}(I) = \emptyset$ for nonblocking I, this implies the claim.

If $r = 0$, then $LL = \emptyset$. In this case, $S = S(\mathbf{H}) \overset{s}{\Rightarrow} x_{S(\mathbf{H})}$ and $G \| \|_{\mathbf{L}\in LL} I(\mathbf{L}) = G(\mathbf{H}) \overset{s}{\Rightarrow} x_{G(\mathbf{H})} \overset{v}{\to}$, and since $\mathbf{H}$ is locally controllable, it follows by Def. 13 that $S(\mathbf{H}) \overset{s}{\Rightarrow} x_{S(\mathbf{H})} \overset{v}{\to}$.

Now consider $r = n + 1$, and assume the claim holds for all $\mathbf{L} \in LL$. By assumption $S \parallel \|_{\mathbf{L}\in LL} S(\mathbf{L}) = S(\mathbf{H}) \overset{s}{\Rightarrow} x_{S(\mathbf{H})} = (x_S, (x_{S(\mathbf{L})})_{\mathbf{L}\in LL})$. That is, for each $\mathbf{L} \in LL$ there exists a state $x_{S(\mathbf{L})}$ such that $S(\mathbf{L}) \overset{s}{\Rightarrow} x_{S(\mathbf{L})}$, and likewise there exists a state $x_{G(\mathbf{L})}$ such that $G(\mathbf{L}) \overset{s}{\Rightarrow} x_{G(\mathbf{L})} \overset{v}{\to}$. By inductive assumption, it follows that $S(\mathbf{L}) \overset{s}{\Rightarrow} x_{S(\mathbf{L})} \overset{v}{\to}$ or $s \in \textsc{Conf}(I(\mathbf{L}))$. Let $F'(\mathbf{L}) = F(\mathbf{L}) \setminus (\Sigma \setminus \Sigma_I(\mathbf{L}))$, and note that $F'(\mathbf{L}) \lesssim_{\text{conf}} I(\mathbf{L})$ by Prop. 6, and therefore

$$\mathcal{L}(F'(\mathbf{L})) \cup \textsc{Conf}(F'(\mathbf{L})) \subseteq \mathcal{L}(I(\mathbf{L})) \cup \textsc{Conf}(I(\mathbf{L})) \quad (4)$$

by Lemma 4(ii), for all $\mathbf{L} \in LL$. Consider two cases.

If $s \in \textsc{Conf}(I(\mathbf{L}_0))$ for some $\mathbf{L}_0 \in LL$, then first note that as $F(\mathbf{L}) = G(\mathbf{L}) \parallel S(\mathbf{L}) \overset{s}{\Rightarrow}$ for all $\mathbf{L} \in LL$ and by (4):

$$\begin{aligned} s \in \mathcal{L}(F(\mathbf{L})) \subseteq \mathcal{L}(F'(\mathbf{L})) &\subseteq \mathcal{L}(F'(\mathbf{L})) \cup \textsc{Conf}(F'(\mathbf{L})) \\ &\subseteq \mathcal{L}(I(\mathbf{L})) \cup \textsc{Conf}(I(\mathbf{L}))\ . \quad (5) \end{aligned}$$

As this holds for all $\mathbf{L} \in LL$, and given $s \in \textsc{Conf}(I(\mathbf{L}_0))$ it follows by Lemma 5 that $s \in \textsc{Conf}(G \parallel S \parallel \|_{\mathbf{L}\in LL} I(\mathbf{L}))$. Since furthermore $(G \| S \| \|_{\mathbf{L}\in LL} I(\mathbf{L})) \setminus (\Sigma \setminus \Sigma_I(\mathbf{H})) \lesssim_{\text{conf}} I$ by Def. 12, it follows by Lemma 4(i) that $P_{\Sigma_I(\mathbf{H})}(s) \in \textsc{Conf}\big((G \parallel S \parallel \|_{\mathbf{L}\in LL} I(\mathbf{L})) \setminus (\Sigma \setminus \Sigma_I(\mathbf{H}))\big) \subseteq \textsc{Conf}(I)$, and since I has alphabet $\Sigma_I(\mathbf{H})$ also $s \in \textsc{Conf}(I)$.

Otherwise $s \notin \textsc{Conf}(I(\mathbf{L}))$ for all $\mathbf{L} \in LL$, so $S(\mathbf{L}) \overset{s}{\Rightarrow} x_{S(\mathbf{L})} \overset{v}{\to}$ and thus $F(\mathbf{L}) \overset{sv}{\Rightarrow}$ for all $\mathbf{L} \in LL$. It remains to be shown that $S \overset{s}{\Rightarrow} x_S \overset{v}{\to}$. Since $sv \in \mathcal{L}(F(\mathbf{L}))$, it follows from (4) for all $\mathbf{L} \in LL$ that

$$\begin{aligned} sv \in \mathcal{L}(F(\mathbf{L})) \subseteq \mathcal{L}(F'(\mathbf{L})) &\subseteq \mathcal{L}(F'(\mathbf{L})) \cup \textsc{Conf}(F'(\mathbf{L})) \\ &\subseteq \mathcal{L}(I(\mathbf{L})) \cup \textsc{Conf}(I(\mathbf{L}))\ . \quad (6) \end{aligned}$$

As $s \notin \textsc{Conf}(I(\mathbf{L}))$ for all $\mathbf{L} \in LL$, it follows that $sv \in \mathcal{L}(I(\mathbf{L}))$ for all $\mathbf{L} \in LL$. This is because $s \notin \textsc{Conf}(I(\mathbf{L}))$ and $sv \in \textsc{Conf}(I(\mathbf{L}))$ implies $sv \in \mathcal{L}(I(\mathbf{L}))$ by Lemma 2 and 3. It follows that $G \| \|_{\mathbf{L}\in LL} I(\mathbf{L}) \overset{sv}{\Rightarrow}$, and since $S \overset{s}{\Rightarrow} x_S$ and $\mathbf{H}$ is locally controllable, it follows from Def. 13 that $S \overset{s}{\Rightarrow} x_S \overset{v}{\to}$. $\square$

In summary, an HIS is guaranteed to be globally controllable and nonblocking if each subsystem satisfies the local conditions of well-formedness, interface consistency, and local controllability.

Well-formedness is a straightforward syntactic condition, and local controllability can be checked by standard algorithms after construction of the synchronous composition (Ramadge and Wonham, 1989). To verify interface consistency, it is necessary to determine whether the interface is more conflicting than the automata in the subsystem and the interfaces at the next level. The conflict preorder can be checked by an exponential algorithm (Ware and Malik, 2011).

For improved performance, polynomial abstraction algorithms (Flordal and Malik, 2009; Malik and Leduc, 2009) can replace the subsystem by a conflict equivalent abstraction using only the interface events. Then the smaller abstraction can be compared to the interface by the conflict preorder algorithm. This algorithm (Ware and Malik, 2011) can analyse automata with several thousand states, which is likely to be sufficient for many applications.

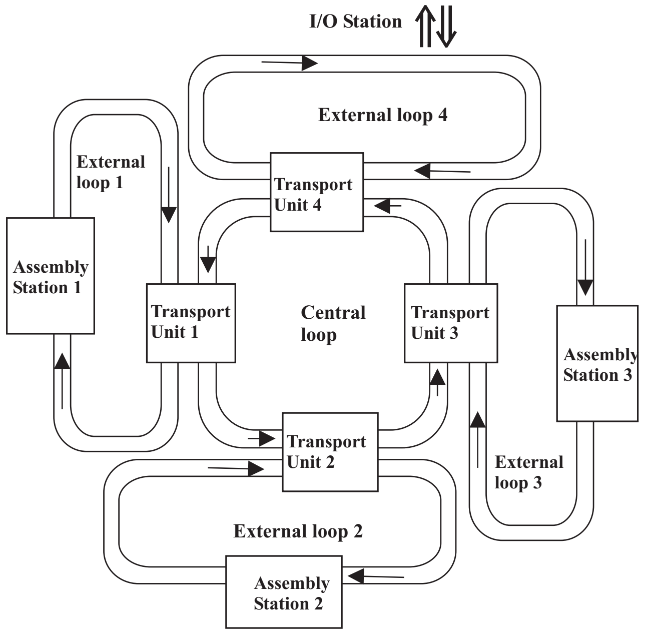

Fig. 1. The Atelier Inter-établissement de Productique.

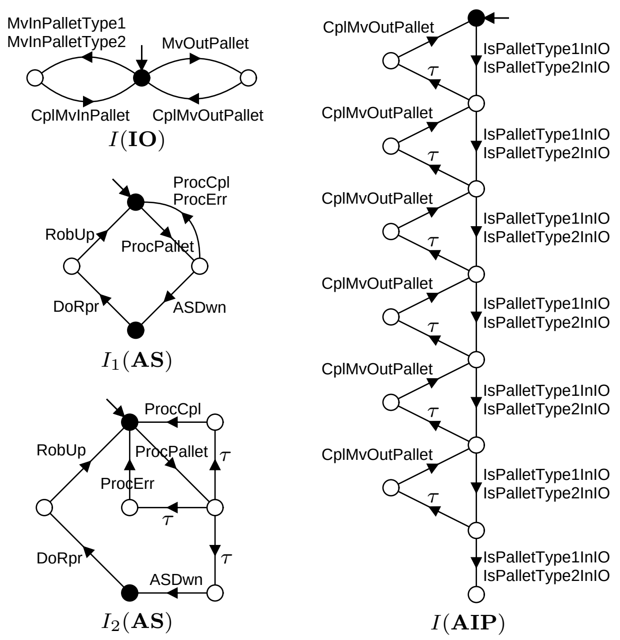

Fig. 2. Some interfaces for AIP manufacturing system.

The abstraction algorithms can also be used to automatically generate conflict equivalent interfaces for legacy subsystems. These interfaces can later be edited manually, or used to confirm that a subsystem continues to satisfy interface consistency after modifications.

4. EXAMPLE

This section discusses hierarchical interface design in the context of modelling a large manufacturing system, the Atelier Inter-établissement de Productique (AIP). The system was first modelled as a discrete event system by (Brandin and Charbonnier, 1994), and later by (Leduc, 2002) using HISC and by (Ma and Wonham, 2005) using state tree structures. This paper is based on a more detailed hierarchical model (Song, 2006).

The AIP system coordinates the transport and processing of workpieces in pallets. The system consists of a central loop and four external loop conveyors as shown in Fig. 1. Pallets can be transferred between the central and external loops by four transport units. External loops 1, 2, and 3 each have an assembly station with a robot to process pallets. External loop 4 is linked to an input/output station to allow pallets to enter and leave the system. Two types of pallets, Type1 and Type2, can enter the system.

According to (Song, 2006), the AIP system is modelled in a two-level hierarchy with one high level coordinating eight low levels, one for each of the four transfer units, the three assembly units, and the input/output station. Fig. 2 shows some possible interface automata for this model.

The input/output station is described by $I(\mathbf{IO})$ (Song, 2006). The input/output station can be requested to transfer a pallet into external loop 4 (MvInPalletType1 or MvInPalletType2) or to remove a pallet from there (MvOutPallet), and it reports back on completion of the transfer (CplMvInPallet or CplMvOutPallet). Due to its simplicity, this interface from (Song, 2006) can be used unchanged with HISC-CP.

The assembly stations are more involved. An assembly station is given a pallet (ProcPallet), which it presents to its robot for assembly. It then releases the pallet, and reports the success (ProcCpl) or failure (ProcErr) of the assembly operation. Alternatively, the robot may break down (ASDwn), which also leads to release of the pallet, but afterwards the robot must be repaired (DoRpr, RobUp) before another pallet can be processed (Leduc, 2002).

The assembly station interface $I_1(\mathbf{AS})$ is a so-called *command-pair interface* (Leduc, 2002). *Request* events ProcPallet and DoRpr are paired with *answer* events ProcCpl, ProcErr, ASDwn, and RobUp. The HISC interface conditions (Leduc, 2002) require the high level always to be capable of processing all answer events that are possible according to the interface. For example, at any time after the request ProcPallet, the robot may break down (ASDwn), so the high level must be able to process this answer.

HISC-CP does not distinguish event types, so the answer event semantics must be expressed through nonblocking. This is achieved by the more detailed interface $I_2(\mathbf{AS})$. Silent transitions linked to additional states show that under certain circumstances the low level may allow only some events. As the high level cannot synchronise on τ, after sending the request ProcPallet, it must be able to continue with each answer ProcCpl, ProcErr, and ASDwn to avert blocking.

HISC-CP interfaces may be larger than command-pair interfaces for master-slave hierarchies with clear request and answer events. Also, as HISC-CP only considers nonblocking, a high level using $I_2(\mathbf{AS})$ only needs to be able to process each of the events ProcCpl, ProcErr, and ASDwn *eventually*, whereas an HISC high level using $I_1(\mathbf{AS})$ must be able to accept these answers *immediately*.

On the other hand, HISC-CP interfaces are more flexible and can also be used when there are no obvious request and answer events. For example, $I(\mathbf{AIP})$ presents an interface for the entire AIP system. The manufacturing cell can be presented up to six Type1 or Type2 pallets in the input/output station (IsPalletType1InIO or IsPalletType2InIO), and each pallet will eventually be ejected (CplMvOutPallet). If more than six pallets are presented, the system may block. The blocking interface $I(\mathbf{AIP})$ models a subsystem that does not prevent blocking on its own, and shows what is required of a high-level coordinator to ensure nonblocking.

The τ-transitions show that at any time—even if no further pallets are presented—the AIP can eventually eject a pallet. However, after starting the output of a pallet (MvOutPallet), the system temporarily stops to accept new pallets as input, i.e., events IsPalletType$X$InIO are not possible until output is completed. A high level using this interface can assume that pallets will eventually be ejected (CplMvOutPallet), but it should not assume that the subsystem will always be able to accept a new pallet (IsPalletType$X$InIO).

$I(\mathbf{AIP})$ cannot be interpreted as a command-pair interface (Leduc, 2002), which does not permit two request events to occur in sequence without an answer in between. *Low data interfaces* (Leduc, 2009) are more expressive and allow the specification of behaviour like $I(\mathbf{AIP})$, albeit after the introduction of polling with additional events in the interface alphabet. HISC-CP does not restrict the choice of events: an interface exists for every subsystem and set of interface events, although these interfaces may be nondeterministic and include τ-transitions.

5. CONCLUSIONS

The framework of *Hierarchical Interface-Based Supervisory Control using the Conflict Preorder (HISC-CP)* has been proposed as an alternative approach to hierarchical supervisor design.

Previous HISC frameworks such as (Leduc, 2009) are based on a master-slave relationship between subsystem, and for such systems offer good event localisation and more structure and guidance for system design. This leads to smaller subsystems with smaller interfaces, which are deterministic and easy to understand.

HISC-CP allows a wide range of interfaces, including nondeterministic interfaces. It it is more flexible in how the system is decomposed, potentially leading to smaller interfaces and subsystems when there is no clear master-slave relationship. The algorithms to check HISC-CP interface consistency are exponential, which may pose challenges when faced with large interfaces. On the other hand, it is possible to automatically generate interfaces using polynomial algorithms.

The framework of HISC-CP is not directly comparable to previous HISC frameworks, with the interface consistency conditions of neither approach implying the other. It remains to be investigated which framework gives better models under which circumstances.

REFERENCES

Brandin, Bertil A., Robi Malik and Petra Malik (2004). Incremental verification and synthesis of discrete-event systems guided by counter-examples. *IEEE Trans. Control Syst. Technol.* **12**(3), 387–401.

Brandin, Bertil and François Charbonnier (1994). The supervisory control of the automated manufacturing system of the AIP. In: *Proc. Rensselaer's 4th Int. Conf. Computer Integrated Manufacturing and Automation Technology.* Troy, NY, USA. pp. 319–324.

De Nicola, R. and M. C. B. Hennessy (1984). Testing equivalences for processes. *Theoretical Comput. Sci.* **34**(1–2), 83–133.

Flordal, Hugo and Robi Malik (2009). Compositional verification in supervisory control. *SIAM J. Control and Optimization* **48**(3), 1914–1938.

Hill, R. C., J. E. R. Cury, M. H. de Queiroz, D. M. Tilbury and S. Lafortune (2010). Multi-level hierarchical interface-based supervisory control. *Automatica* **46**(7), 1152–1164.

Hoare, C. A. R. (1985). *Communicating Sequential Processes.* Prentice-Hall.

Leduc, R. J., Pengcheng Dai and Raoguang Song (2009). Synthesis method for hierarchical interface-based supervisory control. *IEEE Trans. Autom. Control* **54**(7), 1548–1560.

Leduc, Ryan J. (2009). Hierarchical interface-based supervisory control with data events. *Int. J. Control* **82**(5), 783–800.

Leduc, Ryan J., Mark Lawford and W. M. Wonham (2005). Hierarchical interface-based supervisory control—part II: Parallel case. *IEEE Trans. Autom. Control* **50**(9), 1336–1348.

Leduc, Ryan James (2002). Hierarchical Interface-based Supervisory Control. PhD thesis. Dept. of Electrical Engineering, University of Toronto, ON, Canada.

Lin, Feng and W. Murray Wonham (1990). Decentralized control and coordination of discrete-event systems with partial observation. *IEEE Trans. Autom. Control* **35**(12), 1330–1337.

Ma, Chuan and W. Murray Wonham (2005). *Nonblocking Supervisory Control of State Tree Structures.* Vol. 317 of *LNCIS*. Springer.

Malik, Robi (2010). The language of certain conflicts of a nondeterministic process. Working Paper 05/2010, Dept. of Computer Science, University of Waikato, Hamilton, New Zealand.

Malik, Robi and Ryan Leduc (2009). A compositional approach for verifying generalised nonblocking. In: *Proc. 7th Int. Conf. Control and Automation, ICCA '09.* Christchurch, New Zealand. pp. 448–453.

Malik, Robi, David Streader and Steve Reeves (2006). Conflicts and fair testing. *Int. J. Found. Comput. Sci.* **17**(4), 797–813.

Ramadge, Peter J. G. and W. Murray Wonham (1989). The control of discrete event systems. *Proc. IEEE* **77**(1), 81–98.

Song, Raoguang (2006). Symbolic synthesis and verification of hierarchical interface-based supervisory control. Master's thesis. Dept. of Computing and Software, McMaster University, Hamilton, ON, Canada.

Ware, Simon and Robi Malik (2011). A state-based characterisation of the conflict preorder. In: *Proc. 10th Int. Workshop on the Foundations of Coordination Languages and Software Architectures, FOCLASA 2011.* Aachen, Germany. pp. 34–48.

Wong, K. C. and W. M. Wonham (1996). Hierarchical control of discrete-event systems. *Discrete Event Dyn. Syst.* **6**(3), 241–273.