Noname manuscript No.
(will be inserted by the editor)

Hierarchical Modelling of Manufacturing Systems Using Discrete Event Systems and the Conflict Preorder

Robi Malik · **Ryan Leduc**

the date of receipt and acceptance should be inserted later

Abstract This paper introduces *Hierarchical Interface-Based Supervisory Control using the Conflict Preorder* and applies it to the design of two manufacturing systems models of practical scale. Hierarchical Interface-Based Supervisory Control decomposes a large 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.

1 Introduction

Supervisory control theory of discrete event systems [33, 44] is a general framework for the design and synthesis of reactive control functions. Systems are typically modelled using a set of finite-state machines interacting in lock-step synchronisation [15, 33], so the framework is naturally suited for modular decomposition. Since the inception of supervisory control theory, numerous extensions for modular, decentralised, and hierarchical control have been proposed to cope with the ever-increasing size and complexity of industrial-scale control systems.

The simplest approach to modular supervisory control is horizontal decomposition or *decentralised* supervisory control [3,32,34,37,42,44], where related components are grouped together and controlled in isolation. While this approach works well to ensure *controllability*, the essential requirement to be *nonblocking* is not so easy to ensure and often ignored in the early work [5, 22]. It has later been found that natural projection with the *observer property* can help to ensure the nonblocking property in decentralised frameworks [10, 11, 25, 35, 41, 43].

Robi Malik
Dept. of Computer Science, University of Waikato, Hamilton, New Zealand, E-mail: robi@waikato.ac.nz

Ryan Leduc
Dept. of Computing and Software, McMaster University, Hamilton, Canada, E-mail: leduc@mcmaster.ca

Alternatively, the system can be decomposed vertically into high-level subsystems that coordinate the action of other low-level subsystems. This is the idea of *hierarchical* supervisory control. The decomposition can be done using hierarchical events [41, 44] or hierarchical states [13, 23].

Other approaches [2, 9] apply concepts from *software design* [6] to supervisory control. Software engineers have long advocated the decomposition of software into *modules* or *subsystems* that interact via well-defined *interfaces* [31]. The interfaces are typically the first part of a module to be designed. They define the *contract* [29] between a module and its users, facilitate the re-use and documentation of modules, and form the basis for automatic *formal verification*. The software approach can also be applied to automata models [2], where behavioural interfaces are modelled as automata, and a *refinement* relation aids in constructing implementations from the interfaces.

Hierarchical Interface-based Supervisory Control (HISC) [17,19,21] combines the ideas of hierarchical structuring of control functions in supervisory control with interfaces. A large discrete event system is decomposed into subsystems, the behaviour of which is defined by interfaces. After implementation of a subsystem, it is formally verified against its *interface consistency* conditions. General results ensure that, if all subsystems satisfy these interface consistency conditions locally, then global properties such as controllability and nonblocking follow for the combined system. The complete system state space never needs to be explored, offering substantial savings in computational effort for verification.

Most 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 [17, 19]. The expressiveness is increased by the addition of *low data* events [18]. The framework has also been extended to support synthesis of least restrictive subsystem controllers [20] and multi-level hierarchies [14].

While the master-slave structure is common in software, it is not always appropriate for reactive and control systems. As an alternative, *projection-based* abstractions [10,35,42] do not impose a master-slave relationship. While these methods are not as structured as HISC, it is known that a projection that satisfies the *observer property* [25, 42, 43] can serve as an abstraction of a subsystem and thus as its interface. With additional requirements such as *output control consistency* [10] or *local control consistency* [30, 35], it is also possible to ensure maximal permissiveness of supervisors when using synthesis. However, the observer property is a strong requirement, and it is not guaranteed that an interface exists for every given subsystem and choice of interface events [25]. The set of interface events may have to be extended [11] by exposing internal transitions of the subsystem to outside users.

The more recent approach of *Hierarchical Interface-based Supervisory Control using the Conflict Preorder (HISC-CP)* [26] does not assume a master-slave relationship either. Based on results about the *conflict preorder* [28], interfaces are conflict-preserving abstractions of the subsystem they represent. The use of nondeterministic interfaces ensures the existence of interfaces for every subsystem and every choice of interface events, avoiding the expressive limitations of natural projection [11,25,42] and HISC [18,21]. The more complicated interface consistency conditions can be verified by an exponential conflict-preorder algorithm [39]. Due to the increased expressive power, the HISC-CP interface consistency conditions facilitate the design of better hierarchies and allow full flexibility in the construction of interfaces and subsystems.

This paper is an extended version of [26]. The HISC-CP modelling approach is described more clearly using two detailed examples of manufacturing systems, and experimental results demonstrate the feasibility of the approach for a large-scale system. In the following, Section 2 introduces the background of discrete event systems and the conflict preorder. This is followed in Section 3 by the description of the HISC-CP methodology and interface consistency conditions. Next, Section 4 applies hierarchical interface design to two practical examples of manufacturing systems, and Section 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 $\Sigma'_\tau = \Sigma' \dot\cup \{\tau\}$ 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 $L \subseteq \Sigma^*$ 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, $x \xRightarrow{s} y$, 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 $\xRightarrow{s}$ 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, $x \to y$ denotes the existence of $s \in \Sigma_\tau^*$ 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 $Q^\circ \xrightarrow{s} x$. The same notations are introduced for $\Rightarrow$.

The prefix-closed *language* of the automaton G is $\mathscr{L}(G) = \{\, s \in \Sigma^* \mid G \xRightarrow{s} \,\}$. 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 [15] 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

$$G \,\|\, H = \langle Q_G \times Q_H, \Sigma_G \cup \Sigma_H, \rightarrow, Q_G^\circ \times Q_H^\circ, Q_G^\omega \times Q_H^\omega \rangle \tag{1}$$

where

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

In synchronous composition, shared events must be executed by all automata synchronously, while other events (including τ) are executed independently. Under the notation in this paper, where the languages of all automata are defined over the full alphabet Σ, synchronous composition of automata coincides with the intersection of their languages. The following proposition generalises the known result [33] to the case of nondeterministic automata with silent events considered here.

Proposition 1 For any two automata G and H, it holds that $\mathscr{L}(G \,\|\, H) = \mathscr{L}(G) \cap \mathscr{L}(H)$.

Proof First assume $s \in \mathscr{L}(G \,\|\, H)$. Then there exist initial states x_G° of G and x_H° of H and a path

$$(x_G^\circ, x_H^\circ) \xrightarrow{\sigma_1} (x_G^1, x_H^1) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} (x_G^n, x_H^n) \tag{2}$$

in $G \,\|\, H$ such that $s = P_\Sigma(\sigma_1 \cdots \sigma_n)$. Consider a transition $(x_G^{k-1}, x_H^{k-1}) \xrightarrow{\sigma_k} (x_G^k, x_H^k)$ on this path. If $\sigma_k \in \Sigma_G$ then $x_G^{k-1} \xrightarrow{\sigma_k} x_G^k$ by Def. 2. If $\sigma_k = \tau$ then it holds by Def. 2 that $x_G^{k-1} \xrightarrow{\tau} x_G^k$ or $x_G^{k-1} = x_G^k$, which both implies $x_G^{k-1} \xRightarrow{\varepsilon} x_G^k$ in G. If $\sigma_k \notin \Sigma_G \cup \{\tau\}$ then $x_G^{k-1} = x_G^k$ by Def. 2, and thus $x_G^{k-1} \xrightarrow{\sigma_k} x_G^{k-1} = x_G^k$ by the definition of $\rightarrow$ for events not in the automaton alphabet. In all three cases $x_G^{k-1} \xRightarrow{P_\Sigma(\sigma_k)} x_G^k$ in G. As this can be shown for all transitions on the path (2), it follows that $x_G^\circ \xRightarrow{s} x_G^n$, i.e., $s \in \mathscr{L}(G)$. Likewise, it is shown that $s \in \mathscr{L}(H)$ and therefore $s \in \mathscr{L}(G) \cap \mathscr{L}(H)$.

Conversely, assume $s \in \mathscr{L}(G) \cap \mathscr{L}(H)$. Then there exist initial states x_G^0 of G and x_H^0 of H and paths

$$x_G^0 \xRightarrow{\sigma_1} x_G^1 \xRightarrow{\sigma_2} \cdots \xRightarrow{\sigma_n} x_G^n \quad \text{in } G\,; \tag{3}$$

$$x_H^0 \xRightarrow{\sigma_1} x_H^1 \xRightarrow{\sigma_2} \cdots \xRightarrow{\sigma_n} x_H^n \quad \text{in } H\,; \tag{4}$$

such that $s = \sigma_1 \cdots \sigma_n$. Consider a pair of transitions $x_G^{k-1} \xRightarrow{\sigma_k} x_G^k$ and $x_H^{k-1} \xRightarrow{\sigma_k} x_H^k$ on these paths. There exists paths $x_G^{k-1} = y_G^0 \xrightarrow{\tau} \cdots \xrightarrow{\tau} y_G^i \xrightarrow{\sigma_k} y_G^{i+1} \xrightarrow{\tau} \cdots \xrightarrow{\tau} y_G^j = x_G^k$ in G and $x_H^{k-1} = y_H^0 \xrightarrow{\tau} \cdots \xrightarrow{\tau} y_H^l \xrightarrow{\sigma_k} y_H^{l+1} \xrightarrow{\tau} \cdots \xrightarrow{\tau} y_H^m = x_H^k$ in H, which by Def. 2 implies $(x_G^{k-1}, x_H^{k-1}) = (y_G^0, y_H^0) \xrightarrow{\tau} \cdots \xrightarrow{\tau} (y_G^i, y_H^0) \xrightarrow{\tau} \cdots \xrightarrow{\tau} (y_G^i, y_H^l) \xrightarrow{\sigma_k} (y_G^{i+1}, y_H^{l+1}) \xrightarrow{\tau} \cdots \xrightarrow{\tau} (y_G^j, y_H^{l+1}) \xrightarrow{\tau} \cdots \xrightarrow{\tau} (y_G^j, y_H^m) = (x_G^k, x_H^k)$ in $G \,\|\, H$. This means $(x_G^{k-1}, x_H^{k-1}) \xRightarrow{\sigma_k} (x_G^k, x_H^k)$ in $G \,\|\, H$, and as this can be shown for all transitions on the paths (3) and (4), it follows that $s \in \mathscr{L}(G \,\|\, H)$. □

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* [33] 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 $\upsilon \in \Sigma_\mathrm{u}$ such that $K \overset{s}{\Rightarrow}_K x$ and $G \overset{s\upsilon}{\Rightarrow}_G$, it holds that $x \overset{\upsilon}{\rightarrow}_K$.

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 $G \parallel H$ 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 [28] 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 [8]. 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 [28] Automaton G is *less conflicting* than automaton H, written $G \lesssim_\mathrm{conf} H$, if for any automaton T such that $H \parallel T$ is nonblocking, $G \parallel T$ also is nonblocking. G and H are *conflict equivalent*, written $G \simeq_\mathrm{conf} H$, if $G \lesssim_\mathrm{conf} H$ and $H \lesssim_\mathrm{conf} G$.

The relation $\lesssim_\mathrm{conf}$ is known as the *conflict preorder*. Its use in compositional reasoning is based on the fact that it is preserved under the operations of synchronous composition and hiding. These so-called *congruence* properties have been established in [28].

Definition 7 Let $\lesssim$ be a relation on the set of automata.

- The relation $\lesssim$ is a *pre-congruence* with respect to synchronous composition, if for all automata $G \lesssim H$ and for every automaton T, it holds that $G \parallel T \lesssim H \parallel T$.
- The relation $\lesssim$ is a pre-congruence with respect to hiding, if for all automata $G \lesssim H$ and for all $\Upsilon \subseteq \Sigma$, it holds that $G \setminus \Upsilon \lesssim H \setminus \Upsilon$.

Proposition 2 [28] The conflict preorder $\lesssim_\mathrm{conf}$ is a pre-congruence with respect to synchronous composition and hiding.

In fact, the conflict preorder is the coarsest nonblocking-preserving preorder with these congruence properties [28]. It produces the smallest possible conflict-preserving abstractions, and therefore is used for efficient compositional verification of the nonblocking property [12,27,40].

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 8 [24] The *set of certain conflicts* of automaton G is

$$\textsc{Conf}(G) = \{\, s \in \Sigma^* \mid \text{for every automaton } T, \text{ if } T \overset{s}{\Rightarrow} \text{ then } G \parallel T \text{ is blocking} \,\} \ . \tag{5}$$

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 [28]. 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. Lemma 3 below shows that, if $s \in \textsc{Conf}(G)$ then $st \in \textsc{Conf}(G)$ for any trace $t \in \Sigma^*$, even if $st \notin \mathscr{L}(G)$. Conversely, Lemma 4 shows that 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 3 [24] Let G be an automaton. Then $\textsc{Conf}(G) = \textsc{Conf}(G)\Sigma^*$.

Lemma 4 [24] Let G be an automaton, and let $s \in \textsc{Conf}(G)$. Then there exists a prefix $r \sqsubseteq s$ such that $r \in \mathscr{L}(G) \cap \textsc{Conf}(G)$.

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 $\mathscr{L}(G) \subseteq \mathscr{L}(H)$. 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 5 [28] Let G and H be arbitrary automata. If $G \lesssim_{\text{conf}} H$ then

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

The following lemma is needed below to prove Prop. 10. It follows from the results cited above and shows how certain conflicts can be preserved under synchronous composition.

Lemma 6 Let G and H be two automata. If $s \in \textsc{Conf}(G)$ and $s \in \mathscr{L}(H) \cup \textsc{Conf}(H)$, it follows that $s \in \textsc{Conf}(G \parallel H)$.

Proof Let $s \in \textsc{Conf}(G)$ and $s \in \mathscr{L}(H) \cup \textsc{Conf}(H)$, 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 $s \in \mathscr{L}(H)$, then clearly $H \parallel T \overset{s}{\Rightarrow}$, 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 4 there exist prefixes $r_G, r_H \sqsubseteq s$ such that $r_I \in \mathscr{L}(I) \cap \textsc{Conf}(I)$ 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 [24] and to test whether two given automata are related through the conflict preorder [39].

3 HISC Using the Conflict Preorder

This paper proposes a development methodology of hierarchical *interface-first design*, which is similar to common design practises in software engineering [6, 31]. When faced with the task of developing a large control system, engineers will first identify its key components or *subsystems*. Subsystems may be groups of related physical system components or software components with related functionality.

Having identified the subsystems, the next step is to describe their behaviour. For each subsystem, the *interface* is defined to capture the behaviour of the subsystem as it presents itself to an outside user. Each subsystem is assigned a set of *interface events*, which it can use to communicate with other subsystems, and its behavioural *interface* is specified using interface automata. The interface defines the sequencing of events and the communication protocol to be implemented by the subsystem. It forms a *contract* [29] for the use of the subsystem and at the same time provides vital design documentation. Ideally, an interface only describes the external interactions of the subsystem and reveals as little as possible about its internal workings.

Once the subsystems and their interfaces are defined, the design process is separated, and control solutions for each subsystem are developed in isolation. The designer for each subsystem implements the subsystem contract as defined by the interface, using standard design methods for small-scale discrete event systems. If a subsystem relies on functionality provided by another subsystem, then that functionality is accessed only through the defined interface of the other subsystem.

HISC imposes *interface consistency* conditions [19, 21] to be satisfied by each subsystem, and tools are available to verify these conditions automatically. If the subsystems are implemented in such a way that each subsystem satisfies its interface conditions locally, then it is guaranteed that the global system consisting of all the subsystems together has desired properties such as controllability and being nonblocking. This guarantee can be made without the need to analyse the complete global system. If a subsystem is modified at a later stage, only its local interface consistency conditions need to be rechecked to ensure the properties of the global system.

In the following, Section 3.1 describes the structure of subsystems using the formal concept of Hierarchical Interface Structures. Then, Section 3.2 describes the interface consistency conditions used in this paper for Hierarchical Interface-Based Supervisory Control using the Conflict Preorder (HISC-CP). Finally, Section 3.3 shows that these interface consistency conditions are powerful enough to capture any kind of behavioural interface.

3.1 Hierarchical Interface Structures

With *Hierarchical Interface-Based Supervisory Control using the Conflict Preorder (HISC-CP)*, subsystems are arranged in tree-like hierarchy as shown in Fig. 1. Subsystems are modelled recursively as Hierarchical Interface Structures, which contain automata and possibly other subsystems.

Definition 9 A *Hierarchical Interface Structure (HIS)* is a 5-tuple

$$\mathbf{H} = \langle r, I, G, S, LL \rangle \tag{6}$$

where

- $r \in \mathbb{N}_0$ is the *rank* of $\mathbf{H}$, also denoted by $\mathrm{rank}(\mathbf{H})$.

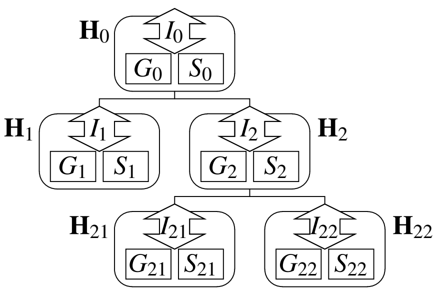

Fig. 1 Hierarchical Interface Structure.

- 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 $\operatorname{rank}(\mathbf{L}) < r$ for each $\mathbf{L} \in LL$.

Fig. 1 shows the overall structure of an HIS. An HIS contains its interface, plant, and specification, each of which may be a single automaton or the synchronous composition of several automata. In addition, an HIS contains zero or more lower-level subsystems, which are in turn HIS. In the figure, HIS $\mathbf{H}_0$ contains its interface I_0, plant G_0, and specification S_0, as well as two lower levels $\mathbf{H}_1$ and $\mathbf{H}_2$, where $\mathbf{H}_2$ contains two further lower levels $\mathbf{H}_{21}$ and $\mathbf{H}_{22}$. The rank prevents cyclic structures by setting a maximum nesting depth of the lower-level subsystems. In the figure, the rank of $\mathbf{H}_0$ could be 2, if the rank of $\mathbf{H}_1$ and $\mathbf{H}_2$ is 1 and the rank of $\mathbf{H}_{21}$ and $\mathbf{H}_{22}$ is 0.

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 [14, 21], 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 10 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 11 An HIS $\mathbf{H}$ is *globally controllable* if $S(\mathbf{H})$ is controllable with respect to $G(\mathbf{H})$.

Definition 12 An HIS $\mathbf{H}$ is *globally nonblocking* if $F(\mathbf{H})$ is nonblocking.

3.2 Interface Consistency

HISC abandons global conditions such as controllability and nonblocking 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 13 An HIS $\mathbf{H}$ is *interface consistent* if it satisfies the following conditions.

– $(G \,\|\, S \,\|\, \|_{\mathbf{L} \in LL} I(\mathbf{L})) \setminus (\Sigma \setminus \Sigma_I(\mathbf{H})) \lesssim_{\text{conf}} I(\mathbf{H})$.
– 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 of its lower-level subsystems. 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 7 Let HIS $\mathbf{H}$ be well-formed and interface consistent. Then it holds that

$$F(\mathbf{H}) \setminus (\Sigma \setminus \Sigma_I(\mathbf{H})) \lesssim_{\text{conf}} I(\mathbf{H}) . \tag{7}$$

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. 13 that $F(\mathbf{H}) \setminus (\Sigma \setminus \Sigma_I) = (G \,\|\, S \,\|\, \|_{\mathbf{L} \in LL} F(\mathbf{L})) \setminus (\Sigma \setminus \Sigma_I) = (G \,\|\, S) \setminus (\Sigma \setminus \Sigma_I) = (G \,\|\, S \,\|\, \|_{\mathbf{L} \in LL} I(\mathbf{L})) \setminus (\Sigma \setminus \Sigma_I) \lesssim_{\text{conf}} I$.

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

$$\begin{aligned}
F(\mathbf{H}) \setminus (\Sigma \setminus \Sigma_I) &= \big(G \,\|\, S \,\| \mathop{\|}_{\mathbf{L} \in LL} F(\mathbf{L})\big) \setminus (\Sigma \setminus \Sigma_I) \\
&= \big(G \,\|\, S \,\| \mathop{\|}_{\mathbf{L} \in LL} [F(\mathbf{L}) \setminus (\Sigma \setminus \Sigma_I(\mathbf{L}))]\big) \setminus (\Sigma \setminus \Sigma_I) \\
&\qquad \text{(by Def. 10 as } \mathbf{H} \text{ is well-formed)} \\
&\lesssim_{\text{conf}} (G \,\|\, S \,\| \mathop{\|}_{\mathbf{L} \in LL} I(\mathbf{L})) \setminus (\Sigma \setminus \Sigma_I) \\
&\qquad \text{(by inductive assumption and Prop. 2)} \\
&\lesssim_{\text{conf}} I \quad \text{(by Def. 13 as } \mathbf{H} \text{ is interface consistent)}
\end{aligned}$$

□

By Prop. 7, 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 8 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. 7. Since I is nonblocking, it follows from Lemma 5(i) that $\textsc{Conf}(F(\mathbf{H}) \setminus (\Sigma \setminus \Sigma_I(\mathbf{H}))) \subseteq \textsc{Conf}(I) = \emptyset$. This means that $F(\mathbf{H}) \setminus (\Sigma \setminus \Sigma_I(\mathbf{H}))$ is nonblocking, i.e., $F(\mathbf{H})$ is nonblocking. Thus, $\mathbf{H}$ 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.

Prop. 8 does not require that all subsystems of an HIS be nonblocking; this is only required of the top level. A subsystem could be blocking along with its interface: it then is up to an upper level to establish nonblocking through interaction with the lower level's interface. Such a situation is allowable in this framework and may work fine if the blocking is corrected by the levels above, but it can make understanding a system more difficult. As a design guideline, the concept of being *locally nonblocking* is introduced in Def. 14, which states that every subsystem is nonblocking on its own. It is not required that an HIS be locally nonblocking, but if it is, each subsystem is more self-contained and easier to understand. If the system is interface consistent, then it follows from Def. 13 that the system is locally nonblocking if all interfaces are nonblocking.

Definition 14 An HIS $\mathbf{H} = \langle r, I, G, S, LL \rangle$ is *locally nonblocking* if it satisfies the following conditions.

- $G \parallel S \parallel \|_{\mathbf{L} \in LL} I(\mathbf{L})$ is nonblocking.
- Every $\mathbf{L} \in LL$ is locally nonblocking.

Global controllability is more difficult to prove. It is known that controllability of subsystems implies controllability of the global system [5]. This suggests that global controllability can be ensured if each subsystem is controllable on its own.

Definition 15 An HIS $\mathbf{H} = \langle r, I, G, S, LL \rangle$ is *subsystem controllable* if it satisfies the following conditions.

- S is controllable with respect to G.
- Every $\mathbf{L} \in LL$ is subsystem controllable.

Proposition 9 Let $\mathbf{H} = \langle r, I, G, S, LL \rangle$ be an HIS. If $\mathbf{H}$ is subsystem controllable, then $\mathbf{H}$ is globally controllable.

Proof This is an immediate consequence of Prop. 3 and 4 in [5]. $\square$

Unfortunately, the above result does not take interfaces into account. Sometimes, the following stronger condition is needed to prove global controllability. In the definition that follows, the interfaces of the subsystems at the next level are added to the plant of HIS $\mathbf{H}$ for the purpose of checking controllability. Thus, the designer of the high-level HIS $\mathbf{H}$ treats $G \parallel \|_{\mathbf{L} \in LL} I(\mathbf{L})$ as the plant model when designing the local supervisor S.

Definition 16 An HIS $\mathbf{H} = \langle r, I, G, S, LL \rangle$ 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 $\mathbf{L} \in LL$ 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 10 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 $\upsilon \in \Sigma_{\mathrm{u}}$ such that $S(\mathbf{H}) \overset{s}{\Rightarrow} x_{S(\mathbf{H})}$ and $G(\mathbf{H}) \overset{s}{\Rightarrow} x_{G(\mathbf{H})} \overset{\upsilon}{\rightarrow}$, it holds that $S(\mathbf{H}) \overset{s}{\Rightarrow} x_{S(\mathbf{H})} \overset{\upsilon}{\rightarrow}$ 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 $G \parallel \|_{\mathbf{L}\in LL} I(\mathbf{L}) = G(\mathbf{H}) \overset{s}{\Rightarrow} x_{G(\mathbf{H})} \overset{\upsilon}{\rightarrow}$, and since $\mathbf{H}$ is locally controllable, it follows by Def. 16 that $S(\mathbf{H}) \overset{s}{\Rightarrow} x_{S(\mathbf{H})} \overset{\upsilon}{\rightarrow}$.

Now assume $\mathbf{H}$ has rank $r+1$ and that 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 $G(\mathbf{L}) \overset{s}{\Rightarrow} x_{G(\mathbf{L})} \overset{\upsilon}{\rightarrow}$. By inductive assumption, it follows that $S(\mathbf{L}) \overset{s}{\Rightarrow} x_{S(\mathbf{L})} \overset{\upsilon}{\rightarrow}$ 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 $F'(\mathbf{L}) \lesssim_{\mathrm{conf}} I(\mathbf{L})$ by Prop. 7, and therefore

$$\mathscr{L}(F'(\mathbf{L})) \cup \textsc{Conf}(F'(\mathbf{L})) \subseteq \mathscr{L}(I(\mathbf{L})) \cup \textsc{Conf}(I(\mathbf{L})) \tag{8}$$

by Lemma 5(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 (8):

$$s \in \mathscr{L}(F(\mathbf{L})) \subseteq \mathscr{L}(F'(\mathbf{L})) \subseteq \mathscr{L}(F'(\mathbf{L})) \cup \textsc{Conf}(F'(\mathbf{L})) \subseteq \mathscr{L}(I(\mathbf{L})) \cup \textsc{Conf}(I(\mathbf{L}))\ .$$

As this holds for all $\mathbf{L} \in LL$, and given $s \in \textsc{Conf}(I(\mathbf{L}_0))$ it follows by Lemma 6 that $s \in \textsc{Conf}(G \parallel S \parallel \|_{\mathbf{L}\in LL} I(\mathbf{L}))$. Since furthermore $(G \parallel S \parallel \|_{\mathbf{L}\in LL} I(\mathbf{L})) \setminus (\Sigma \setminus \Sigma_I(\mathbf{H})) \lesssim_{\mathrm{conf}} I$ by Def. 13, it follows by Lemma 5(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 $S(\mathbf{L}) \overset{s}{\Rightarrow} x_{S(\mathbf{L})} \overset{\upsilon}{\rightarrow}$ and thus $F(\mathbf{L}) \overset{s\upsilon}{\Rightarrow}$ for all $\mathbf{L} \in LL$. It remains to be shown that $S \overset{s}{\Rightarrow} x_S \overset{\upsilon}{\rightarrow}$. Since $s\upsilon \in \mathscr{L}(F(\mathbf{L}))$, it follows from (8) for all $\mathbf{L} \in LL$ that

$$s\upsilon \in \mathscr{L}(F(\mathbf{L})) \subseteq \mathscr{L}(F'(\mathbf{L})) \subseteq \mathscr{L}(F'(\mathbf{L})) \cup \textsc{Conf}(F'(\mathbf{L})) \subseteq \mathscr{L}(I(\mathbf{L})) \cup \textsc{Conf}(I(\mathbf{L}))\ .$$

That is, $s\upsilon \in \mathscr{L}(I(\mathbf{L}))$ or $s\upsilon \in \textsc{Conf}(I(\mathbf{L}))$ for all $\mathbf{L} \in LL$. If $s\upsilon \in \textsc{Conf}(I(\mathbf{L}))$, then since $s \notin \textsc{Conf}(I(\mathbf{L}))$ it follows that $s\upsilon$ is a shortest trace of certain conflicts, and thus $s\upsilon \in \mathscr{L}(I(\mathbf{L})) \cap \textsc{Conf}(I(\mathbf{L})) \subseteq \mathscr{L}(I(\mathbf{L}))$ by Lemma 4. Hence, in both cases, $s\upsilon \in \mathscr{L}(I(\mathbf{L}))$ for all $\mathbf{L} \in LL$. It follows that $G \parallel \|_{\mathbf{L}\in LL} I(\mathbf{L}) \overset{s\upsilon}{\Rightarrow}$, and since $S \overset{s}{\Rightarrow} x_S$ and $\mathbf{H}$ is locally controllable, it follows from Def. 16 that $S \overset{s}{\Rightarrow} x_S \overset{\upsilon}{\rightarrow}$. □

In summary, an HIS is guaranteed to be globally controllable and nonblocking, if the top-level interface is nonblocking and 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 [33]. To verify interface consistency, it is necessary to determine whether the interface is more conflicting than the composition of the automata in the subsystem and the interfaces at the next level. The conflict preorder can be checked by an exponential algorithm [39]. For improved performance, polynomial abstraction algorithms [12,27,40] 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.

3.3 Expressive Power

Given Def. 13 of interface consistency, the question arises under which circumstances an interface satisfying this definition exists. This question can be answered positively for HISC-CP: an interface exists for every subsystem and choice of interface events.

Proposition 11 Let $A = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an automaton, and let $\Sigma_I \subseteq \Sigma$. Then there exists an interface automaton $I = \langle \Sigma_I, Q_I, \rightarrow_I, Q_I^\circ, Q_I^\omega \rangle$ such that $A \setminus (\Sigma \setminus \Sigma_I) \lesssim_{\text{conf}} I$.

Proof Consider $I = A \setminus (\Sigma \setminus \Sigma_I)$. As $\lesssim_{\text{conf}}$ is a reflexive relation, it holds that $A \setminus (\Sigma \setminus \Sigma_I) = I \lesssim_{\text{conf}} I$. □

As an interface is an *abstraction* of its subsystem rather than a part of the system, the existence of interfaces follows trivially. Every subsystem can serve as an abstraction of itself and thus as its own interface—after hiding of non-interface events.

Other hierarchical approaches do not have this property. With previous HISC methods, interfaces must have *command-pair* [19] or *LD-interface* [18] structure, and not every behaviour can be modelled in this way. With projection-based methods, an interface only exists if the projection to the set of interface events has the *observer property* [25,43]. If these properties are not satisfied, the only way to obtain an interface is to include additional events in the interface alphabet. Yet, this exposes more information about the subsystem to its users, which usually is not desired from an interface design point of view. Prop. 11 shows that HISC-CP does not have such restrictions.

If a legacy subsystem $G \,\|\, S$ without interface is given, then Prop. 11 can be used to *extract* or synthesise an interface for it. While the interface $I = (G \,\|\, S) \setminus (\Sigma \setminus \Sigma_I)$ from the proof is typically large, it can be simplified using conflict-preserving abstraction algorithms [12,27,40]. These are polynomial algorithms that, given an automaton, compute a conflict equivalent abstraction, which is typically smaller than the original automaton. Yet, the result is conflict equivalent to the subsystem, rather than more conflicting, and it is likely that simpler and better understandable interfaces can be obtained by manual design.

4 Examples

In this section, the process of hierarchical interface design is applied to two practical examples of manufacturing systems. All supervisors and interfaces presented were designed manually, and verified using the discrete event systems modelling tool *Supremica* [1] to satisfy the interface conditions. If verification of some property failed, the models were modified and checked again, until all checks passed.

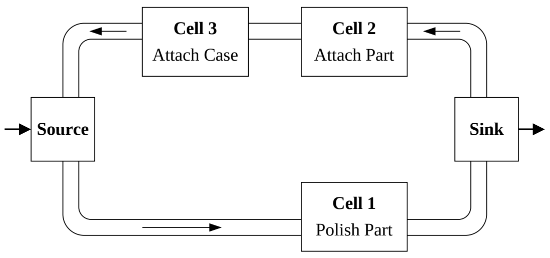

Fig. 2 Simple Manufacturing System.

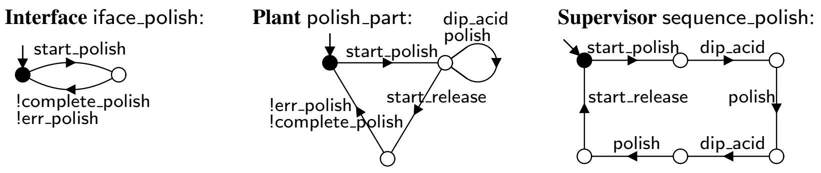

Fig. 3 Cell 1 — Polishing subsystem.

Section 4.1 describes the complete model of a medium-scale manufacturing system. This system was designed following the methodology of hierarchical interface-first design. The interfaces were designed first, and afterwards the subsystems were modelled and verified until each subsystem passed the interface consistency conditions.

Section 4.2 shows the structure and interfaces for a hierarchical model of the large-scale AIP manufacturing system. This system was modified from a legacy model [36]. The hierarchical interface structure was changed to increase the number of hierarchy levels. In a few cases, interfaces were extracted automatically from the subsystems, but the results were only used for guidance. All interfaces were created or modified manually.

4.1 Simple Manufacturing Example

The first example demonstrates hierarchical interface design for a simple manufacturing system. The model presented is inspired by earlier work [4, 19, 38], and extended to demonstrate capabilities of HISC-CP. The manufacturing system consists of three cells connected by a conveyor belt as shown in Fig. 2. Parts enter the system from the source at the left, and are processed by cells 1, 2, and 3 in this order, before exiting through the sink at the right. The model of the system is hierarchically decomposed into three low-level subsystems representing each of the three cells, and a top-level system that coordinates these three cells. In the following, a hierarchical automata model of the simple manufacturing system is proposed, starting with the three subsystems. In the figures, initial states are marked by an incoming arrow, and terminal states are coloured black. Uncontrollable events are prefixed with an exclamation mark (!).

The model of Cell 1 is shown in Fig. 3. Interface iface_polish specifies that the high-level can request the cell to start the polishing sequence (start_polish), and the cell will respond that polishing has completed with success (!complete_polish) or failure (!err_polish). The decision about success or failure is made at the very end of the process, so both results are possible until the end. The interface has the structure of a *command-pair interface* [19].

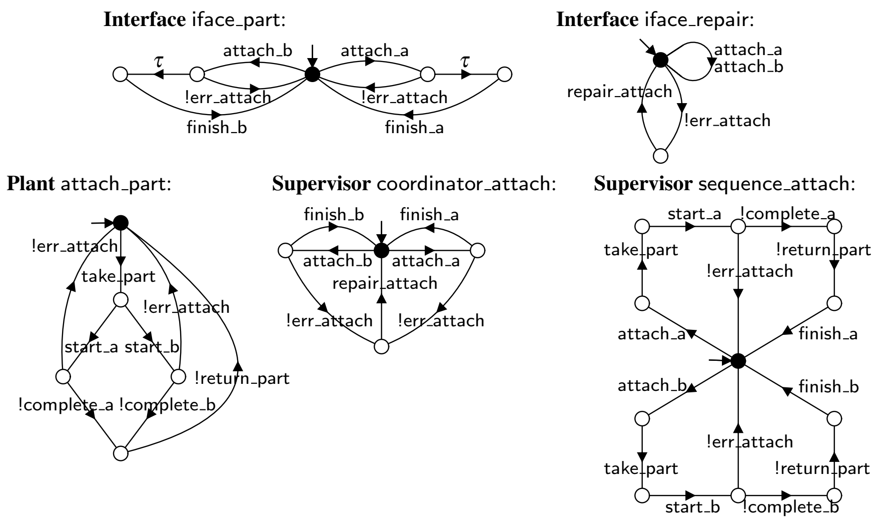

Fig. 4 Cell 2 — Attach Part subsystem.

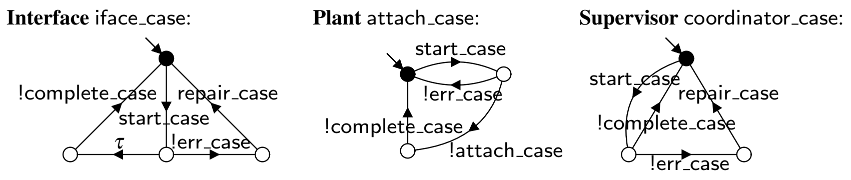

Fig. 5 Cell 3 — Attach Case subsystem.

The interface behaviour is reflected by plant polish_part, which permits polishing action between the start_polish and start_release events, and afterwards reports the success or failure of the operation. The supervisor sequence_polish ensures that the dip_acid and polish actions are executed twice during the polishing sequence.

The interface of Cell 2, shown in Fig. 4, is the synchronous composition of two automata iface_part and iface_repair. Interface iface_part describes the normal behaviour of the cell: the high-level coordinator may request to attach a part of type A or B (attach_a or attach_b), which may result in successful completion (finish_a or finish_b), or in an error (!err_attach). Unlike Cell 1, the error is not always possible, and the τ events indicate that a coordinator interacting with Cell 2 cannot rely on an error to occur. The second interface component iface_repair specifies that, if an error occurs, the cell must be repaired (repair_attach) before it can be used again.

The plant model attach_part describes the possible sequencing of low-level commands in this cell, which is coordinated by the supervisors coordinator_attach and sequence_attach. These components ensure the behaviour specified by the interface, i.e., that a request to attach part A or B initiates the commands for the requested part, and after an error no further request is possible until the cell is repaired.

Cell 3, shown in Fig. 5, works in a similar way to Cell 2, except that it can only attach a single type of part. Interface iface_case specifies that a start_case command may result in an error (!err_case) or in successful completion (complete_case), with the error not always

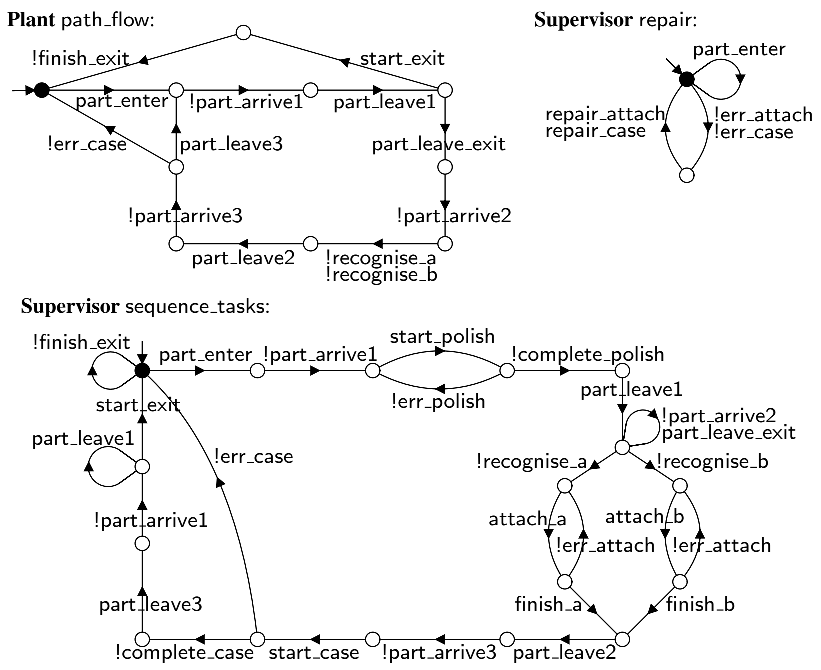

Fig. 6 Top-level Subsystem for Simple Manufacturing System.

possible. After an error, the cell must be repaired (repair_case) before it can be used again. The low-level plant attach_case and supervisor coordinator_case ensure this behaviour.

Fig. 6 shows the components of a top-level subsystem, which coordinates the three cell subsystems through their interfaces. The plant model path_flow describes the layout of the manufacturing system as shown in Fig. 2. In front of each cell, there is a part acquisition unit that automatically stops a part and holds it until given a release command. In front of Cell 2, there is a part recognition sensor that determines the need to attach part A or B. The supervisor sequence_tasks routes the parts through the cells in the required order, executes the appropriate commands for the cell and the part type, and then allows the completed part to leave the system. This model assumes that it is possible to recover from errors in Cells 1 and 2 by repeating the operation that caused the error, while an error in Cell 3 causes a loss of the workpiece and forces a restart. In addition, supervisor repair prevents parts from entering the system while Cell 2 or 3 is under repair. The top-level subsystem has no interface, except for the implicit interface I_{top} to specify a globally nonblocking system.

The three cell subsystems and the top-level subsystem are easily found to be interface consistent and locally controllable by Supremica. It follows by Prop. 8 that the complete system is nonblocking, and by Prop. 10 that the complete system is controllable.

4.2 AIP Automated Manufacturing System

The second example describes a HISC-CP model of a a large manufacturing system, the Atelier Inter-établissement de Productique (AIP). This system was first modelled as a discrete event system in [4], and later in [17, 36] using HISC and in [23] using state tree structures.

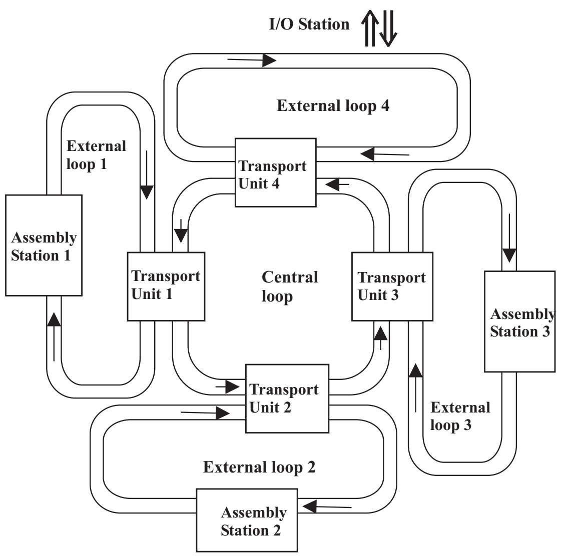

Fig. 7 The Atelier Inter-établissement de Productique.

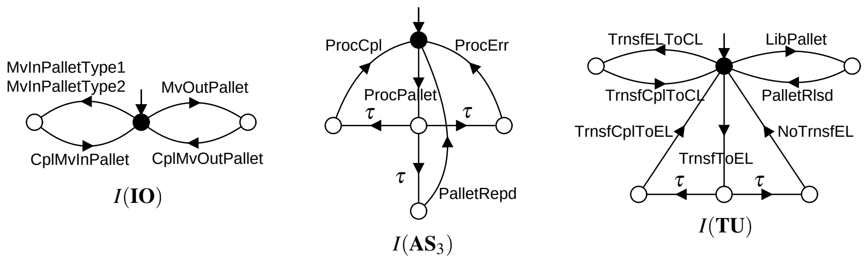

Fig. 8 Interfaces for AIP manufacturing system components.

The model proposed here is a modified version of [36]. The previous two-level hierarchical was modified to a three-level hierarchy with HISC-CP, and the interaction between the subsystems was changed in order to reduce polling.

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. 7. 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.

The AIP system is modelled in a three-level hierarchy. The top level is a coordinator that also controls the central loop. It contains four subsystems controlling each of the four external loops, which contain two subsystems each for the transport units, the assembly machines, and the input/output station. The external loops 1 and 2 and their transfer units and assembly stations are identical, while the other subsystems are all slightly different.

The complete model consists of 13 HIS with 209 automata in total. The reachable state space exceeds 10^8 states, and brute-force verification using explicit state enumeration or BDDs was unsuccessful in Supremica. Figures 8, 9, and 10 show some key interfaces in this model, which are explained below.

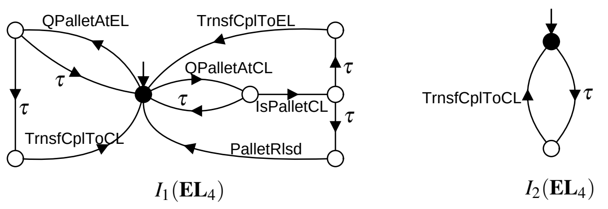

Fig. 9 Interfaces for AIP external loop 4.

Interface $I(\mathbf{IO})$ in Fig. 8 describes the behaviour of the input/output station. It can be requested to transfer a pallet of type 1 or type 2 into external loop 4 (MvInPalletType1 or MvInPalletType2) or to remove a pallet from there (MvOutPallet), and reports back on completion of the transfer (CplMvInPallet or CplMvOutPallet). Due to its simplicity, this interface is identical to the command-pair interface proposed in [36].

The assembly stations and transfer units are more involved. Assembly station 3, modelled by interface $I(\mathbf{AS}_3)$ in Fig. 8, is given a pallet (ProcPallet), which it presents to its robot for assembly. It then releases the pallet, and reports whether the pallet was successfully processed (ProcCpl) or repaired (PalletRepd), or indicates a failure (ProcErr) of the operation [17]. As the assembly station can decide internally which result occurs, another subsystem using the component must allow for all the three results to ensure nonblocking. This is modelled using the τ-transitions, which can neither be controlled nor observed by a high level using this interface. As the high-level user is unaware of which state the subsystem has entered, it always needs to be prepared for a situation where only one of the three answers ProcCpl, PalletRepd, or ProcErr is possible, and ensure that termination remains possible in each case.

The transfer units, modelled by interface $I(\mathbf{TU})$ in Fig. 8, can perform three different operations. They can be requested to transfer a pallet from their external loop to the central loop (TransfELtoCL), or to keep a pallet in the central loop (LibPallet). These requests are followed by replies (TransfCplToCL or PalletRlsd respectively) indicating successful completion. Alternatively, transfer units can be requested to transfer a pallet from the central to their external loop (TransfToEL), which may result in success (TransfCplToEL) or failure (NoTransfEL). Again, τ-transitions in $I(\mathbf{TU})$ model the fact that the transfer unit decides internally on the success or failure of the operation.

The interfaces in Fig. 8 are adapted from the HISC command-pair interfaces [17] proposed in [36]. However, the original interfaces do not include any τ-transitions and therefore are smaller. The need for the next higher level to support all possible results of an operation is expressed in HISC through different event types. For example, event ProcPallet in $I(\mathbf{AS}_3)$ is a so-called *request* event, while ProcCpl, PalletRepd, and ProcErr are *answer* events. The HISC semantics [17] requires the high-level subsystem to support all possible answers, eliminating the need for τ-transitions.

HISC-CP does not distinguish event types, so the answer event semantics is expressed through nonblocking. Silent τ-transitions linked to additional states show that the low level may allow only some events. As the high level cannot synchronise on τ, after sending the request ProcPallet to the assembly station, it must be able to continue with each answer ProcCpl, PalletRepd, and ProcErr to ensure nonblocking in combination with the interface $I(\mathbf{AS}_3)$. While this approach uses more states, it is more expressive and makes it possible to specify exactly which answers must be supported by a high-level subsystem.

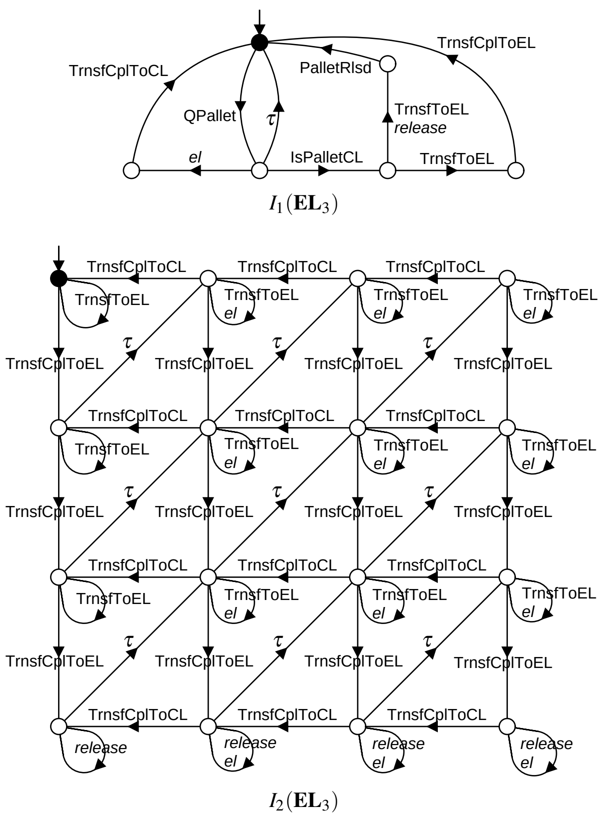

Fig. 10 Interfaces for AIP external loop 3.

The interface for external loop 4 is given by the synchronous composition $I_1(\mathbf{EL}_4) \parallel I_2(\mathbf{EL}_4)$ of the automata in Fig. 9. This external loop simultaneously serves as the source and sink of pallets.

Interface component $I_1(\mathbf{EL}_4)$ models the basic behaviour pattern of this subsystem. The high-level coordinator can request a pallet from the external loop (QPalletAtEL), which may result in completion of transfer (TransfCplToCL) or in no answer at all if no pallet is available. Alternatively, the high-level coordinator can request a pallet to be moved into the external loop (QPalletAtCL), which results in a notification (IsPalletCL) if a pallet is present or no answer otherwise. After the notification, the transfer may succeed (TransfCplToEL) or fail (PalletRlsd).

As the external loop always accepts new pallets from the input/output station, the high-level coordinator cannot indefinitely refuse new pallets. This is modelled by the second interface component $I_2(\mathbf{EL}_4)$, which says that it must always be possible for event Transf-CplToCL to occur in order to ensure nonblocking. This implies that the high-level coordinator must repeatedly request new pallets through the event QPalletAtEL, or otherwise the system will block.

The interface of external loop 3 is given by the synchronous composition of the automata in Fig. 10, more precisely

$$I(\mathbf{EL}_3) = (I_1(\mathbf{EL}_3) \parallel I_2(\mathbf{EL}_3)) \setminus \{\mathit{el}, \mathit{release}\} \ . \tag{9}$$

Table 1 Experimental Results.

Subsystem	$\|I\|$	$\|F\|$	**States**	**LCont**	**IConsist**
AIP	0	17	18,101,632	0.03 s	5.89 s
$\mathbf{EL}_1, \mathbf{EL}_2$	3	8	1,636	0.01 s	1.46 s
$\mathbf{EL}_3$	2	6	442	0.01 s	0.24 s
$\mathbf{EL}_4$	2	9	1,854	0.01 s	0.19 s
$\mathbf{TU}_1, \mathbf{TU}_2$	1	23	98	0.02 s	0.23 s
$\mathbf{TU}_3$	1	27	204	0.02 s	0.22 s
$\mathbf{TU}_4$	1	19	152	0.02 s	0.17 s
$\mathbf{AS}_1, \mathbf{AS}_2$	1	16	120	0.01 s	0.14 s
$\mathbf{AS}_3$	1	24	106	0.02 s	0.14 s
IO	1	1	2	0.00 s	0.00 s

The internal events *el* and *release* are used to model the interface more concisely, but they are not available for interaction with the next higher level and are not part of the interface.

Interface component $I_1(\mathbf{EL}_3)$ models the basic behaviour pattern of external loop 3. This subsystem can be requested to check for presence of a pallet (QPallet). This may result in no answer, if no pallet is detected, or in transfer of a pallet detected at the external loop to the central loop (TransfCplToCL). Alternatively, if a pallet is detected in the central loop, first a notification is sent (IsPalletCL). Then the subsystem may decide to reject the pallet (PalletRlsd) if the external loop is full, or it may send a second notification to signal acceptance (TransfToEL). The subsequent attempt to transfer may succeed (TransfCplToEL) or fail (PalletRlsd).

The second interface component $I_2(\mathbf{EL}_3)$ models the capacity restrictions of external loop 3. The conveyors of the external loop can only hold six pallets, three pallets travelling from the transfer unit to the assembly station and three pallets travelling from the assembly station to the transfer unit. The notification TransfToEL is only sent when the external loop has capacity for another pallet. Once three pallets have been transferred to the external loop, the first conveyor may be full resulting in the immediate rejection of incoming pallets (*release*). Yet, this is not necessarily the case as up to three additional spaces can become available when pallets are processed by the assembly station, indicated by the τ-transitions in $I_2(\mathbf{EL}_3)$. Furthermore, automaton $I_2(\mathbf{EL}_3)$ models the restriction that pallets returning to the central loop can only be detected in the external loop (*el*) after they have been transferred into the external loop and processed by the assembly station.

The external loop interfaces cannot be modelled as command-pair interfaces [17], as it is the case for the assembly station and test unit interfaces. While notification events such as IsPalletCL and TransfToEL can be modelled using *low-data events* [18], this technique does not allow for requests such as QPalletAtCL and QPallet that are not always followed by an answer. HISC-CP interfaces are more flexible and support such behaviour. This makes it possible to replace previously proposed two-level hierarchies [17, 36] for the AIP model by the three-level hierarchy shown here.

Table 1 gives an overview of the subsystems of the AIP model and the time taken by Supremica [1] to verify them. It shows for each subsystem the number $|I|$ of interface automata and the number $|F|$ of plant and supervisor automata, as well as the number of reachable states of the composition of the local plants and supervisors and the interfaces of the next lower level (**States**). Then it shows the time taken by Supremica to verify local controllability (**LCont**) and interface consistency (**IConsist**). All experiments were run on a standard desktop computer with a 3.3 GHz CPU and 8 GB of RAM.

The three-level hierarchy model proposed here exhibits the same behaviour as its two-level precursor [36]. Both models permit up to seven pallets simultaneously in the central loop, which is the maximum possible while preventing blocking under the capacity restrictions. Yet, the two-level hierarchy [36] results in a high level with $2.6 \cdot 10^{10}$ reachable states and on the same computer takes 48 s to verify using BDDs and the HISC tool DESpot [16].

Table 1 shows that all HISC-CP properties are verified easily by Supremica in a matter of seconds. Local controllability is checked by explicit state enumeration; for the large top-level subsystem **AIP** it terminates early due to the absence of uncontrollable events. Interface consistency is verified by computing a conflict equivalent abstraction of the subsystem using the heuristic abstraction method [12], and comparing the result to the interface using the conflict preorder algorithm [39]. This reduces to a compositional conflict check [12] for the top-level subsystem **AIP**, which only uses the implicit interface I_{top}. As all checks pass, it is concluded that the system is globally controllable and nonblocking.

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. The feasibility of the framework has been demonstrated by modelling two manufacturing systems.

Previous HISC frameworks such as [18] are based on a master-slave relationship between subsystems, 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. The framework 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 choice of interface events is not restricted: an interface exists for every subsystem and set of interface events, although these interfaces may be nondeterministic and include silent transitions.

Experience with modelling the AIP system shows that HISC-CP interfaces are not always easy to design manually. The need to ensure that the interface is *more conflicting* than the subsystem can make it necessary to expose more detail about the subsystem to the higher levels than desired. Better interfaces may be possible by exploiting the property of answer events used in earlier HISC approaches [18], which must always be enabled by the higher levels. Taking this property into account requires changes to the conflict preorder and associated algorithms, which is an interesting topic of future research.

While previous HISC interface consistency conditions can be checked in polynomial time [7], the algorithms to check HISC-CP interface consistency are exponential [39], which may pose challenges when faced with large interfaces. Thanks to compositional minimisation algorithms [12, 27, 40], this was not found to be a problem for the large AIP model. These minimisation algorithms can also be used to automatically extract interfaces for legacy subsystems in polynomial time.

The HISC-CP design methodology employs automatic *verification*. The designer models the components of a subsystem, and then uses tools to verify local interface consistency. If verification is successful for all subsystems locally, it is guaranteed that the global properties of controllability and nonblocking are satisfied.

While it is possible to synthesise, i.e., automatically compute subsystems satisfying *low data interface consistency* [20], it is not clear whether the same can be achieved for the more general HISC-CP conditions. It is an interesting question for future research to investigate to what extent it is possible to synthesise least restrictive subsystems that are *less conflicting* than a given interface, and what the properties of such synthesis results would be.

References

1. Åkesson, K., Fabian, M., Flordal, H., Malik, R.: Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. In: Proc. 8th Int. Workshop on Discrete Event Systems, WODES '06, pp. 384–385. IEEE, Ann Arbor, MI, USA (2006)
2. de Alfaro, L., Henzinger, T.A.: Interface automata. In: Proc. 9th ACM SIGSOFT Int. Symp. on Foundations of Software Engineering 2001, pp. 109–120. Vienna, Austria (2001)
3. Barrett, G., Lafortune, S.: Decentralized supervisory control with communicating controllers. IEEE Trans. Autom. Control **45**(9), 1620–1638 (2000)
4. Brandin, B., Charbonnier, F.: The supervisory control of the automated manufacturing system of the AIP. In: Proc. Rensselaer's 4th Int. Conf. Computer Integrated Manufacturing and Automation Technology, pp. 319–324. IEEE Computer Society Press, Troy, NY, USA (1994)
5. Brandin, B.A., Malik, R., Malik, P.: Incremental verification and synthesis of discrete-event systems guided by counter-examples. IEEE Trans. Control Syst. Technol. **12**(3), 387–401 (2004). DOI 10.1109/TCST.2004.824795
6. Bruegge, B., Dutoit, A.H.: Object-Oriented Software Engineering Using UML, Patterns, and Java, 2 edn. Pearson Prentice Hall (2004)
7. Dai, P.: Synthesis method for hierarchical interface-based supervisory control. Master's thesis, Dept. of Computing and Software, McMaster University, Hamilton, ON, Canada (2006). URL `http://www.cas.mcmaster.ca/~leduc`
8. De Nicola, R., Hennessy, M.C.B.: Testing equivalences for processes. Theoretical Comput. Sci. **34**(1–2), 83–133 (1984). DOI 10.1016/0304-3975(84)90113-0
9. Fabian, M.: On object oriented nondeterministic supervisory control. Ph.D. thesis, Chalmers University of Technology, Göteborg, Sweden (1995). URL `https://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=1126`
10. Feng, L., Wonham, W.M.: Computationally efficient supervisor design: Abstraction and modularity. In: Proc. 8th Int. Workshop on Discrete Event Systems, WODES '06, pp. 3–8. IEEE, Ann Arbor, MI, USA (2006)
11. Feng, L., Wonham, W.M.: On the computation of natural observers in discrete-event systems. Discrete Event Dyn. Syst. **20**(1), 63–102 (2010)
12. Flordal, H., Malik, R.: Compositional verification in supervisory control. SIAM J. Control and Optimization **48**(3), 1914–1938 (2009). DOI 10.1137/070695526
13. Harel, D.: Statecharts: a visual formalism for complex systems. Sci. Comput. Programming **8**(3), 231–274 (1987)
14. Hill, R.C., Cury, J.E.R., de Queiroz, M.H., Tilbury, D.M., Lafortune, S.: Multi-level hierarchical interface-based supervisory control. Automatica **46**(7), 1152–1164 (2010). DOI 10.1016/j.automatica.2010.04.002
15. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
16. Leduc, R.: DESpot — unlock the DES potential (2011). URL `http://www.cas.mcmaster.ca/~leduc/DESpot.html`
17. Leduc, R.J.: Hierarchical interface-based supervisory control. Ph.D. thesis, Dept. of Electrical Engineering, University of Toronto, ON, Canada (2002). URL `http://www.cas.mcmaster.ca/~leduc`
18. Leduc, R.J.: Hierarchical interface-based supervisory control with data events. Int. J. Control **82**(5), 783–800 (2009). DOI 10.1080/00207170802291411
19. Leduc, R.J., Brandin, B.A., Lawford, M., Wonham, W.M.: Hierarchical interface-based supervisory control—part I: Serial case. IEEE Trans. Autom. Control **50**(9), 1322–1335 (2005)
20. Leduc, R.J., Dai, P., Song, R.: Synthesis method for hierarchical interface-based supervisory control. IEEE Trans. Autom. Control **54**(7), 1548–1560 (2009)
21. Leduc, R.J., Lawford, M., Wonham, W.M.: Hierarchical interface-based supervisory control—part II: Parallel case. IEEE Trans. Autom. Control **50**(9), 1336–1348 (2005)
22. Lin, F., Wonham, W.M.: Decentralized control and coordination of discrete-event systems with partial observation. IEEE Trans. Autom. Control **35**(12), 1330–1337 (1990)
23. Ma, C., Wonham, W.M.: Nonblocking Supervisory Control of State Tree Structures, *LNCIS*, vol. 317. Springer (2005)

24. Malik, R.: The language of certain conflicts of a nondeterministic process. Working Paper 05/2010, Dept. of Computer Science, University of Waikato, Hamilton, New Zealand (2010). URL `http://hdl.handle.net/10289/4108`
25. Malik, R., Flordal, H., Pena, P.N.: Conflicts and projections. In: Proc. 1st IFAC Workshop on Dependable Control of Discrete Systems, DCDS '07, pp. 63–68. Paris, France (2007)
26. Malik, R., Leduc, R.: Hierarchical interface-based supervisory control using the conflict preorder. In: Proc. 11th Int. Workshop on Discrete Event Systems, WODES '12, pp. 163–168. IFAC, Guadalajara, Mexico (2012)
27. Malik, R., Leduc, R.: Compositional nonblocking verification using generalised nonblocking abstractions. IEEE Trans. Autom. Control **58**(8), 1–13 (2013). DOI 10.1109/TAC.2013.2248255
28. Malik, R., Streader, D., Reeves, S.: Conflicts and fair testing. Int. J. Found. Comput. Sci. **17**(4), 797–813 (2006). DOI 10.1142/S012905410600411X
29. Mitchell, R., McKim, J.: Design by Contract, by Example. Addison-Wesley (2001)
30. Moor, T., Baier, C., Wittmann, T.: Consistent abstractions for the purpose of supervisory control. In: Proc. 52nd IEEE Conf. Decision and Control, CDC 2013, pp. 7291–7296. Firenze, Italy (2013)
31. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Communications of the ACM **15**(12), 1053–1058 (1972). DOI 10.1145/361598.361623
32. de Queiroz, M.H., Cury, J.E.R.: Modular supervisory control of large scale discrete event systems. In: Proc. 5th Int. Workshop on Discrete Event Systems, WODES '00, pp. 103–110. Ghent, Belgium (2000)
33. Ramadge, P.J.G., Wonham, W.M.: The control of discrete event systems. Proc. IEEE **77**(1), 81–98 (1989)
34. Rudie, K., Wonham, W.: Think globally, act locally: Decentralized supervisory control. IEEE Trans. Autom. Control **37**(11), 1692–1708 (1992)
35. Schmidt, K., Breindl, C.: Maximally permissive hierarchical control of decentralized discrete event systems. IEEE Trans. Autom. Control **56**(4), 723–737 (2011)
36. Song, R.: Symbolic synthesis and verification of hierarchical interface-based supervisory control. Master's thesis, Dept. of Computing and Software, McMaster University, Hamilton, ON, Canada (2006). URL `http://www.cas.mcmaster.ca/~leduc`
37. Teixeira, M., Cury, J.E.R., de Queiroz, M.H.: Local modular supervisory control of DES with distinguishers. In: Proc. 16th IEEE Int. Conf. Emerging Technologies and Factory Automation, ETFA '11, pp. 1–8. Toulouse, France (2011)
38. Wang, B.: Top-down design for RW supervisory control theory. Master's thesis, Dept. of Electrical Engineering, University of Toronto, ON, Canada (1995)
39. Ware, S., Malik, R.: A state-based characterisation of the conflict preorder. In: Proc. 10th Int. Workshop on the Foundations of Coordination Languages and Software Architectures, FOCLASA 2011, pp. 34–48. Aachen, Germany (2011). DOI 10.4204/EPTCS.58.3
40. Ware, S., Malik, R.: Conflict-preserving abstraction of discrete event systems using annotated automata. Discrete Event Dyn. Syst. **22**(4), 451–477 (2012). DOI 10.1007/s10626-012-0133-3
41. Wong, K.C., Wonham, W.M.: Hierarchical control of discrete-event systems. Discrete Event Dyn. Syst. **6**(3), 241–273 (1996)
42. Wong, K.C., Wonham, W.M.: Modular control and coordination of discrete-event systems. Discrete Event Dyn. Syst. **8**(3), 247–297 (1998)
43. Wong, K.C., Wonham, W.M.: On the computation of observers in discrete-event systems. Discrete Event Dyn. Syst. **14**(1), 55–107 (2004)
44. Wonham, W.M.: Supervisory control of discrete-event systems. Systems Control Group, Dept. of Electrical Engineering, University of Toronto, ON, Canada; at `http://www.control.utoronto.edu/DES/` (2009)