Hierarchical control of discrete-event systems

Lenko Grigorov

grigorov@cs.queensu.ca
School of Computing, Queen's University,
Kingston, Ontario K7L 3N6, Canada

June 29, 2005

Abstract

This paper is an overview of the current research on hierarchical control of discrete-event systems. Four major approaches are identified and described: bottom-up design, top-down design, state aggregation, and interface-based design. The research examined is grouped into these sections, in an attempt to unify the terminology and concepts where possible without an in-depth theoretical investigation. The motivation for this paper is to examine the different approaches to hierarchical control of discrete-event systems with an eye on their implications for the usability of the proposed models.

1 Introduction

Computerization is one of the major advancements in technology during the past few decades. Nowadays it is almost unthinkable to design and build a new machine without considering some kind of computer control for it. Even the use of inherently continuous systems like radio signals or chemical processes is "digitized" by the introduction of digital protocols and digital controllers.

The process of digitalization implies, at a low level, the discretization of the process or device which will be controlled. While Classical Control Theory deals with the control of continuous systems, a new approach is necessary for the control of discrete systems. In the eighties, Ramadge and Wonham publish a number of very influential articles [36, 50, 37]. They proposed a framework for the control of a class of discrete systems, called Discrete-Event Systems (DESs). We will introduce the framework and related matter next, followed by the motivation for this paper.

1.1 Preliminaries

Discrete-Event Systems are systems where events (changes of state) occur sequentially and asynchronously. For example, an elevator in a building can be modeled as a DES. The events would be the opening and closing of the doors, the pressing of the buttons, the arrival at

a floor, etc. The states of the system would include information about requests for a stop at different floors, the position of the elevator, the direction in which it moves, etc. Events occur sequentially: the model does not allow the simultaneous pressing of two buttons (in reality, the low-level event processing unit ensures the sequencing of events). Also, events are asynchronous because, for the purpose of the model, it does not matter how much time passes between two events. Even though an elevator is a much more complicated system, in reality, the electronic control of elevators uses a model similar to the one described.

There are many ways to model DESs, including Petri nets [34], fuzzy matrices [31], and modal logic [38]. However, the most commonly used and natural model is that of an automaton and, for practical purposes, a Finite-State Machine (FSM). The latter is also used in the Ramadge and Wonham framework. This approach is not only very intuitive, it allows the use of results from Automata Theory and, as the main interest of this paper, it is amenable to hierarchical structuring.

An FSM is a tuple $G = (\Sigma, Q, \delta, q_0, Q_f)$, where Σ is a finite set of symbols (also referred to as the "alphabet"), Q is a finite set of states, $\delta : Q \times \Sigma \to Q$ is a partial transition function, q_0 is the initial state and $Q_f \subseteq Q$ is the set of final states (frequently referred to as Q_m, the set of "marked" states). Note that in many of the articles discussed in this paper, FSMs are defined to have a set of initial states, Q_0, instead of a single initial state, q_0. Usually, during discussion, it is assumed that such FSMs are restricted to some initial state $q_0 \in Q_0$ as needed. The "empty symbol" ϵ, which is not in Σ, is used to denote a string of symbols with length zero. The notation Σ^* stands for the set of all finite strings of symbols from Σ and ϵ. The transition function δ can be naturally extended to $Q \times \Sigma^* \to Q$. Such an FSM can be interpreted as a DES if the states are considered to be states of the system and the symbols from Σ to be the events which can occur in the system. Thus, strings of symbols would describe sequences of events.

The language $L(G)$ is defined to be the set of all possible sequences of events in the system. The FSM G is said to generate $L(G)$. The language $L_m(G)$ is defined to be the set of all sequences of events which lead to a final state. The FSM G is said to accept $L_m(G)$. More formally,

$$L(G) = \{s \mid s \in \Sigma^*, \delta(q_0, s) \text{ is defined}\},$$
$$L_m(G) = \{s \mid s \in \Sigma^*, \delta(q_0, s) \text{ is defined}, \delta(q_0, s) \in Q_f\}.$$

The language $L(G)$ can be viewed as the unrestricted behavior of a DES and $L_m(G)$ as the sequences of events that accomplish a task, also called *marked strings*.

The length of a string s is denoted $|s|$. The string t is called a prefix of the string s, denoted $t \leq s$, if $\exists u \in \Sigma^*, s = tu$. The empty string ϵ is a prefix of any string. The *prefix-closure* of a language is defined to be the set of all prefixes of strings in the language. For all automata, prefix-closing the generated language produces a language equal to the generated language itself. More formally,

$$\overline{L} = \{t \mid t \in \Sigma^*, \exists s \in L, t \leq s\},\ \overline{L(G)} = L(G).$$

A *prefix-closed language* is a language which equals its prefix-closure. Prefix closure is an important operation because it describes the possible partial behavior of a DES. An FSM G is called *non-blocking* if $L(G) = \overline{L_m(G)}$. In other words, all string prefixes it can generate eventually lead to a final state. The non-blocking property is important because, when it is not satisfied, the DES may get "stuck" during runtime, i.e., reach a state from which a final state is not reachable.

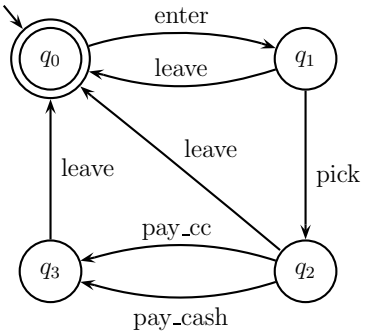


Figure 1: DES model of a customer in a store.

An example of a DES is the simplified model of a customer at a store [21] (Fig. 1). The customer can enter the store, pick something to buy, pay with cash or a credit card, and leave at any time. Here $\Sigma = \{$"enter", "pick", "pay_cash", "pay_cc", "leave"$\}$. The set of states is $Q = \{q_0, q_1, q_2, q_3\}$. The transition function can be determined from the diagram in Fig. 1, e.g., $\delta(q_1, \text{pick}) = q_2$. The initial state is q_0. The set of final states is $Q_f = \{q_0\}$. Examples of event sequences are "enter, leave" or "enter, pick, pay_cc". The second sequence is not "complete"—it does not belong to L_m. However, it belongs to $\overline{L_m}$, since it is a prefix of the sequence "enter, pick, pay_cc, leave", which is in L_m. This particular example is very simple, but one can easily imagine the application of DESs in factory processes, computer protocols, and other areas.

After having defined a DES, one of the questions of greatest interest is how one would be able to influence its unrestricted behavior. In other words, what restrictions would one use so that certain specifications on the behavior are met? The largest body of research on DESs deals with this specific problem: the control of DESs.

The basic FSM model does not provide any means of control. Thus, Ramadge and Wonham [36] extend it by distinguishing between *controllable* and *uncontrollable* events. Controllable events are events which can be "disabled", or prevented from occurring, and "enabled". Uncontrollable events remain enabled all the time. The sets of all controllable and uncontrollable events are denoted Σ_c and Σ_{uc}, respectively. Thus,

$$\Sigma = \Sigma_c \cup \Sigma_{uc},\ \Sigma_c \cap \Sigma_{uc} = \emptyset.$$

A specification for the desired behavior of a DES G is given as a language $K \subseteq L(G)$. The restriction of the complete behavior is done by disabling the controllable events when needed. This can be formalized by the construction of an FSM $S = (\Sigma, Q^S, \delta^S, q_0^S, Q_f^S)$, such that $K = L(S)$. Consequently, the controlled behavior of the DES, $L(S/G)$, can be obtained by intersecting the two languages: $L(S/G) = L(S) \cap L(G) = L(S \times G)$. This method is called *supervisory control* and S is termed a *supervisor*.

Unfortunately, control of DES is not such a trivial issue. Sometimes, the specification K would contain a string t such that an uncontrollable event can follow in the system while the specification does not permit it, i.e., $\exists s \in L(G), s = t\sigma, \sigma \in \Sigma_{uc}, t\sigma \notin K$. Let us consider the example in Fig. 1. The only controllable events are $\Sigma_c = \{$"pay_cash", "pay_cc"$\}$ (paying

with cash or a credit card, respectively). Imagine that the credit card reader is broken. Then, the desirable behavior from a customer would be $K = \{$"enter, leave", "enter, pick, pay_cash, leave"$\}$. This specification, however, cannot be implemented using supervisory control: the event "leave" is uncontrollable, thus, it cannot be disabled after the string "enter, pick". Despite our best intentions to prevent theft from the store, the underlying system does not have the necessary capability. This discussion leads to the following definition: a language K is called *controllable with respect to a system* G if and only if

$$\{s\sigma \mid s \in \overline{K}, \sigma \in \Sigma_{uc}, s\sigma \in L(G)\} \subseteq \overline{K}.$$

Controllability of a specification language is important because only in such a case can the required restrictions be implemented via supervisory control. In [50], the authors show that the class of all controllable sublanguages with respect to a DES, $\mathcal{C}(K, G) = \{L \mid L \subseteq K, L$ is controllable with respect to $G\}$, is a complete semilattice with respect to set union and has a supremal element. The largest controllable sublanguage of K with respect to G, $\sup \mathcal{C}(K, G)$ (also denoted $K^{\uparrow}$ where G is clear from the context), can be computed in polynomial time in terms of the number of states. In the example from Fig. 1, $\sup \mathcal{C} = \emptyset$ since, once we let a customer in the store, we cannot prevent theft. In most research papers, the specification language is assumed to be controllable and the interest lies in other aspects of control.

In many cases, a complete system is composed of separate modules which interact. This fact can be utilized when the system is modeled as a DES. *Modular control* of DESs, [51], uses the operation called *synchronous product* (also known as *parallel composition*) and denoted $||$ to compose DES systems (modules) into supersystems. For two systems, $G_1 = (\Sigma_1, Q_1, \delta_1, q_{01}, Q_{f1})$ and $G_2 = (\Sigma_2, Q_2, \delta_2, q_{02}, Q_{f2})$, the synchronous product is defined to be the automaton $G_1 \| G_2 = (\Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \delta, [q_{01}, q_{02}], Q_{f1} \times Q_{f2})$, where the states are elements of the Cartesian product of the sets of states of the two automata, the transition function δ is defined as $\delta : (Q_1 \times Q_2) \times (\Sigma_1 \cup \Sigma_2) \to Q_1 \times Q_2$,

$$\delta([q_1, q_2], \sigma) = \begin{cases} [\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)] & \text{if both } \delta_1(q_1, \sigma) \text{ and } \delta_2(q_2, \sigma) \text{ are defined,} \\ [\delta_1(q_1, \sigma), q_2] & \text{if only } \delta_1(q_1, \sigma) \text{ is defined and } \sigma \notin \Sigma_2, \\ [q_1, \delta_2(q_2, \sigma)] & \text{if only } \delta_2(q_2, \sigma) \text{ is defined and } \sigma \notin \Sigma_1, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

In other words, the modules interact, and are synchronized, through their common events. The parallel composition can be defined equivalently in linguistic terms. Let $P_{1,2} : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_{1,2}^*$ be the natural projections of strings from the combined alphabets to Σ_1^* and Σ_2^*, respectively. Then, $L(G_1) \| L(G_2) = P_1^{-1}(L(G_1)) \cap P_2^{-1}(L(G_2))$. Modular DES design is an elegant and convenient architectural approach, however, it has a significant drawback. In general, the state space of a supersystem may grow exponentially with the number of modules. Thus, the computation of a supervisor for the complete system becomes intractable even for moderately-sized real systems. The good result is that, under certain conditions, *local supervisors* can be constructed for each module such that their combined use ensures the global specification is met [51, 17].

Another extension to the basic model arises from real-world problems. The *observability* of events can be taken into account [30]. Some events from the DES model may not be observable to a supervisor. For example, if there is no signaling device at the door of a store, the cashier may not be able to observe the events "enter" and "leave" for the customers.

The alphabet Σ can be partitioned into Σ_o and Σ_{uo}, the sets of observable and unobservable events, respectively. Thus,

$$\Sigma = \Sigma_o \cup \Sigma_{uo},\ \Sigma_o \cap \Sigma_{uo} = \emptyset.$$

Notice that observability and controllability of events are not related, thus, events can be observable and controllable, observable but uncontrollable, etc. The natural projection $P : \Sigma^* \to \Sigma_o^*$ is defined as

$$\begin{aligned} P(\epsilon) &= \epsilon, \\ P(\sigma) &= \begin{cases} \sigma & \text{if } \sigma \in \Sigma_o, \\ \epsilon & \text{if } \sigma \in \Sigma_{uo}, \end{cases} \\ P(s\sigma) &= P(s)P(\sigma) \text{ where } s \in \Sigma^*, \sigma \in \Sigma. \end{aligned}$$

A language K is said to be observable with respect to L if and only if for all $s, s' \in K$ such that $P(s) = P(s')$

$$\forall \sigma \in \Sigma : s\sigma \in \overline{K} \wedge s' \in \overline{K} \wedge s'\sigma \in L \Rightarrow s'\sigma \in \overline{K},$$

$$s \in K \cap L_m(G) \wedge s' \in \overline{K} \cap L_m(G) \Rightarrow s' \in K.$$

In other words, if the projections of two strings look the same, their one-step continuations should be the same and, if both are marked, then either both belong to K or neither does. (If K is prefix-closed, the second requirement is trivially fulfilled.) This specification is important because otherwise it will not be possible to implement a supervisor for K that observes only $P(K)$. Indeed, when we allow for unobservable events, there exists a supervisor for a specification K if and only if K is controllable and observable [30].

Further information on discrete-event systems and their control can be found in [6, 25, 49].

1.2 Motivation

While the initial concepts of DES formalization and modeling seem to be simple and easy, real-world applications have been very scarce. In my opinion, this has two major reasons:

- the "hidden" computational complexity of DES and
- the user unfriendly model.

By "hidden" complexity I mean the following. The synthesis of a supervisor is polynomial in terms of the number of states of a DES. However, the number of states in a real system would normally be so large, e.g., 2^{20}, that the computation cannot be carried out using reasonable resources. This somewhat unexpected result stems from the exponential increase of the state space of a system with the composition of the subsystems from which the system consists. Much of the attention of DES control research is dedicated to the development of methodologies that avoid computations over the complete system. Big advancements have been made, with the introduction of modular control [51], online control [11], decentralized control [39] and, as we will explore in greater detail in this paper, hierarchical control [54, 2, 3, 46, 27].

Besides purely computational problems (for which some workarounds are already available), the standard DES model suffers greatly from a purely human-usability viewpoint. Despite the fact that intuition is not part of any precise formalisms, it plays a significant role

in people's cognitive processes [43]. This is recognized by Wonham, the pioneer of DES, who mentions in his recent informal presentations that greater transparency is needed for the inner workings of DES control [48]. The concept of a state machine with events as transitions seems very intuitive, and it is so as long as there are no more than a few states and transitions. A system with ten thousand states and thirty thousand transitions cannot be comprehended, let alone updated, if so desired. The synthesis of supervisors for such systems results in objects which, to the users, are nothing more than "magical boxes". Would somebody employ models which are computationally demanding, incomprehensible and hard to modify? The answer is, most probably not. Surprisingly, DES researchers continue devising increasingly complicated and obscure control paradigms and schemes while virtually no research is done for the provision of more intuitive interfaces to DESs. Fortunately, as a side effect, some of the research has resulted in more user-friendly DES designs, namely, the modular and hierarchical architectures. The modular approach breaks down the complexity of perception in a "horizontal" fashion, while the hierarchical approach breaks down the complexity in a "vertical" fashion.

Hierarchical structuring is important for complex control systems, as exemplified in the command hierarchies in the military or the government and as formally examined in [18]. Thus, I believe, hierarchical structuring is of major interest for the study of better human interfaces to DES. Modular control is also of importance, however, since its introduction in [51], it has been investigated in a relatively consistent way using the synchronous product (or a minor modification thereof) as a base. On the other hand, research of hierarchical control has produced a number of wildly varied approaches. Thus, it is of greater value to conduct an in-depth review of the latter. This paper will focus on hierarchical control of DESs, and hierarchical DESs in general.

1.3 Overview of hierarchical control of DESs

As already discussed, the study of hierarchical control of DESs is far from being unified around a common base. The first popular paper on the topic is by Zhong and Wonham [54], where the problem is stated as that of the generation of a *hierarchically consistent* abstraction of a DES. This approach is generally termed "bottom-up design" and further refinements are found in [47, 20, 35, 7, 24]. Soon after, Brave and Heymann publish a paper, [2], which uses a modified version of Statecharts, [22], to model hierarchical systems. This article does not generate enough interest until much later, in the work of Marchand and Gaudin [33, 19] and Ma and Wonham [32]. In 1995, Bing Wang defends a Master's Thesis [46] on a new approach termed "top-down design" for hierarchical DESs. A hierarchical *State Tree Structure* is employed which starts with a simple DES model and where subsystems are gradually refined. This initial research is also used in [32]. At about the same time, Caines and Wei publish an influential paper on lattices of hierarchical abstractions of state machines [3]. This work is extended in [4, 5, 41, 42, 23] and, later, in [1]. Leduc *et al.* propose an interesting method for hierarchical control, borrowing ideas from Object-Oriented Programming [27, 28, 29]. Low-level systems have interfaces and the communication between high level and low level is restricted to these interfaces. Cury and his research group publish an article on a novel approach to DES control [13] which is more abstract than the Ramadge and Wonham framework. This serves as the base for a natural extension to hierarchical control of DESs [15, 16, 44]. Other approaches to hierarchical control can be found in the

work of Chen and Lin [10] and Chen and Hanisch [8].

In the rest of the paper, the above approaches will be examined in greater detail.

2 Approaches to hierarchical control of DES

2.1 Bottom-up design

In the bottom-up approach, a high-level abstraction of a low-level system is made. Control is applied on the top level with the idea that the low level will perform accordingly.

2.1.1 Hierarchical control based on the standard framework

The bottom-up design concept for DESs first appeared in [53] and then in [54]. There, a two-level model is considered (see Fig. 2). At the low level, there is a system G_{lo} which is

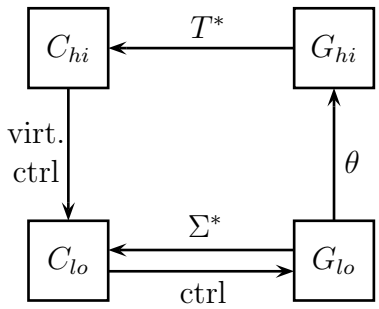


Figure 2: A two-level hierarchical system.

controlled by a low-level supervisor C_{lo}. The high-level abstraction of G_{lo} is called G_{hi} and the high-level supervisor for G_{hi} is called C_{hi}. The events at the high level are "reported" by the lower level through an information channel. In this way, the behavior of the high-level system is "driven" by the behavior of the low-level system. Furthermore, the high-level supervision is only virtual. The high-level supervisor has no direct influence over G_{hi}. Instead, the control decisions are communicated to the low-level supervisor which has to implement them on G_{lo}.

The low-level system $G_{lo} = (\Sigma, Q, \delta, q_0, Q_f)$ is a standard DES. The set of events at the high level is denoted T. The abstraction G_{hi} is defined using a *causal map* $\theta : L(G_{lo}) \to T^*$ such that

$$\begin{aligned} \theta(\epsilon) &= \epsilon, \\ \theta(s\sigma) &= \text{either } \theta(s) \\ &\quad \text{or } \theta(s)\tau \text{ for some } \tau \in T, \end{aligned}$$

where $s \in \Sigma^*, \sigma \in \Sigma$. The abstraction specification can be visualized conveniently if we imagine that the states of G_{lo} are *vocalized*. Let $\tau_0 \notin T$ be a "silent" event symbol and

$T_0 = T \cup \{\tau_0\}$. The *output function* $\omega : Q \to T_0$ is such that $\omega(q_0) = \tau_0$ and for any other state q reachable via the string $s\sigma$,

$$\begin{aligned} \omega(q) &= \tau \quad \text{if } \theta(s\sigma) = \theta(s)\tau, \\ \omega(q) &= \tau_0 \quad \text{if } \theta(s\sigma) = \theta(s). \end{aligned}$$

The interpretation is that each state of G_{lo} contains a high-level event which is output when the state is reached (see Fig. 3). The silent event τ_0 is ignored by the high level. Depending

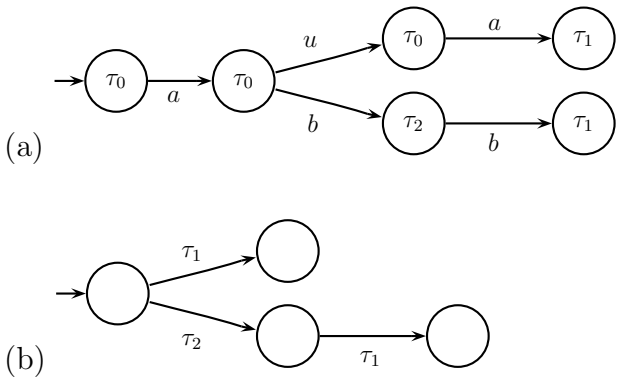


Figure 3: The low-level (a) and high-level (b) model of a hierarchical system. The output of the function ω is displayed in the states of the low-level system. Here $\Sigma_c = \{a, b\}$ and $\Sigma_{uc} = \{u\}$.

on the specific map θ, it might not be possible to construct a finite automaton to visualize ω. However, it is assumed that a practical choice of θ is made and that it leads to a finite representation. The high-level system G_{hi} is such that the language it generates equals the image of the low-level language under θ, i.e., $L(G_{hi}) = \theta(L(G_{lo}))$.

In order to be able to define a supervisor for G_{hi}, the alphabet T has to be partitioned in controllable and uncontrollable events T_c and T_{uc}. However, it can be seen easily that there may be no natural way how to derive this partitioning from Σ_c and Σ_{uc}. For example, in Fig. 3, assuming the given controllability of low-level events, the high-level event τ_1 is both "uncontrollable" in the string *aua* and "controllable" in the string *abb*. Thus, the following definition is introduced: (G_{lo}, θ) is *output-control-consistent* when, for all $\tau \in T$, it is unambiguous if τ is controllable or uncontrollable. If a hierarchical system (G_{lo}, θ) is not output-control-consistent, there is an algorithm to compute a new alphabet T' from T and then adjust θ accordingly so that ambiguous events τ are separated into τ_c and τ_{uc} and the system becomes output-control-consistent.

Let $E_{hi} \subseteq L(G_{hi})$ be a prefix-closed specification language which is controllable with respect to $L(G_{hi})$. Let $E_{lo} = \theta^{-1}(E_{hi})$. Then, if the system is output-control-consistent, there exists a control strategy C_{hi} over T such that its implementation in the low level by a supervisor C_{lo} results in $L(C_{lo}/G_{lo}) = E_{lo}^{\uparrow}$. Furthermore, $\theta(E_{lo}^{\uparrow}) \subseteq E_{hi}$. Intuitively, this means that, in reality, only a subset of the "expected" high-level specification may be achievable because of the low level control. A system (G_{lo}, θ) is called *hierarchically consistent*

when specifications "virtually" achievable at the high level can actually be achieved through low-level control, i.e., $\theta(E_{lo}^{\uparrow}) = E_{hi}$. A sufficient condition for hierarchical consistency is that of *strict output-control consistency*. A system is strictly output-control-consistent when it is output-control-consistent and the low-level system allows for the enablement and disablement of every controllable high-level event independently. If a system is output-control-consistent but not strictly so, there is an algorithm which introduces new high-level events and modifies θ so that additional information about the low level is available at the high level and the system becomes strictly output-control-consistent. If G_{lo} has n states and m transitions, the complexity of the combined algorithms (for output-control consistency and for strict output-control consistency) is $O(n^3m + n^2m^2)$ [49].

When the low-level system is not completely observable, i.e., $\Sigma_{uo} \neq \emptyset$, then strict output-control consistency is not sufficient to guarantee hierarchical consistency [24]. The notion of *H-observability* is introduced. Let $K \subseteq L(G_{lo})$. The language $E_{hi} \subseteq L(G_{hi})$ is H-observable with respect to $L(G_{hi})$ and K when, for all $s, s' \in K$ such that $P(s) = P(s')$, a version of the observability condition for high-level controllable events holds:

$$\forall \tau \in T_c : \theta(s)\tau \in \overline{E_{hi}} \wedge \theta(s') \in \overline{E_{hi}} \wedge \theta(s)\tau \in L(G_{hi}) \Rightarrow \theta(s')\tau \in \overline{E_{hi}}.$$

Then, if G_{lo} is strictly output-control-consistent, E_{hi} is H-observable with respect to $E_{lo}^{\uparrow}$ and $L(G_{hi})$ and $E_{lo}^{\uparrow}$ is observable with respect to $L(G_{lo})$, there exists a high-level control strategy over T such that its implementation in the low level results in E_{hi}.

2.1.2 Non-blocking hierarchical control

The approach discussed above does not guarantee non-blocking solutions when the specifications are not prefix-closed. In [47], Wong and Wonham generalize the methodology and present further conditions under which the hierarchical control will be non-blocking.

Instead of using controllable and uncontrollable events as the means to control the behavior of a system, the generalized concept of *control structures* is introduced. Let the set of prefix-closed sublanguages of a language L be denoted F_L. Then, $C : F_L \rightarrow 2^{2^L}$ is a control structure on L if, for all $H \in F_L$,

1. $C(H)$ is a complete upper semilattice with respect to 2^H and $\cup$,
2. $\emptyset, H \in C(H)$,
3. $K \in C(H) \Rightarrow \overline{K} \in C(H)$ and
4. $\forall I \in F_L, H \subseteq I : C(I) \cap 2^H \subseteq C(H)$ with equality if $H \in F_I \cap C(I)$.

The control structure assigns to every "subsystem" H a set of languages which are "controllable" using some control method. Every control structure is closed under union, is non-empty, is closed under prefix-closure and is harmonious with respect to subsystems (if H is a subsystem of I, it inherits all relevant controllable sublanguages of I; as well, if H is a controllable sublanguage of I, all controllable sublanguages of H are inherited from I). These axioms also guarantee that a supremal controllable sublanguage always exists. It is verifiable that the standard control mechanism in the Ramadge and Wonham framework [36] satisfies the requirements for a control structure.

Let θ, G_{lo} and G_{hi} be defined as before and C be a control structure on $L(G_{lo})$. The tuple (G_{lo}, θ, C) is called *control consistent* if there is a high-level control structure C' implied by

the low-level control structure, i.e., $C' \circ \theta = \theta \circ C$. The control structure C' exists if and only if the controllable sublanguages attached to low-level languages with the same high-level image also have the same high-level image:

$$\ker \theta \leq \ker(\theta \circ C).$$

If we assume that there is a finite number of prefix-closed low-level sublanguages, $n = |F_{L(G_{lo})}|$, a direct verification of this condition would have a complexity of $O(n^2m^2)$, where m is the number of prefix-closed sublanguages controllable with respect to $L(G_{lo})$. In [7], the authors propose a simplification of the computation when standard supervisory control structures are used. The kernel condition is split into three independently-verifiable conditions with a total computational complexity of $O(n^2m)$. Additionally, a number of weaker and less computationally expensive necessary conditions for control consistency are presented. They can be used to check if a system is not control-consistent.

In the generalized settings of control structures, control consistency plays the same role as hierarchical consistency in the standard supervisory control. Let $H \in F_L$ and $\kappa_H : 2^H \to C(H)$ be a mapping such that, for any sublanguage K of H, it gives the largest controllable sublanguage of K. In standard supervisory control, $\kappa_H(K)$ equals $K^{\uparrow}$ with respect to H. Let C and C' be control structures on $L(G_{lo})$ and $L(G_{hi})$, respectively. Then, control consistency means that the high-level supervisory solutions for high-level specifications are equivalent to the low-level supervisory solutions for the low-level images of the high-level specifications:

$$C'(E_{hi}) = \theta(C(E_{lo})) \Leftrightarrow \kappa_{E_{hi}} = \theta \circ \kappa_{E_{lo}} \circ \theta^{-1}.$$

In order to give conditions for a non-blocking hierarchical supervisory solution, the notion of *observer map* is defined. The map θ is called an observer for $K \subseteq L(G_{lo})$ when

$$\forall s \in K, \forall \tau \in T : \theta(s)\tau \in \theta(K) \Rightarrow \exists u \in (\Sigma^* \setminus \{\epsilon\}) : su \in K \wedge \theta(su) = \theta(s)\tau$$

In other words, "unobserved" evolution at the low level cannot change the evolution options at the high level: θ is refined enough to distinguish critical branching in the low level (see Fig. 4 for a counterexample). If $K = L(G_{lo})$, θ is simply called an observer.

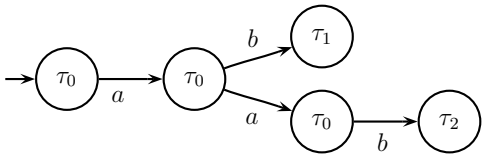


Figure 4: A counterexample for an observer map. Here $\theta(aa)\tau_1 = \epsilon\tau_1 = \tau_1 \in L(G_{hi})$ but there is no extension $u \in \Sigma^*$ of aa such that $\theta(aau) = \tau_1$.

If (G_{lo}, θ, C) is control-consistent, then

$$\kappa_{E_{lo}} \circ \theta^{-1} \text{ is non-blocking } \Rightarrow \kappa_{E_{hi}} \text{ is non-blocking .}$$

This means that the non-blocking property of a low-level supervisory solution is carried over to the high level. If this is to be valid in the reverse direction, an additional requirement

has to be imposed on θ. The system (G_{lo}, θ, C) is called *strongly observable* with respect to a set of languages $J \subseteq 2^{L(G_{lo})}$ if the observer property of θ is invariant for any language from J. Let X be the set of prefix-closed controllable sublanguages which are supremal with respect to their high-level images. Let there be consistency of marking between levels, $\theta^{-1}(L_m(G_{hi})) = L_m(G_{lo})$, and let (G_{lo}, θ, C) be strongly observable with respect to X, then,

$$\kappa_{E_{lo}} \circ \theta^{-1} \text{ is non-blocking } \Leftrightarrow \kappa_{E_{hi}} \text{ is non-blocking.}$$

A generalized version of the strict output-control consistency can be used for a simpler verification of strong observability with respect to X.

2.1.3 Multi-level modular hierarchical control

The hierarchical control discussed so far referred to a two-level hierarchy only. However, inherently, there is nothing which prevents the extension of this approach to a multi-level hierarchy. A study discussing such a design is [35].

For a language $K \subseteq \Sigma^*$, let $Voc(K) = \{s | s \in K, \omega(s) \neq \tau_0\}$ be the set of "vocal" (non-silent) strings of K and let θ_v denote the restriction of θ to vocal strings, $\theta_v = \theta|_{Voc(L(G_{lo}))}$. Then K is called *weakly observable* if θ is an observer for $Voc(K)$. The abstraction map θ is a *weak observer* if it is an observer for $Voc(L(G_{lo}))$.

Let C' be a control structure on G_{hi}. The system G_{hi} is called a *consistent abstraction* of G_{lo} if the following conditions are satisfied:

1. $L(G_{hi}) = \theta(L(G_{lo}))$,
2. $\forall t \in L(G_{hi}), \forall s \in \theta_v^{-1}(t) : t \in L_m(G_{hi}) \Rightarrow s \in L_m(G_{lo})$,
3. $\forall K \in C'(L(G_{hi})) : \exists$ a non-blocking supervisor $S, \theta(L(S/Voc(G_{lo}))) = K$.

In other words, G_{hi} is a high-level image of G_{lo}, the marking is consistent between levels and, for vocal strings, all the controllable behavior of the high level can be implemented by a low-level supervisor. The above conditions differ from the definition of hierarchical and control consistencies in some details: for example, conditions on the marking of strings are introduced, however, the control conditions are weaker.

The relation between the existence of consistent abstractions and the abstraction map θ is defined as follows: a consistent abstraction of G_{lo} exists if and only if θ is a weak observer. Furthermore, if θ is a weak observer, then a supremal (in terms of largest control structure) consistent abstraction of G_{lo} exists.

Let G_{hi} be a consistent abstraction. When a controllable high-level language $K \subseteq C'(L(G_{hi}))$ is considered, a supremal (in terms of low-level permissiveness and marking) low-level supervisor, S, exists such that $\theta(L(S/G_{lo})) = K$.

Thus far, hierarchical control of a single DES was considered. However, as was shown earlier, many times it is beneficial to use a composition of modules (DES subsystems). While it is possible to compute $G = G_1 \| G_2$ for some DESs G_1 and G_2 and then apply hierarchical control on G, it is more desirable to be able to work with each subsystem separately. In other words, abusing the notation, the case when $\theta(G_1 \| G_2) = \theta_1(G_1) \| \theta_2(G_2)$ is of interest. In terms of languages and marking, θ is distributive. However, the control structures are not preserved.

Let Σ_1, T_1 and Σ_2, T_2 be the sets of low- and high-level events, respectively, for two hierarchical systems (G_1, θ_1, C_1) and (G_2, θ_2, C_2). The disjunction of the event sets is assumed:

$\Sigma_1 \cap \Sigma_2 = \emptyset$ and $T_1 \cap T_2 = \emptyset$. Let C' be the high-level control structure implied by the control structure C of a low-level system. The control structure $C'(\theta_1(G_1))\|C'(\theta_2(G_2))$ for the parallel composition of the two high-level systems is defined as the intersections of the controllable languages in $C'(\theta_1(G_1))$ and $C'(\theta_2(G_2))$ amended with T_2^* and T_1^*, respectively. Generally,

$$C'(\theta(G_1\|G_2)) \subseteq C'(\theta_1(G_1))\|C'(\theta_2(G_2)).$$

In order to obtain the desirable result of distributiveness of the control structures across parallel composition (i.e., equality in the expression above), additional requirements have to be posed on the abstraction map.

The system G_{hi} is called a *reliable abstraction* of G_{lo} if it is a consistent abstraction and, for all strings (not only vocal), all the controllable behavior of the high level can be implemented by a low-level supervisor. If θ is an observer, a supremal reliable abstraction, denoted Θ, exists. Then,

$$C'(\Theta(G_1\|G_2)) = C'(\Theta_1(G_1))\|C'(\Theta_2(G_2)).$$

Having the above results, a multi-level hierarchical and modular control architecture can be introduced. At each level, a number of DESs can work in parallel, synchronized using the operation $\|$. Each module can be a simple DES, or an abstraction of a lower-level system. An example is shown in Fig. 5. In order to maintain consistency throughout such a composed system, a few requirements have to be satisfied. If there is a low-level specification for a module, it has to be applied before the high-level abstraction of the module is made. Also, in order to have consistent controllability for parallel modules, the abstractions of lower-level systems have to be reliable.

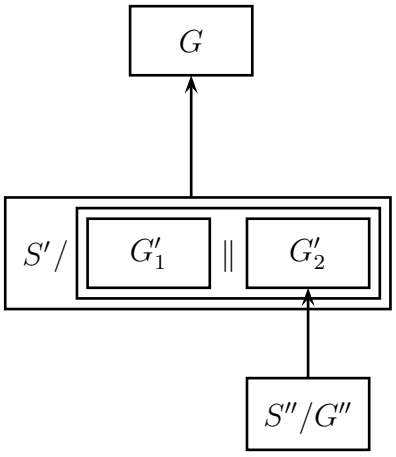


Figure 5: A hierarchical system composed of multiple modules. The arrows indicate reliable abstractions.

2.1.4 Hierarchical control based on Γ control structures

A recent approach to DES control was first popularized in [13] and presented in greater detail in [14]. The authors propose an advanced control structure for DESs, termed the Γ control structure. While this control enhancement applies to DESs in general, it can be used to streamline hierarchical control in particular.

Let L be a language over Σ. A Γ control structure is a map $\Gamma : L \rightarrow 2^{2^{\Sigma} \times \{M,N\}}$. For every $s \in L$, $\Gamma(s) = \{(\gamma, \#)\}$, where $\gamma \subseteq \Sigma$ is a set of enabled events after s and $\#$ is either M or N meaning the that when γ is selected s is considered a marked or a non-marked string, respectively. Thus, the control structure Γ offers a number of control options after each string in L. Further requirements on Γ are:

- $(\gamma_1, N), (\gamma_2, N) \in \Gamma(s) \Rightarrow (\gamma_1 \cup \gamma_2, N) \in \Gamma(s)$ and
- $(\gamma_1, M), (\gamma_2, \#) \in \Gamma(s) \Rightarrow (\gamma_1 \cup \gamma_2, M) \in \Gamma(s)$ where $\#$ can be either M or N.

These requirements express the fact that the control options are closed under union and that marking is consistent between control options. Let the union of two control options $(\gamma_1, \#_1)$ and $(\gamma_2, \#_2)$ be defined as $(\gamma_1, \#_1) \cup (\gamma_2, \#_2) = (\gamma, \#)$ where $\gamma = \gamma_1 \cup \gamma_2$ and $\# = M$ if and only if at least one of $\#_1$ and $\#_2$ equals M. The union of two sets of control options $\Gamma_1 \sqcup \Gamma_2$ contains control options $(\gamma, \#)$ such that there are $(\gamma_1, \#_1) \in \Gamma_1$ and $(\gamma_2, \#_2) \in \Gamma_2$ and $(\gamma, \#) = (\gamma_1, \#_1) \cup (\gamma_2, \#_2)$.

A supervisor for a DES with a Γ control structure is a map which simply selects an available control option after a string s. A Γ-*compatible language* is a language which can be fully achieved by restricting L using only the available control options in Γ. As a result, a supervisor for a language exists if and only if the language is Γ-compatible. If a specification language is not Γ-compatible, then a supremal Γ-compatible sublanguage exists and can be computed in $O(|\Sigma| n^2 m^2)$ steps, where n is the number of states in the automaton for L and m is the number of states in the automaton for the specification language. This result is comparable to the one in the standard Ramadge and Wonham framework.

Based on the above control scheme, the authors of [15] propose a method for hierarchical control of DESs. For every string $s \in L(G_{lo})$, let

$$L_{voc}(s) = \{v | v \in \Sigma^+, \omega(sv) \in T, \forall u < v : \omega(su) = \tau_0\}$$

be the set of shortest extensions of s that form a vocal string and

$$L_m(s) = L_{voc}(s) \cup \{v | v \in \Sigma^*, sv \in L_m(G_{lo}), \forall u < v : \omega(su) = \tau_0\}$$

be the set of shortest vocal extensions of s together with all non-vocal extensions of s which lead to a marked state. Let the set $\Gamma_{voc}(s)$ contain all control options $(\gamma, \#)$ such that there is controllable sublanguage of $L_m(s)$ or $L_{voc}(s)$ if $\# = M$ or $\# = N$, respectively, such that it accomplishes γ as the set of possible high-level events after s and it contains a marked string if and only if $\# = M$. In other words, $\Gamma_{voc}(s)$ contains all high-level control options for s which can be achieved at the low level. If this is a multi-level hierarchical system, controllability at the low level can be defined as Γ'-compatibility for some low-level control structure Γ'. There is a maximal high-level control structure Γ which is defined as

$$\Gamma(t) = \bigsqcup \Gamma_{voc}(s) \text{ for all } t \in L(G_{hi}), s \in \theta_v^{-1}(t). \tag{1}$$

Then, the system (G_{lo}, θ, Γ) is hierarchically consistent (every Γ-compatible high-level language can be achieved by a non-blocking supervisor at the low level). Furthermore, if every two different vocal strings have a different high-level image, the system is control consistent (every low-level non-blocking controllable language has a non-blocking controllable high-level image).

In [16], Γ control is discussed in the context of modular DES. Similar to [35] (see Section 2.1.3), the goal is to obtain consistent abstractions of two modules, G_1 and G_2 with $\Sigma_1 \cap \Sigma_2 = \emptyset$, such that the parallel composition of the two abstractions produces a consistent abstraction of the parallel composition of the modules. Let us denote Γ^c the maximal hierarchically consistent abstraction of G as discussed above. Then, $(G_1, \Gamma_1^c) \| (G_2, \Gamma_2^c)$ contains more control options than $(G_1 \| G_2, \Gamma^c)$. Thus, the notion of reliable abstraction is adapted to the Γ control settings:

$$\Gamma^r(t) = \bigsqcup \Gamma_{voc}(s) \text{for all } t \in L(G_{hi}), s \in \theta^{-1}(t).$$

Then, (θ, Γ^r) is called a reliable abstraction of G_{lo}. Note that this definition differs from (1) in using θ^{-1} instead of θ_v^{-1}. This results in Γ^r being a more restrictive control structure than Γ^c. However, reliable abstractions guarantee that $(G_1, \Gamma_1^r) \| (G_2, \Gamma_2^r) = (G_1 \| G_2, \Gamma^r)$. In other words, composing reliable abstractions of modules results in a reliable abstraction of the composition of the modules. The authors, however, propose a refinement of the control structure so that fewer control options get lost by moving from Γ^c to Γ^r. The following proof rule, called *assume and guarantee*, served as an inspiration:

$$\begin{aligned} &G_1 \cap E_2 \subseteq S_1 \text{ and } G_2 \cap E_1 \subseteq S_2 \\ &G_1 \subseteq E_1 \text{ and } G_2 \subseteq E_2 \\ &\Rightarrow G_1 \cap G_2 \subseteq S_1 \cap S_2, \end{aligned}$$

where intersection can be interpreted as parallel composition and inclusion as satisfying the property of hierarchically consistent abstraction. The "environment" E_2 for the system G_1 is defined as the high-level abstraction of G_2 where all events are uncontrollable and all strings are marked. In a similar way, E_1 is defined. Then, $(G_1 \| E_2, \Gamma_1^c) \| (G_2 \| E_1, \Gamma_2^c) = (G_1 \| G_2, \Gamma^c)$. Generally, this construction results in a more permissive control structure than Γ^r.

2.1.5 Discussion

The bottom-up design for hierarchical control is one of the best studied approaches to solving the problem. It can be seen that many advanced results have been obtained. The algebraic flavor to the works of [47] and [35] results in more elegant and consistent theories. The Γ control structures provide a generalized control framework which can also be applied to control problems other than hierarchical control, thus serving as a base for unification. Unfortunately, solving more complicated problems results in the use of more complicated methodologies. The proliferation of very abstract concepts, like the many kinds of "consistencies" between the two levels of hierarchical systems, makes the research very hard to follow. Furthermore, algorithms for the computation of some of the abstract constructs are not currently available. For example, it is not clear how to compute control structures directly (a key concept for most of the research). In the case of composed hierarchical systems, the current results are valid only when disjoint modules are considered. Thus, a lot of further research is necessary on this topic.

In my opinion, however, the greatest issue with bottom-up design is in the inherent dependence between levels. Using such an architecture, the high-level system is directly derived from and highly dependent on the low level. This help in synthesizing the control strategies, however, makes the modification of such systems a very demanding task. A small change at the bottom of the hierarchy may propagate all the way up to the very top level. In a dynamical production environment, where methods and goals change constantly, easy upgradeability of the control system is highly desirable. It is my belief that, no matter how advanced are the results using bottom-up architectures, managers would avoid deploying systems which need a complete makeover when even a small detail changes. In real life, it would not be acceptable to have to re-verify the control strategy for the whole manufacturing plant when a new machine feeder is installed.

2.2 Top-down design

The top-down design for hierarchical DESs is largely influenced by the work of Harel on the so-called *statecharts* [22]. A statechart is a finite-state machine which is enhanced by additional capabilities. Two of them are of special importance to the modeling paradigm discussed in this section: orthogonality (parallelism) and hierarchy. In other words, states can be composed into concurrent units and may consist of substates.

In the top-down architecture for DESs, the system model is recursively defined from the top (the highest-level system) to the bottom (the lowest-level subsystems). This is achieved by a process of refinement of coarse models to more detailed descriptions. The bottom-level subsystems need not represent the finest possible refinement—it may be that the system designer is not interested in a finer model. This approach is in some sense "dual" to the bottom-up design where, instead of refinement of coarse systems, abstraction of detailed systems is used.

2.2.1 Hierarchical State Machine Models

The first publication on hierarchical DESs modeled after statecharts was an article by Brave and Heymann [2]. They propose the use of hierarchical state machines (HSMs), a model derived from statecharts. An HMS is defined using a partial ordering function $\vdash$ on states which specifies a tree of state containment (the hierarchy). There is a unique state r such that for any state a, $r \vdash^* a$ (where $\vdash^*$ is the transitive closure on the relation $\vdash$). In general, states are divided into *OR states*, *AND states* and *basic states*. Basic states do not have substates while OR and AND states do. In OR states, the system can occupy only one of the substates at a time. In AND states, the system occupies all substates simultaneously (this models parallelism). Furthermore, for every HSM, there is an equivalent *canonical HSM* where every AND state contains only OR substates and the type of substates vertically across the hierarchy alternates between AND and OR. As in FSMs, transitions between states signify the occurrence of events, only transitions can be defined between levels, as well. Since there can be a number of states occupied at the same time, instead of states, it is convenient to talk about *configurations* of the state machine, i.e., tuples of mutually-orthogonal states. Every HSM has an equivalent representation as a standard "flat" DES modeled by an FSM. An HSM would be used instead of an FSM because the descriptive complexity of HSM is exponentially lower. For an example of an HSM, see Fig. 6.

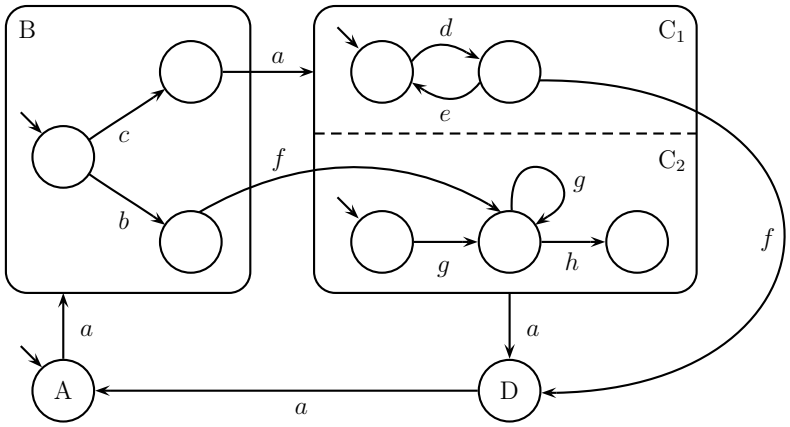


Figure 6: A hierarchical state machine. The state B is an OR state. The state C is an AND state and it contains two concurrent OR states, C_1 and C_2.

In [2], the authors consider *asynchronous HSMs*, i.e., HSMs where distinct substates cannot share the same event. This guarantees that the substates of an AND state cannot be synchronized via a mutual event. The first problem discussed is that of reachability in HSMs. Given a state a, a configuration q and a set of configurations P, the goal is to check if there exists a sequence of events in a which leads from q to a configuration $p \in P$. In order to compute this efficiently, for every state a, two sets are defined. The input set of a, $X_a(q)$, is the set of configurations of a which are reachable from q through parent states, $b \vdash a$. The output set of a, $Y_a(P)$, is the set of configurations of a from which P is reachable through parent states, $b \vdash a$. Thus, the computation of reachability can be viewed as the recursive problem of reachability between $X_a(q)$ and $Y_a(P)$ for relevant states a. The algorithm proposed for this calculation has a complexity of $O(m^2k^3n + mk^2n|P|)$, where m is the depth of the hierarchical structure, k is the branching factor of the structure (how many immediate substates a superstate has) and n is the number of basic states. The second issue considered is that of the forbidden configuration problem. Given a partition of the event set into controllable and uncontrollable events, a set of configurations F and a starting configuration q_0, the goal is to control the system in a way so that it never occupies a configuration from F. Let G_u be a version of the HSM G in which all controllable transitions are removed. Then, a solution to the problem exists if and only if there is no configuration in F which is reachable from q_0 in G_u. There is an algorithm to compute the maximally permissive supervisor which avoids the configurations in F with a complexity of $O(|Q|)$, where Q is the set of states. Furthermore, a related algorithm can be used to provide *online control* of the DES, similar to [11].

This line of work has been continued recently in [33, 19], where a similar model derived from statecharts is used. In this version of HSM, the most notable change is in the restriction of interactions between levels: transitions can no longer lead between states across multiple levels. Instead, higher-level transitions can lead only to the states of immediate substates and vice versa. Additionally, all concurrent substates of a lower level must have a single final

state and the "return" to the higher level is synchronized on these final states. Otherwise, substates are still considered asynchronous, i.e., they do not share common events. Again, every HSM system has an equivalent representation in the standard FSM form, which may result in exponential expansion of the number of states.

The major problem addressed in [33] is again the forbidden configuration problem. Given a set of basic states F, the goal is to design a non-blocking supervisor so that a state belonging to F is never reached. For a two-level system, an algorithm is proposed which relies on the use of the standard synthesis of a supervisor for each substate separately. Then, an adjustment is made to the high level so that entrance to low-level components is restricted as needed, and the standard synthesis is invoked again, this time for the high level. Additionally, the problem of optimal control is addressed. The optimal control paradigm follows [40]. Two types of cost are associated with each event $\sigma \in \Sigma$: cost of occurrence, $c_e : \Sigma \to \mathbf{R}^+$, and cost of disablement, $c_c : \Sigma \to \{0, \infty\}$. The cost of disablement is used to encode the partitioning of events into controllable (cost 0) and uncontrollable (cost ∞). Then, for a supervisor S, the cost of $L(S/G)$, $c(S)$, is defined as the supremal cost of all marked strings in the language and includes the costs of all events that need to be disabled by the supervisor along the way. An optimal supervisor is a supervisor S for which it holds that any other supervisor S', $c(S') \geq c(S)$. Furthermore, a *DP-optimal* supervisor is such that, for any prefix, the supervisor is optimal with respect to the continuations of the prefix. Again, an algorithm is presented which finds a DP-optimal supervisor for a two-level HSM by the application of the algorithm in [40] on every substate. Both of the proposed hierarchical algorithms can easily be generalized to multi-level systems. In [19], a refinement of the supervisor synthesis for the forbidden configuration problem is proposed, so that the set F, besides basic states, may include also state configurations. The approach used relies on the computation of weak forbidden configurations, that is, the configurations from which a string of uncontrollable events can lead to a forbidden configuration. This is done using an iterative process which performs only "local" computations at the lower level. Then, using the obtained configuration sets, local supervisors for each superstate can be found.

The major contribution of these works, in comparison to [2], is that the resulting control is non-blocking and the structure of the supervisor matches the structure of the supervised hierarchical system. Unfortunately, analysis of the computational complexity of the synthesis methodology is not provided.

2.2.2 State Tree Structures

In 1995, Bing Wang defended a Master's thesis on top-down design for DESs in the Ramadge and Wonham framework [46]. This work, despite its incompleteness, is widely cited and serves as an inspiration for more advanced theories. In it, a very structured method for the hierarchical refinement of DESs is proposed.

First, the notion of *state trees* is defined. A state tree is a tree of states (nodes), where each superstate is expanded in one of two possible ways: as an OR superstate or as an AND superstate. Furthermore, the substates of an AND superstate have to be OR superstates; that is, to have a more meaningful model. Figure 7 shows an example of a state tree.

Second, the notion of *holons* is defined. A holon is an FSM with the following amendments: the alphabet Σ is partitioned into two alphabets, Σ^B and Σ^I, the sets of boundary events and internal events, respectively; also, the set of states Q is partitioned into two sets Q^E and Q^I, the sets of external and internal states, respectively. The set of final states is empty

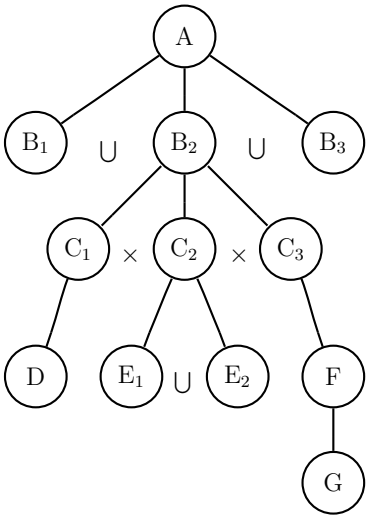


Figure 7: A state tree. Here '∪' identifies the siblings for an OR superstate and '×' identifies the siblings for an AND superstate.

(irrelevant) and $Q_0 \subseteq Q^E$. Furthermore, the transition function δ is defined to consist of three parts:

- $\delta^I : Q^I \times \Sigma^I \to Q^I$ (internal transitions),
- $\delta^{BI} : Q^E \times \Sigma^B \to Q^I$ (incoming transitions) and
- $\delta^{BO} : Q^I \times \Sigma^B \to Q^E$ (outgoing transitions).

The interpretation is that the internal states represent the "inner working" of the system, while the boundary states and the boundary transitions represent the interface to the rest of the world.

Finally, the *state-tree structure* (STS) is defined. A state-tree structure is a state tree where every superstate has a holon associated with it. Furthermore, the holons are properly connected at each level (their interfaces match) and, for every two holons at two different levels, their internal events are distinct (different levels deal with different events). An important drawback in the consistency of the theoretical foundation of this work is that the holon associated with an AND superstate is defined over the monolithic result of the parallel composition of all the OR substates of this state. In other words, concurrency is not preserved in the model.

A specification for an STS system is recursively given as local restrictions for every holon in the structure. If the specification for the system behavior is initially given in a standard non-hierarchical form, a procedure to convert it to an STS specification exists. Controllability and supremal supervisory solutions in the STS settings, however, require very complicated recursive definitions. The main problem is that the "interface" of a lower-level holon does not provide sufficient information about the relation between input and output boundary events. If one considers only the higher level, any pair of input and output events for a given superstate is achievable, and there is no relation between the controllability of the

input and output events. However, the inner structure of the lower-level holon may render some pairs of input and output events unfeasible or may allow for the disablement of a higher-level uncontrollable transition through lower-level restrictions. Thus, the algorithm proposed for the synthesis of a supremal STS supervisor for an STS specification starts from the top of the hierarchy and uses higher-level restrictions on lower levels when going down, combined with back-tracking to utilize lower-level control feedback in the design of higher-level control. For any STS specification, a supremal (most permissive) STS supervisor exists such that the language achieved under its supervision is a subset of the specification language. Unfortunately, the complexity of the involved algorithms is not provided.

This work is extended further by Ma and Wonham in [32]. The major shortcoming, namely, the inability to deal with AND superstates, is addressed. Furthermore, a much more elegant and consistent theoretic approach is used. Instead of using traditional tools to work with state machines, symbolic computation with binary decision diagrams is applied. Single transitions between specific states in the model are replaced by transitions between *sub state trees* [*sic*]. A sub state tree of a state tree T is obtained by removing some of the OR superstates from T. The set of all sub state trees of T is denoted $\mathcal{ST}(T)$. Let $T_1 \in \mathcal{ST}(T)$, then there is a transition on event σ to $T_2 \in \mathcal{ST}(T)$, where T_2 is defined as the result of making σ transitions in T_1 in all possible places. Conversely, a *backward transition function*, Γ, is defined, where $\Gamma(T_1, \sigma)$ is the largest sub state tree that can reach $T_1 \in \mathcal{ST}(T)$ on σ.

Every element of $\mathcal{ST}(T)$ can be encoded using a binary predicate which describes the structure of the sub state tree. The structure of each AND superstate is described as the conjunction of the structures of its substates and the structure of each OR superstate as the disjunction of the structures of its substates. For example, the sub state tree in Fig. 8(b) is described by the predicate $(v_{B_1} = \mathrm{C}_1) \wedge (v_{B_2} = \mathrm{D} \wedge v_D = \mathrm{F}_2)$, where v_X stands for the variable for node X. Note that the part of the predicate for B_3 is omitted since, in this sub state tree, trivially $v_{B_3} = 1$ (B_3 has a complete subtree).

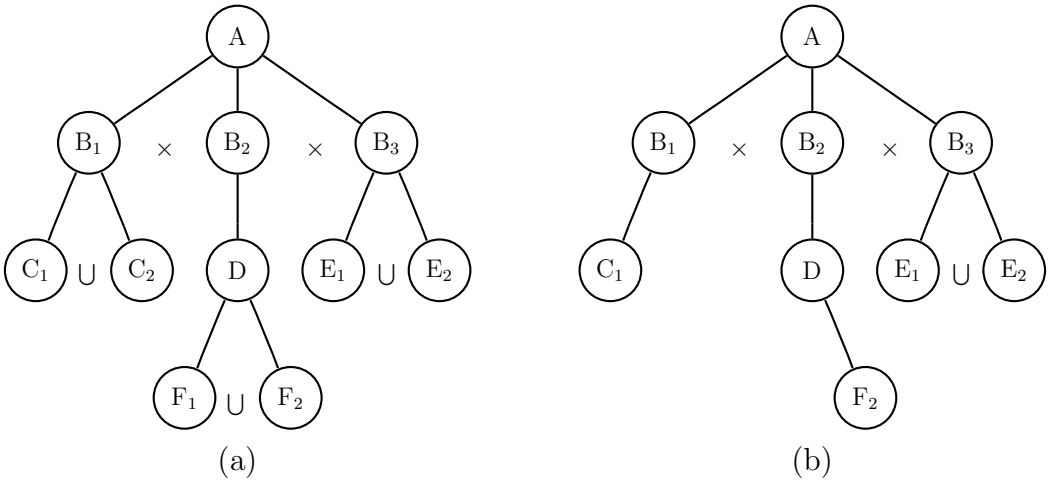


Figure 8: A state tree (a) and one of its sub state trees (b).

The Γ function is redefined to work on predicates. Then, the problem of enforcing a

specification S given as a set of illegal sub state trees of T can be cast as finding the largest non-blocking and controllable sub-predicate of the predicate $\neg P_S$, where P_S is the predicate corresponding to S. An example of this approach is provided in the computation of the *coreachability predicate*. Given a predicate P and a predicate P_m, the algorithm computes a predicate which contains all sub state trees which can reach P_m via sub state trees satisfying P.

Another work based on the top-down approach of Wang is [20]. The authors cast the problem in purely linguistic terms. The concept of *structured language* is introduced with a recursive definition where strings in each level k ($k = 0$ for the top-most level) consist of events $\sigma \in \Sigma_k$ and tokens $x \in X_k$. Tokens represent the languages of subsystems in the lower level. With each $x \in X_k$, a language $L_x \in (\Sigma_{k+1} \cup X_{k+1})^*$ and an output map $\theta_x : L_x \to 2^{\Sigma_k}$ is associated. The language represents the strings which can be generated by the subsystem, while the map provides the interface to the higher-level system (for each lower-level string, it gives the the higher-level events which can follow). Next, recursive definitions for language containment, join (union), meet (intersection) and controllability of structured languages are defined. According to these definitions, the set of all controllable structured sublanguages forms an upper semi-lattice with respect to containment and join. Thus, a supremal (largest) controllable structured sublanguage of any structured sublanguage exists. This result brings together this hierarchical approach and the original Ramadge and Wonham framework. The proposed computation of the supremal controllable structured sublanguage involves the use of a special operator on languages, Ω. When applied on a language, it removes the strings that are one-step uncontrollable. The operator is contractive and monotone and its fixpoints are controllable structured sublanguages. Thus, by iterative application on a structured language, Ω produces the supremal controllable structured sublanguage. The computational complexity of the application of the operator is not provided.

2.2.3 Discussion

Top-down design of hierarchical DESs is not as well studied as bottom-up design and, consequently, the theories relevant to it are not as advanced. The two major directions taken by researchers, statechart-derived models and recursive STS models, are similar in the use of system refinement, however, in many ways they differ. State tree structure models appear to be more structurally coherent, since the interactions between subsystems and between levels uses well-defined interfaces. The hierarchical state machines, especially the ones considered in [2], can have very "messy" transition structures and, even more so, supervisors. On the other hand, the introduction of STSs in [46] is incomplete and looks more like the result of "hacking" rather than rigorous theory building. It takes the work of Ma and Wonham, [32], to provide a theoretically more solid base. Further research is needed in the supervision of hierarchical systems with mutually-synchronized concurrent subsystems. As well, most works do not consider marking of strings in the language of a system. However, the biggest indication of the nascent nature of top-down design is the lack of complexity analysis of the many algorithms proposed by the different authors.

Even though Brave and Heymann published their work, [2], very soon after Zhong and Wonham [54], the number of publications dealing with top-down design is markedly smaller than the number of publications on bottom-up design. I find this lack of interest on the part of researchers a bit puzzling, since top-down design provides a very natural approach for managerial tasks. The main goal is specified and then it is refined into executable units.

Another good feature of recursive definition is that recursion can stop at any level as long as a condition is satisfied. In the case of supervisory control, for example, there is no need to provide specifications for subsystems whose behavior is already acceptable. Unfortunately, as Bing Wang shows, the interfaces between levels are not sufficient to provide good decoupling of the different levels. Feedback from lower levels is needed to guarantee that a higher level has correct assumptions about the real achievability of strings. Another disadvantage in recent works, [32], is the departure from intuitive representations of hierarchical DESs in order to provide more advanced and theoretically sound solutions.

2.3 State aggregation

Another line of work, very similar to the bottom-up design, is found in the work of Caines *et al.* [3, 4, 5]. This work is initially inspired by hybrid systems (systems whose different dynamics are modeled as either continuous or discrete) and thus it concerns states in discrete-event systems. Hierarchy is introduced by aggregating states in state partitions. The control specifications are not given linguistically; rather, by using conditions on the traversal of states.

2.3.1 Partition machines

The first article on partition machines cast in DES terms, [3], was published in 1995 and became much cited in this area of research. The set of states, Q, of an FSM G is partitioned into blocks $Q_i, i \in \{1 \dots n\}$ such that $\bigcup_i Q_i = Q$ and $i \neq j \Rightarrow Q_i \cap Q_j = \emptyset$. An important concept is that of *dynamical consistency*. An ordered pair of blocks Q_i and Q_j is said to be dynamically consistent (DC), denoted $\langle Q_i, Q_j \rangle$, if, for every state $q_i \in Q_i$, there is a path to some $q_j \in Q_j$ (the path traverses only states in Q_i before the last transition into Q_j). Then, a dynamically consistent partition machine M is an FSM such that the set of states $Q^M = \{Q_i | i \in \{1 \dots n\}\}$ and the transition function δ^M is defined so that there is a transition from Q_i to Q_j if and only if $\langle Q_i, Q_j \rangle$. Each transition is on a newly defined event which results in all transitions having unique events. The partition machine can be considered a high-level abstraction of the low-level (original) system. A multiple-level hierarchy can be built by considering the high level as a low level and doing another partitioning. Indeed, dynamical consistency can be defined with respect to a partition π, denoted $\langle Q_i, Q_j \rangle^\pi$. In this case, $\langle Q_i, Q_j \rangle = \langle Q_i, Q_j \rangle^{id}$, where id is the identity partition with $Q^M = \{\{q\} | q \in Q\}$.

Next, *controllability* is defined. In this context, controllability stands for reachability between states. A system is controllable if, for any two states q_1 and q_2, there is a string $s \in \Sigma^*$ such that $\delta(q_1, s) = q_2$. Then, a partition machine M of G is *in-block controllable* if each block considered separately is controllable, denoted $M \in IBCP(G)$, and is *between-block controllable* if the high-level structure is controllable.

Two operators which act on partitions are defined: chain union, $\cup^C$, and lower bound, $\sqcap$. Both preserve the $IBCP$ property of partitions. Thus, if $\leq$ is a partial order of partitions based on their fineness, $(IBCP(G), \sqcap, \cup^C, \leq)$ forms a lattice bounded below by the identity partition with $Q^M = \{\{q\} | q \in Q\}$ and bounded above by the trivial partition with $Q^M = \{Q\}$ (in case G is controllable).

The following can be concluded: if the state partition is in-block controllable (i.e., $M \in IBCP(G)$), then the high-level system is controllable at the high level if and only if the low-level system is controllable. In other words, an in-block controllable partition guarantees consistency of controllability between levels. Furthermore, let $\pi_1 \leq \pi_2$ be two partitions and

let $\pi_1 \in IBCP(G)$. Let Q_1 and Q_2 be two blocks in the partition π_2. Then, $\langle Q_1, Q_2 \rangle^{\pi_1}$ if and only if $\langle Q_1, Q_2 \rangle$. In other words, in order to verify dynamical consistency between two blocks, one need not resort to checking at the lowest level: it suffices to check using a finer but in-block controllable partition.

In [4], the above theories are extended to the so called *ST-systems* (source-target systems) which, translated to traditional DES settings, means FSMs with non-empty sets of initial and marked states, i.e., $Q_0, Q_f \neq \emptyset$. A system is called *strong ST-controllable* when for all $q_s \in Q_0, q_t \in Q_f$, there exists a string $s \in \Sigma^*$ so that $\delta(q_s, s) = q_t$. A *weak ST-controllable* systems is a system where for every source state, there is a string which leads to some target state. Similar to [46], the input and output states of blocks, $I(Q_i)$ and $O(Q_i)$, are defined as the states to which a transition from another block leads and the states from which a transition leads to another block, respectively. The states in Q_0 are also input states and the states in Q_f are also output states.

The other concepts from [3] are also updated to reflect the ST setting. The ST version of dynamical consistency is defined as follows: $\langle Q_i, Q_j \rangle$ if for each $q_s \in I(Q_i)$ there is a string $s\sigma$ such that $\delta(q_s, s) \in O(Q_i)$, passing only through states in Q_i, and $\delta(q_s, s\sigma) \in I(Q_j)$. Then, an ST partition machine is a machine where transitions between blocks exist if and only if the states form an ST dynamically-consistent pair. In-block controllability is defined as follows: for every block, there exists a path from each input state to some output state and each output state is reachable from any other output state. Again, as in [3], the ST version of $IBCP$, $ST\text{-}IBCP$, forms a lattice of hierarchical aggregate models of the base system G.

Given two ST systems $M_1, M_2 \in ST\text{-}IBCP(G)$ such that $M_1 \leq M_2$, then M_2 is ST between-block and ST in-block controllable with respect to M_1 as a base machine. Also, every ST controllability problem at the high level can be decomposed and recursively defined as block-to-block locally solvable ST controllability problems at the lower levels.

An application of the hierarchical design described above is presented in [41, 42]. The authors utilize the $IBCP$ lattice to speed up the computation of an optimal path between a source and a target state. Let a low-level cost function $c : Q \times \Sigma \to (0, \infty)$ be defined which assigns a positive cost to events leaving states. The cost of a path between two states is defined as the sum of the costs of all state-event pairs along the path. A path between two states is called *optimal* if there is no other path between the states which has a smaller cost. A hierarchy (G, M), where M is a partition machine of G, is *optimality consistent*, $\mathrm{OC}(C)$, if there is a high-level cost function C such that:

- for all states $(q, r) \in Q_i \times Q_j$, for all low-level optimal paths u between q and r, there is a high-level optimal path between Q_i and Q_j which contains u and
- for all blocks Q_i and Q_j, for all high-level optimal paths U between Q_i and Q_j, for all states $(q, r) \in Q_i \times Q_j$ there is a low-level optimal path between q and r which is contained in U.

Let the following high-level costs functions be defined:

- $D^+(Q_i, Q_j)$ equals the most costly path among all low-level optimal paths between states in Q_i and Q_j,
- $D^{+/-}(Q_i, Q_j)$ equals the most costly path among all low-level optimal paths between states in Q_i and the corresponding states that are least costly to reach in Q_j and
- $D^-(Q_i, Q_j)$ equals the least costly path among all low-level optimal paths between states in Q_i and Q_j.

Two dynamical programming algorithms are presented which attempt to find a possibly sub-optimal low-level path between a source and a target state. Both algorithms first generate a high-level optimal path. Then, the *P-HADP* algorithm searches for a low-level optimal path within the blocks from the high-level path, while the *HADP* algorithm searches for a low-level optimal path which traverses the partition blocks in the same way that the high-level path does. If the hierarchical system is OC($D^{+/-}$), the *P-HADP*($D^{+/-}$) algorithm produces an optimal solution. If, for all dynamically consistent pairs $\langle Q_i, Q_j \rangle$, $D^{+/-}(Q_i, Q_j) = D^{-}(Q_i, Q_j)$, then the *HADP*($D^{+/-}$) algorithm finds an optimal solution. Furthermore, estimates of the sub-optimality of solutions in general are provided. The *HADP* algorithm is applied to the Broken Manhattan Grid class of problems with a significant acceleration in the computing while the solutions remain only marginally sub-optimal. In this case, the hierarchical structure of the system is used to advantage: a high-level solution is gradually refined as per the levels in the hierarchy. This results in sub-optimal solutions, however, at much smaller computational cost. An estimate of the increase of efficiency of the optimal-path search is bounded below by $O(\sqrt{n})$, where n is the number of states in G.

An extension of the partition-machine hierarchical control to a setup closer to the original Ramadge and Wonham framework is presented in [5, 23]. In these works, events are considered to be controllable and uncontrollable. Accordingly, the notions of dynamic consistency, in-block controllability, etc. have to be redefined appropriately.

First, *trace-DC* is defined. Since, unlike before, there are uncontrollable transitions, two types of trace-DC relations are defined. The relation $\langle Q_i, Q_j \rangle_u$ holds when, for every state in $I(Q_i)$, there is an uncontrollable string $s\sigma \in \Sigma_{uc}^*$ such that s traverses only states in Q_i and σ leads to Q_j. Conversely, the relation $\langle Q_i, Q_j \rangle_d$ holds when, for every state in $I(Q_i)$ there exists a string $s\sigma \in \Sigma^*$ such that s traverses only states in Q_i and σ leads to Q_j and all strings s contain at least one controllable event. These two relations cannot hold simultaneously. A trace-DC partition machine is such that there is a high-level uncontrollable transition V_i^j from Q_i to Q_j whenever $\langle Q_i, Q_j \rangle_u$, there is a high-level controllable transition U_i^j whenever $\langle Q_i, Q_j \rangle_d$ and there are no other high-level transitions. Trace-DC partitioning implies output-control consistency (see Section 2.1.1, [54]). Based on the partitioning, a natural map $\Theta_\pi : \Sigma^* \to \Sigma_\pi^*$ can be defined, where Σ_π stands for the high-level alphabet. The map Θ_π reports the crossings of the boundaries between blocks. It is an observer in the sense of [47] (see Section 2.1.2) and a trace-DC system is control consistent in the same sense.

The controllability of a high-level system depends on the actual control that can be exercised by the low-level system. Thus, it is important that high-level controllable transitions can be enabled and disabled independently using low-level control. A subset of states $R \subseteq Q$ is called *non-blocking-controllable* if all states in R are reachable via states in R, there are no uncontrollable events leading to states not in R and, from every state in R, a final (target) state can be reached via states in R. A partition machine is called *non-blocking IBC* if, for every pair $\langle Q_i, Q_j \rangle_d$ and all $q \in I(Q_i)$, there is a non-blocking controllable set

$$R_i^j \subseteq Q_i \cup I(Q_j) \bigcup_{\langle Q_i, Q_k \rangle_u} I(Q_k)$$

and, if $Q_i \cap Q_m \neq \emptyset$, there is a non-blocking controllable set

$$R_i^m \subseteq Q_i \bigcup_{\langle Q_i, Q_k \rangle_u} I(Q_k),$$

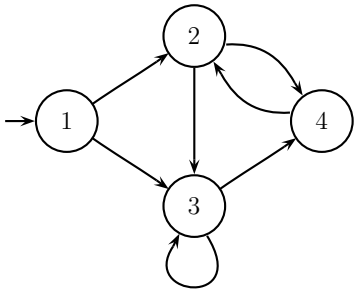
such that $q \in R_i^j, q \in R_i^m$ and $R_i^m \cap Q_m \neq \emptyset$. In other words, every block of the partition machine allows for independent non-blocking control of high-level transitions. An algorithm, High-Low Synthesis (HLS), is proposed which translates a high-level supervisor to a low-level supervisor by restricting the low-level behavior to the sets R_i^j and R_i^m as needed. Let K be a non-blocking and controllable high-level language and the result of the HLS algorithm for K be denoted $K_{HLS}(K)$. Provided that a partition is non-blocking IBC, $K_{HLS}(K)$ is a non-blocking and controllable language and $\Theta_\pi(K_{HLS}(K)) = K$. Furthermore, it can be concluded that the hierarchical system (G, Θ_π) is hierarchically consistent in the sense of [54] (see Section 2.1.1). Unfortunately, the hierarchical consistency of a system (G, Θ_π) does not guarantee that there is a non-blocking IBC partitioning of G. It may be necessary to split some states of G to facilitate the partitioning. A vocalized lifting (VL) algorithm is proposed to create a non-blocking IBC partition of G if (G, Θ_π) is hierarchically consistent.

The authors in [23] give a rough estimate of the computational advantage of using this hierarchical model for the synthesis of supervisors as opposed to using the low-level system solely. Let K be a high-level specification, K_{lo} be the low-level image of K, m be the number of states of an automaton for K, N be the number of states in the low-level system and n be the number of states in the high-level system (the cardinality of the partition). Then, to compute $K^\uparrow$ and then the low-level implementation of the supervisor would cost roughly $n^2m^2 + N^4/n^3$, while the computation of $K_{lo}^\uparrow$ would cost roughly $(N^4m^2)/n^2$. When $m = n = \sqrt{N}$, the ratio is $1/(N\sqrt{N})$ in advantage for the hierarchical approach.

2.3.2 State aggregation based on matrices

Another approach to state aggregation for DESs is described in [1]. There, the whole problem is cast in boolean matrix algebra which results in an efficient method for state partitioning of large systems.

Let the states of a DES system G be ordered, $q_1, q_2, \ldots, q_n$, where $n = |Q|$. Then, the one-step reachability of the system can be represented by an binary adjacency matrix $n \times n$, A, such that there is a 1 in A_{ij} if there is a transition from q_i to q_j, otherwise $A_{ij} = 0$. See Fig. 9 for an example. A permutation of the matrix is a matrix where the columns and



$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Figure 9: A DES and the corresponding adjacency matrix

rows have been shuffled (i.e., the indexes form a permutation of the original indexes). The

adjacency matrix A is called *reducible* if there is a permutation matrix P such that

$$P^t A\, P = \begin{pmatrix} B & C \\ 0 & D \end{pmatrix},$$

where B and D are square matrices and boolean algebra is used. If B or D are reducible, this process can be repeated on them, thus finally yielding an irreducible matrix R. A more intuitive interpretation of the blocks on the diagonal is as follows. Each block signifies a set of states in G. The states in blocks closer to the bottom of the matrix do not have transitions leading to states in blocks closer to the top of the matrix. Another property of the matrix A is that A^k (the k-th power of A) signifies the k-step reachability between states.

The reduction of the adjacency matrix A will, indeed, result in an aggregation of the states of G in blocks where each block is only one-way accessible from some other block(s) and all states in the block are mutually reachable. An algorithm to obtain the irreducible matrix R of A is proposed. Its complexity is of the order $O(\log n)$ matrix manipulations.

A nice side result from the reduction of A is that the grouping of the states of G forms an $ST\text{-}IBCP$ partition. Furthermore, this aggregation algorithm can be used for a solution to the forbidden configuration problem in the ST settings or the Ramadge and Wonham settings. Let $F \subseteq Q$ be the set of forbidden states. The goal is to control the system in a way that it reaches a target state from a source state and it never enters a state from F on the way. First, a sub-automaton of G is considered where the surviving states, $Q_u \subseteq Q$, are either sources or targets of an uncontrollable transition and only the uncontrollable transitions remain. The adjacency matrix of the sub-automaton is reduced, resulting in an aggregation where the states in a block are mutually reachable through uncontrollable events and thus can be considered "equivalent" in terms of control. The blocks connected to a block containing a forbidden state can be grouped with it since they can lead uncontrollably to the forbidden state and thus are equivalent. Then, the original automaton is reinstantiated by replacing the states in Q_u with the corresponding blocks. As a result, all transitions from states in $Q \backslash Q_u$ to the blocks are controllable and thus a supervisor will be able to prevent the entrance to blocks containing forbidden states. If the source or target states belong to a block with forbidden states, there is no supervisory solution to the problem. Unlike the traditional $ST\text{-}IBC$ partitioning, the aggregation proposed here is invariant with respect to the source and target states.

A problem with the proposed aggregation algorithm is that some graphs may be irreducible to start with (e.g., fully connected graphs). Thus, the problem becomes one of finding a decomposition of an irreducible matrix A into irreducible blocks. There are many approaches that can be taken, however, the authors chose to use the method of cutting asymmetric transitions. Let A^s be the symmetric matrix of A (where $A^s_{ij} = 1$ if and only if $A_{ij} = A_{ji} = 1$). Then, if the matrix A^s is reducible, there is an algorithm to compute an aggregation such that the blocks are as big as possible and have comparable sizes. The algorithm takes $O((logn)^2)$ steps.

2.3.3 State aggregation with Γ-control

In [44], the authors propose a hierarchical control approach which combines the ideas of Γ control structures [13] (see Section 2.1.4) and state aggregation. The system at the low level, G, is given in the standard FSM way, while the high-level aggregate model is enhanced with a Γ control structure.

A subset of the events, $\Sigma^A \subseteq \Sigma$, is selected as the significant events for the high level. Then, a partition π of the states of G is made such that $q_1, q_2 \in Q_i$ if and only if there is a string of non-significant events $s \in (\Sigma \backslash \Sigma^A)^*$ between them. The transition function $\delta^A : \pi \times \Sigma^A \to \pi$ of the high level is defined as $\delta^A(Q_i, \sigma) = Q_j$ if there are states $q_1 \in Q_i, q_2 \in Q_j$ such that $\delta(q_1, \sigma) = q_2$. Let Θ be the natural projection from Σ^* to Σ^{A*}.

For every block Q_i and every state $q_j \in I(Q_i)$, two sets of languages are defined. The set C_{ij} contains all supremal controllable sublanguages within Q_i with initial state q_j that contain strings reaching a marked state or leading out of Q_i. The set D_{ij} contains all supremal controllable sublanguages within Q_i with initial state q_j that contain strings leading out of Q_i and do not pass through a marked state inside Q_i. Using these sets of languages, a Γ control structure for the high-level system can be constructed: for every block Q_i and input state q_j, Γ_{ij} contains control options equal to $\Theta(K)$ for all $K \in C_{ij} \cup D_{ij}$ (the possible sets of events leading out of Q_i) and, for every K which contains a marked string, the corresponding control option is marked. This definition constructs a valid Γ control structure. Furthermore, there exists a high-level Γ-compatible sublanguage $E_{hi} \subseteq \Theta(L(G))$ if and only if there exists a controllable $E_{lo} \subseteq L(G)$ such that $\Theta(E_{lo}) = E_{hi}$. An algorithm is proposed for the synthesis of the low-level supervisor given a high-level supervisor. It makes use of the languages in C_{ij} and D_{ij} for every block and can be used either offline (to compute the complete low-level supervisor) or online (to compute local low-level supervisors for every block that is reached).

2.3.4 Discussion

The state-based hierarchical modeling in the form of partition machines was originally inspired by hybrid systems, and thus the first articles published on the topic have a much different flavor than that of the standard Ramadge and Wonham framework. Controllability is initially considered as the existence of paths between states; only later, in [5], controllable and uncontrollable events are introduced. While the first results seem very crude, recently the theory is greatly advanced [23, 1]. Nevertheless, many open questions remain. The greatest problem is that of the efficient computation of IBC partitions: not all $IBCP$ lattice elements can be computed directly, while the non-blocking IBC partitions in [23] do not form a lattice at all. Furthermore, even if IBC partitions are available, there is no analysis as to what properties of the partitions are desirable and how many levels are to be used in a multi-level hierarchy to achieve optimal performance. In [1], reduction of the adjacency matrices of DES systems is considered, however, only initial research is presented for the case when a matrix is irreducible. The research on Γ control structures does not indicate how the approach can be extended to multi-level hierarchies.

In my opinion, partition machines form a very intuitive hierarchical modeling paradigm. The aggregation of states and the definition of transitions seem to be consistent with what people would naturally expect. However, the paradigm has a significant drawback in that the actual event occurring in the DES system take a back seat in the models. Indeed, in the aggregate models, most (or all) high-level transitions occur on unique events and there is no inherent relation between high-level and low-level events. While connectivity between states is well preserved, the high-level sequences give no indication as to what—even approximately—is happening in the system (in the low level). If one is interested only in states of the system and not in the specific evolution of the system behavior, partition machines may provide a viable supervisory methodology. However, if a more traditional viewpoint is used, where the actual event occurrences are also important, this approach needs greater reconciliation with

the rest of DES control theory.

2.4 Interface-based design

Recently, the work of Leduc *et al.* [27, 28, 29] introduced a novel approach to hierarchical DESs. Unlike previous approaches where different levels of the hierarchy are more or less closely coupled, in this one, a strict decoupling of the levels is achieved through the use of well-defined interfaces. This is in line with the methods which nowadays are applied most successfully in the design of complicated systems such as circuitry and software and is, indeed, inspired by them.

2.4.1 Interface-based supervisory control

In [27, 28], a master-slave hierarchical DES is considered. The high-level system, G_{hi}, requests services from the low-level system, G_{lo}, through an interface, G_I, modeled as a DES. The alphabet of the complete system, Σ, is partitioned into the set of high-level events, Σ_H, the set of low-level events, Σ_L, the set of request events, Σ_R, and the set of answer events, Σ_A. These sets are mutually exclusive. The set $\Sigma_I = \Sigma_R \cup \Sigma_A$ contains the events for the interface. The system G_{hi} is defined over Σ_H and Σ_I; G_{lo} is defined over Σ_L and Σ_I; and G_I is defined over Σ_I. Thus, the interaction between the high level and the low level is restricted by the interface. A *command-pair* interface has the following restrictions. First, it alternates requests and answers. In other words, $L(G_I) \subseteq \overline{(\Sigma_R\Sigma_A)^*}$. Second, the only marked strings are the empty string and the strings ending with an answer, i.e., $L_m(G_I) = (\Sigma_R\Sigma_A)^* \cap L(G_I)$. The complete system is $G = G_{hi} \| G_{lo} \| G_I$. For a picture of an interface-based hierarchical DES, see Fig. 10.

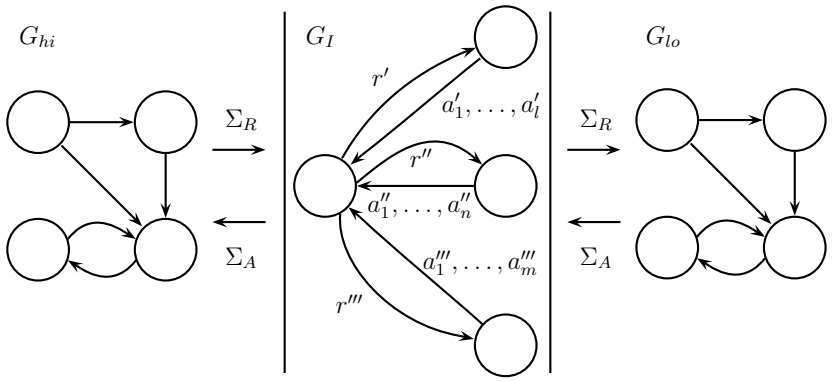


Figure 10: An interface-based hierarchical system.

Let S_{hi} and S_{lo} be supervisors for the high and the low level. Let $\Sigma_{HI} = \Sigma_H \cup \Sigma_I$ and $\Sigma_{LI} = \Sigma_L \cup \Sigma_I$. The natural projections from Σ^* to Σ_I^*, Σ_{HI}^* and Σ_{LI}^* are denoted P_I, P_{HI}

and P_{LI}, respectively. Let

$$\begin{array}{rclcrcl} \mathcal{H} & = & P_{HI}^{-1}(L(S_{hi}/G_{hi})), & & \mathcal{H}_m & = & P_{HI}^{-1}(L_m(S_{hi}/G_{hi})), \\ \mathcal{L} & = & P_{LI}^{-1}(L(S_{lo}/G_{lo})), & & \mathcal{L}_m & = & P_{LI}^{-1}(L_m(S_{lo}/G_{lo})), \\ \mathcal{I} & = & P_{I}^{-1}(L(G_I)), & & \mathcal{I}_m & = & P_{I}^{-1}(L_m(G_I)), \\ L & = & P_{LI}^{-1}(L(G_{lo})), & & L_S & = & P_{LI}^{-1}(L(S_{lo})), \\ H & = & P_{HI}^{-1}(L(G_{hi})) & \text{and} & H_S & = & P_{HI}^{-1}(L(S_{hi})). \end{array}$$

A system is called *serial interface consistent* if the following properties hold:

1. the high level and the low level only share interface events;
2. the FSM G_I is a command-pair interface;
3. the high level must always accept an answer event if it is eligible in the interface;
4. the low level must always accept a request event if it is eligible in the interface;
5. immediately after a request event occurs, in the low level there exist strings that reach all answer events eligible for the request and
6. every string marked by the interface can be extended by a low-level string such that it is marked in both the interface and the low level.

A system is called *serial level-wise non-blocking* if $\mathcal{H} \cap \mathcal{I} = \overline{\mathcal{H}_m \cap \mathcal{I}_m}$ and $\mathcal{L} \cap \mathcal{I} = \overline{\mathcal{L}_m \cap \mathcal{I}_m}$. In other words, the high and low levels restricted by the interface are independently non-blocking. If a system is serial interface consistent and serial level-wise non-blocking, then the complete system $G = G_{hi} \| G_{lo} \| G_I$ is non-blocking.

The hierarchical specification given by S_{hi} and S_{lo} is called *serial level-wise controllable* if the following holds:

1. the alphabets for S_{hi} and S_{lo} are Σ_{HI} and Σ_{LI}, respectively;
2. $\{s\sigma | s \in L_S \cap \mathcal{I}, \sigma \in \Sigma_{uc}, s\sigma \in L\} \subseteq L_S \cap \mathcal{I}$, i.e., the low-level specification together with the interface is controllable with respect to the low-level system and
3. $\{s\sigma | s \in H_S, \sigma \in \Sigma_{uc}, s\sigma \in H \cap \mathcal{I}\} \subseteq H_S$, i.e., the high-level specification is controllable with respect to the high-level system together with the interface.

If a system is serial interface consistent and serial level-wise controllable, then the combined specification $S_{hi} \| S_{lo}$ is controllable with respect to the complete system G.

The above discussion can be extended to the case when there are multiple low-level subsystems working in parallel and interacting with the high level via separate interfaces (see Fig. 11). Let n be the number of low-level subsystems and $i \in \{1, \ldots, n\}$. Then, the alphabet Σ is partitioned into the disjunct alphabets Σ_H, Σ_{L_i}, Σ_{R_i} and Σ_{A_i} and $\Sigma_{I_i} = \Sigma_{R_i} \cup \Sigma_{A_i}$. Similarly, G_{lo_i}, G_{I_i} and S_{lo_i} are the low-level subsystems, the corresponding interfaces and their supervisors, respectively.

Since every low-level subsystem and its interface do not share events with other subsystems and interfaces, they can be considered separately. Thus, *serial system extraction* can be defined. The j^{th} serial system extraction is:

$$\begin{array}{rcl} G_{hi}(j) & = & G_{hi} \| G_{I_1} \| \ldots \| G_{I_{j-1}} \| G_{I_{j+1}} \| \ldots \| G_{I_n}, \\ G_{lo}(j) & = & G_{lo_j},\ G_I(j) \ = \ G_{I_j}, \\ \Sigma(j) & = & \Sigma \setminus \bigcup_{i=\{1,\ldots,n\}\setminus\{j\}} \Sigma_{L_i}, \\ \Sigma_H(j) & = & \Sigma_H \cup \bigcup_{i=\{1,\ldots,n\}\setminus\{j\}} \Sigma_{I_i}, \\ \Sigma_L(j) & = & \Sigma_{L_j},\ \Sigma_R(j) \ = \ \Sigma_{R_j},\ \Sigma_A(j) \ = \ \Sigma_{A_j}, \\ S_{hi}(j) & = & S_{hi} \text{ and } S_{lo}(j) \ = \ S_{lo_j}. \end{array}$$

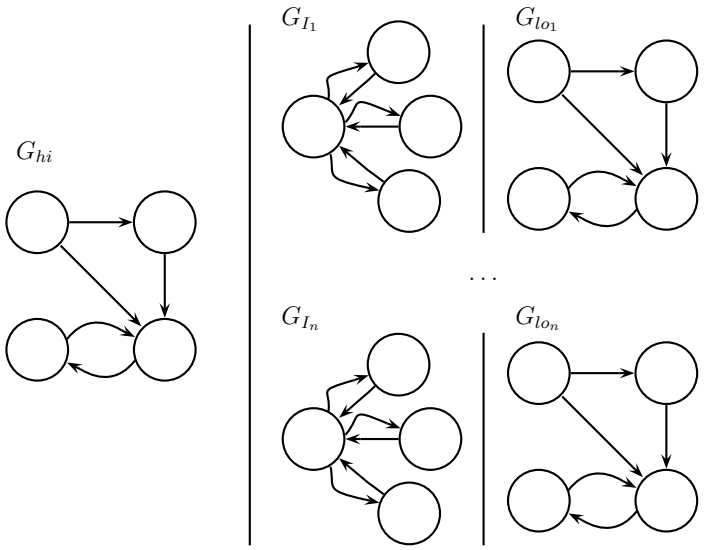


Figure 11: An interface-based hierarchical system with multiple parallel subsystems.

In other words, the j^{th} serial extraction restricts the low level system to the j^{th} low-level subsystem and the interfaces of the other subsystems are considered as a part of the high level.

A system is called *interface consistent* if every serial extraction is serial interface consistent. Similarly, it is *level-wise non-blocking* and the hierarchical specification is *level-wise controllable* if every serial extraction is serial level-wise non-blocking and the corresponding hierarchical specification is serial level-wise controllable. As before, if a system is interface consistent, level-wise non-blocking and the specification is level-wise controllable, the complete system is non-blocking and the specification is controllable.

In [26], algorithms for the verification of serial interface consistency, serial level-wise non-blocking and serial level-wise controllability are presented. Mostly, a combination of standard procedures from the CTCT software [12] are used. The author does not provide analytical complexity results for the algorithms, instead relying on experimental estimates. Let n be the greatest state size of the components of the hierarchical system. Then, the estimated complexity of the combined algorithm (for all verifications) is $O(n^3)$. For a system with m parallel low-level subsystems, the algorithm has to be repeated m times, thus resulting in a linear increase with the addition of parallel subsystems. However, in systems with parallel subsystems, the verification requires the construction of the parallel composition of the high level G_{hi} with all the interfaces G_{I_i}. Thus, comparing this to a monolithic system $G = G_{hi} \| G_{lo_1} \| \ldots \| G_{lo_n}$, one can see that there is a computational advantage only if the state size of each interface is orders of magnitude smaller than the state size of the corresponding low-level subsystem.

2.4.2 Discussion

The interface-based approach to hierarchical control is recent research and so far is limited to work done by Leduc *et al.*. The main novelty is in the introduction of interface-based decoupling of levels in hierarchical DESs. This allows for the independent level-wise verification of the non-blocking and controllability properties. This convenience comes at a cost. The architectural constraints posed on the hierarchical systems, the requirement for interfacing, limits the interaction of levels. Thus, the resulting supervisory control may be sub-optimal compared to the supervisory control for a hierarchical system where levels interact directly. Another drawback in the current research is that algorithms for the verification of the properties of interest are presented only. There are no "constructive" algorithms, i.e., algorithms that can synthesize non-blocking and controllable supervisors given the hierarchical system and specification.

While there are definitely things that need to be improved or advanced in the interface-based approach, I see this work as a big accomplishment with respect to making complex hierarchical DESs more comprehensible and modular. The encapsulation of subsystems guarantees a large degree of maintainability and reusability of the components. This is a very important trait since the real-world complexity of systems renders monolithic approaches inapplicable. Furthermore, the way components are designed fits well with the concepts of object-oriented programming. I believe that this makes the interface-based approach naturally much more intuitive to work with. As with object-oriented programming, modularization and ease are benefits out-weighting the drawback of suboptimal solutions.

3 Other hierarchical approaches

Other research dealing with hierarchical DESs can be found for example in [10, 8].

In [10], Lin extends their previous work on FSMs with parameters [9] to propose a way of obtaining a high-level abstraction from a detailed system (the low-level). Finite-state machines with parameters are standard FSMs enhanced with a discrete parameter space P and the state of the machine is given by a state in Q together with a vector in P. Transitions occur based on events and the fulfillment of "guarding" predicates on P. Furthermore, upon occurrence, the transitions can modify the parameter vectors. A hierarchical FSM with parameters consists of two levels, where the low level is the original (detailed) system, while the high level is an aggregate of the low level: strings are aggregated by the use of a special type of output function (a set of low-level strings are represented by a high-level event) and the parameter vectors are aggregated by the use of a transformation function (for example, the outputs of two machines can be summed-up). This hierarchical approach appears to be interesting since the parametrization of the state space allows for a very compact and convenient representation of complex DESs where the dynamics can be functionally well-defined. However, the work does not go beyond proposing a modeling framework and does not discuss DES control at all. Thus, this approach needs to be extended significantly in order to reach the level of other research currently underway.

In [8], the authors model DESs using the condition/event automata paradigm and then propose a state aggregation method which preserves the input/output functionality of the system. Condition/event automata are FSMs where transitions are enabled by condition signals and are enforced by the event signals. Upon the occurrence of a transition, event

and condition signals may be generated. Condition and event signals may be used within the FSM, however, there may be some which are expected from the environment (input) and some which are emitted to the environment (output). Two systems are input/output equivalent if they generate the same output sequences for equivalent input sequences. The authors propose a state aggregation method which results in input/output-equivalent systems. However, following the example provided in the paper, I discovered a problem with the algorithm. After contacting the authors, they proposed a modification in which transitions with no inputs or outputs are replaced by ϵ-transitions in the beginning of the algorithm.

4 Discussion and conclusion

In this paper I presented an overview of the current research on hierarchical control of discrete-event systems. Four major approaches were identified and described: bottom-up design, top-down design, state aggregation, and interface-based design. Of these, bottom-up design is the most mature approach. Recent work on the top-down approach [32] advances the theory for this paradigm significantly. Interface-based design is the most recent approach and thus is only in the initial stage of research. However, it has very big potential.

All the different approaches to hierarchical control of DESs were grouped into sections. However, the structure of this paper should be taken only as a general guideline. There are very few direct relations between the articles discussed here. Each work differs significantly from the others, either in content or in presentation. I encountered great difficulty in producing a somewhat coherent discussion of the different methodologies and in deciding on a grouping criterion for the articles. Sometimes it was necessary to alter significantly the identifiers and terms used in some articles in order to achieve a more consistent naming convention throughout this paper. At other times, it remains unclear without an in-depth theoretical investigation if two similar terms from two different paper are equivalent (for example, *hierarchical consistency* is a frequently used term but it is not always clear whether it implies the same concept). In my opinion, the fact that almost all research is so disparate in terminology points to an immature research field. This is further confirmed by the abundance of algorithms to achieve similar tasks and the lack of corresponding complexity analysis. The complexity analysis provided usually concerns the proposed algorithm and does not discuss any boundaries on the complexity of the problems. Indeed, as discussed throughout the paper, much more research on the theoretical aspects of hierarchical control of DESs is needed.

The motivation for this paper was to examine the different approaches to hierarchical control of DESs with an eye on their implications for the usability of the models. As mentioned in many publications, e.g., [35], hierarchical models help in understanding complex systems. However, I encountered very little work done outside of the purely theoretical aspect of hierarchical control. Most of the usability discussion consists of short descriptions of how different modules can be composed within the particular frameworks. The lack of good software tools within the DES area is notorious, however, this is true even more so for hierarchical DES models. This is very unfortunate because, without good and efficient tools, the DES control paradigms will remain as a purely academic research topic. Most existing tools, e.g., [12, 45], offer only rudimentary interfaces. In [52], Wood discusses a new highly-visual DES modeling package with emphasis on usability. However, the current version does not support hierarchical modeling.

Unfortunately, what most researchers in the field fail to address is the fact that the formal

methods they propose work flawlessly only when everything is cast in the formal framework. It is relatively safe to assume that a formal model of an existing system can be produced. However, the specifications for the controlled behavior of a system are always given in an informal way: it is people who come up with ideas of what restrictions should be imposed on the system. Sometimes it is very simple to translate the requirements into a formal definition. However, frequently it is the case that the designer "makes a guess" of how a verbal specification can be spelled out formally. It is my belief that no matter how advanced are the theories in the field of discrete-event system control, they will remain unusable unless people can cast their wishes in the proper form and then interpret the results correctly. Hierarchical modeling is a small step forward in usability, however, I do not believe this is nearly sufficient for an easy and intuitive use of DESs for system control purposes. The groundwork for a usable interface between humans and formal DES models is still to be laid.

References

[1] A. A. Al-Falou and J. H. van Schuppen. Aggregation in hierarchical discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 13:321–340, 2003.

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

[3] P. Caines and Y. Wei. The hierarchical lattices of a finite machine. *Systems and Control Letters*, pages 257–263, July 1995.

[4] P. E. Caines, V. Gupta, and G. Shen. The hierarchical control of ST-finite state machines. In *Proceedings of the 36th Conference on Decision and Control*, pages 3584–3589, San Diego, California, USA, December 1997.

[5] P. E. Caines, P. Hubbard, and G. Shen. State aggregation and hierarchical supervisory control. In *Proceedings of the 36th Conference on Decision and Control*, pages 3590–3591, San Diego, California, USA, December 1997.

[6] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1999.

[7] Z. Chao and Y. Xi. Necessary conditions for control consistency in hierarchical control of discrete-event systems. *IEEE Transactions on Automatic Control*, 48(3):465–468, March 2003.

[8] H. Chen and H.-M. Hanisch. Model aggregation for hierarchical control synthesis of discrete event systems. In *Proceedings of the 39th Conference on Decision and Control*, pages 418–423, Sydney, Australia, December 2000.

[9] Y.-L. Chen and F. Lin. Modeling of discrete event systems using finite state machines with parameters. In *Proceedings of the 2000 IEEE International Conference on Control Applications*, pages 941–946, Anchorage, Alaska, USA, September 2000.

[10] Y.-L. Chen and F. Lin. Hierarchical modeling and abstraction of discrete event systems using finite state machines with parameters. In *Proceedings of the 40th Conference on Decision and Control*, pages 4110–4115, Orlando, Florida, USA, December 2001.

[11] S.-L. Chung, S. Lafortune, and F. Lin. Limited lookahead policies in supervisory control of discrete event systems. *IEEE Transactions on Automatic Control*, 37(12):1921–1935, 1992.

[12] CTCT software. Available at `http://www.control.toronto.edu/DES/`.

[13] J. E. R. Cury, C. R. C. Torrico, and A. E. C. da Cunha. A new approach for supervisory control of discrete event systems. In *Proceedings of the European Control Conference*, volume 1, pages 1595–1599, Porto, Portugal, September 2001.

[14] J. E. R. Cury, C. R. C. Torrico, and A. E. C. da Cunha. Supervisory control of discrete event systems with flexible marking. *European Journal of Control*, 10(1):47–60, 2004.

[15] A. E. C. da Cunha and J. E. R. Cury. Hierarchically consistent controlled discrete event systems. In *Proceedings of the 15th IFAC World Congress*, volume 15, Barcelona, Spain, July 2002.

[16] A. E. C. da Cunha, J. E. R. Cury, and B. H. Krogh. An assume-guarantee reasoning for hierarchical coordination of discrete event systems. In *Proceedings of the 6th International Workshop on Discrete Event Systems (WODES '02)*, pages 75–80, Zaragosa, Spain, 2002.

[17] M. H. de Queiroz and J. E. R. Cury. Modular control of composed systems. In *Proceedings of the 2000 American Control Conference*, volume 6, pages 4051–4055, June 2000.

[18] R. F. Drenick. *A Mathematical Organization Theory*. Elsevier Science Publishing Company, Inc., New York, New York, USA, 1986.

[19] B. Gaudin and H. Marchand. Supervisory control of product and hierarchical discrete event systems. *European Journal of Control*, 10(2), 2004.

[20] P. Gohari-M. and W. M. Wonham. A linguistic framework for controlled hierarchical des. In *4th International Workshop on Discrete Event Systems (WODES '98)*, pages 207–212. IEE, August 1998.

[21] L. G. Grigorov. Control of dynamic discrete-event systems. Master's thesis, School of Computing, Queen's University, Kingston, Ontario, Canada, 2004.

[22] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[23] P. Hubbard and P. E. Caines. Dynamical consistency in hierarchical supervisory control. *IEEE Transactions on Automatic Control*, 47(1):37–52, January 2002.

[24] S.-G. Kim, K.-H. Cho, and J.-T. Lim. Hierarchical supervisory control of discrete event systems based on H-observability. *IEE Proceedings—Control Theory and Applications*, 150(2):179–182, March 2003.

[25] R. Kumar and V. K. Garg. *Modeling and Control of Logical Discrete Event Systems*. Kluwer Academic Publishers, Norwell, MA, USA, 1995.

[26] R. J. Leduc. *Hierarchical Interface-based Supervisory Control*. PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, 2002.

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

[28] R. J. Leduc, B. A. Brandin, W. M. Wonham, and M. Lawford. Hierarchical interface-based supervisory control: Serial case. In *Proceedings of the 40th Conference on Decision and Control*, pages 4116–4121, Orlando, Florida, USA, December 2001.

[29] R. J. Leduc, W. M. Wonham, and M. Lawford. Hierarchical interface-based supervisory control: Parallel case. In *Proceedings of the 39th Allerton Conference on Communication, Control, and Computing*, pages 386–395, Monticello, Illinois, USA, October 2001.

[30] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44(2):173–198, 1988.

[31] F. Lin and H. Ying. Modeling and control of fuzzy discrete event systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 32(4):408–415, August 2002.

[32] C. Ma and W. M. Wonham. Control of state tree structures. In *Proceedings of the 11th Mediterranean Conference on Control and Automation*, Rhodes, Greece, June 2003. Paper T4-005.

[33] H. Marchand and B. Gaudin. Supervisory control problems of hierarchical finite state machines. In *41th IEEE Conference on Decision and Control*, pages 1199–1204, Las Vegas, USA, December 2002.

[34] J. O. Moody and P. J. Antsaklis. *Supervisory Control of Discrete Event Systems Using Petri Nets*. Kluwer Academic Publishers, 1998.

[35] K. Q. Pu. Modeling and control of discrete-event systems with hierarchical abstraction. Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, 2000.

[36] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, 1987.

[37] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. In *Proceedings of the IEEE*, volume 77, pages 81–98, January 1989.

[38] S. L. Ricker and K. Rudie. Know means no: Incorporating knowledge into discrete-event control systems. *IEEE Transactions on Automatic Control*, 45(9):1656–1668, September 2000.

[39] K. Rudie and W. M. Wonham. Think globally, act locally: decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, 1992.

[40] R. Sengupta and S. Lafortune. An optimal control theory for discrete event systems. *SIAM Journal of Control and Optimization*, 36(2):488–541, March 1998.

[41] G. Shen and P. E. Caines. Control consistency and hierarchically accelerated dynamic programming. In *Proceedings of the 37th Conference on Decision and Control*, pages 1686–1691, Tampa, Florida, USA, December 1998.

[42] G. Shen and P. E. Caines. Hierarchically accelerated dynamic programming for finite-state machines. *IEEE Transactions on Automatic Control*, 47(2):271–283, February 2002.

[43] B. Tognazzini. *Tog on Interface*. Addison-Wesley Professional, 1992.

[44] C. R. C. Torrico and J. E. R. Cury. Hierarchical supervisory control of discrete event systems based on state aggregation. In *Proceedings of the 15th IFAC World Congress*, volume 15, Barcelona, Spain, July 2002.

[45] UMDES software library. Available at `http://www.eecs.umich.edu/umdes/`.

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

[47] K. C. Wong and W. M. Wonham. Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 6:241–273, 1996.

[48] W. M. Wonham. Supervisory control theory: Models and methods. Informal talk at Queen's University, a verison for the 24th International Conference on Application Theory of Petri Nets is available at `http://www.control.toronto.edu/DES/publish.html`, 2003.

[49] W. M. Wonham. Notes on supervisory control of discrete-event systems. Can be downloaded from `http://www.control.toronto.edu/DES/`, July 2004.

[50] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, 25(3):637–659, 1987.

[51] W. M. Wonham and P. J. Ramadge. Modular supervisory control of discrete-event systems. *Mathematics of Control, Signals, and Systems*, 1:13–30, 1988.

[52] M. M. Wood. Application, implementation and integration of discrete-event systems control theory. Master's thesis, Department of Electrical and Computer Engineering, Queen's University, in preparation.

[53] H. Zhong and W. M. Wonham. On hierarchical control of discrete-event systems. In *Proceedings of the 22nd Annual Conference on Information Sciences and Systems*, pages 64–70, Princeton, New Jersey, USA, March 1988.

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