# Supervisory Control of Discrete-Event Systems

Kai Cai[1] and W.M. Wonham[2]
[1]Department of Electrical and Information Engineering, Osaka City University, Osaka, Japan
[2]Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada

**Abstract**

We introduce background and base model for supervisory control of discrete-event systems, followed by discussion of optimal controller existence, a small example, distributed control through localization, and summary of control under partial observations. Control architecture and symbolic computation are noted as approaches to manage state space explosion.

**Keywords**

Asynchronous · Control architectures · Controllability · Discrete · Dynamics · Finite automata · Observability · Optimality · Regular languages · Symbolic computation

## Introduction

Discrete-event (dynamic) systems (DES or DEDS) constitute a relatively new area of control science and engineering, which has taken its place in the mainstream of control research. Recently, DES have been combined with continuous systems in areas called hybrid or cyber-physical systems.

Problems and methods for DES have been investigated for some time, although not necessarily with a "control" flavor. The parent domains can be identified as operations research and software engineering.

Operations research deals with systems of interconnected stores and servers which operate on processed items. For instance, manufacturing systems employ queues, buffers, and bins (which store workpieces). These are served by machines, robots, and automatic guided vehicles (AGVs), which process workpieces. The main problems are to measure quantitative performance and establish trade-offs, for instance, workflow vs. cost, and to optimize design parameters such as buffer size and maintenance frequency.

The relevant areas of software engineering include operating systems control, concurrent computing, and real-time (embedded or reactive) systems, with focus on synchronization algorithms that enforce mutual exclusion and resource sharing in the presence of concurrency, as in the classical problems of "readers and writers" and "dining philosophers." The main objectives are

J. Baillieul, T. Samad (eds.), *Encyclopedia of Systems and Control*,
https://doi.org/10.1007/978-1-4471-5102-9_54-2

(i) to guarantee safety (“nothing bad will ever happen”), as in mutual exclusion and deadlock prevention, and (ii) to guarantee liveness (“something good will happen eventually”), for instance, successful computational termination and eventual access to a desired resource.

## DES from a Control Viewpoint

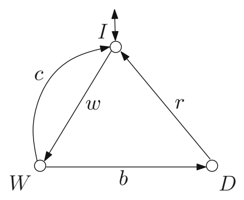
With these domains in mind, we consider DES from a control viewpoint. In general, control deals with dynamical systems, defined as entities consisting of an internal state space, together with a state evolution or transition structure, and equipped (for control purposes) with both an input mechanism for actuation and an output channel for observation and feedback. The objective of control is to bring together information and dynamics in some purposeful combination: the interplay between observation and control or decision-making is fundamental.

In this framework, a DES is a dynamical system that is discrete, in time and usually in state space; is asynchronous or event driven, that is, driven by events or instantaneous happenings in time (which may or may not include the tick of a clock); and is nondeterministic, namely, embodies internal chance or other unmodeled mechanisms of choice which govern its state transitions. With a manufacturing system, for example, the dynamic state might include the status of machines (idle, working, down, under maintenance or repair), the contents of queues and buffers, and the locations and loads of robots and AGVs, while transitions (discrete events) occur when queues and buffers are incremented or decremented, robots load or unload, and machines start work, finish work, or break down (the “choice” between finishing work successfully and breaking down being thus nondeterministic). In this example and many others, the objectives of design and analysis include logical correctness in the presence of concurrency and timing constraints and quantitative performance such as rates of production, all of which depend crucially on feedback control synthesis and optimization. To this end the models will tend to be DES or hybrid systems. Nevertheless one finds the continuing relevance of standard control-theoretic concepts like feedback, stability, controllability, and observability, along with their roles in large-system architectures embodying hierarchical, decentralized, and distributed functional organization.

Here we focus on models and problems from which explicit constraints of timing are absent and which can be considered in a framework of finite-state machines and the corresponding regular languages. While the theory has been generalized to more flexible and technically advanced settings, our restricted framework is already rich enough to support numerous applications and remains challenging for large systems of industrial size.

## Base Model for Control of DES

The formal structure of a DES to be controlled will resemble the simple “machine” called **MACH** shown in Fig. 1. The state set of **MACH** is $Q = \{I, W, D\}$, interpreted as Idle, Working, or Broken Down. **MACH** is initialized at state $q_o = I$, denoted by an entering arrow without source. The transition structure is displayed in Fig. 1 as a transition diagram, whose nodes are the states $q \in Q$ and edges are the transitions, each labeled with a symbol $\sigma$ in the alphabet $\Sigma$, here $\{w, c, b, r\}$. If a transition (labeled) $\sigma$ is an edge from $q$ to $q'$, then “the event $\sigma$ can occur at state $q$, and when $\sigma$ occurs, $q$ transits to state $q'$.” Transitions (or events) are interpreted as instantaneous in time, while states are thought of as locations where **MACH** is able to reside for some indeterminate time interval. The occurrence of $w$ means “**MACH** enters the



**Supervisory Control of Discrete-Event Systems, Fig. 1** MACH

Working state from Idle" and similarly for $c, b, r$. These transitions determine the state-transition function of **MACH**, denoted by $\delta : Q \times \Sigma \to Q$. Thus $\delta(I, w) = W$, $\delta(W, b) = D$, and so on. Notice that $\delta$ is a partial function, defined at each state $q \in Q$ for only a subset of event (labels) in $\Sigma$. To denote that $\delta(q, \sigma)$ is defined at state $q \in Q$ for the event $\sigma \in \Sigma$, we write $\delta(q, \sigma)!$. The function $\delta$ can be extended by iteration to $\delta : Q \times \Sigma^* \to Q$, where $\Sigma^*$ is the set of all finite strings of elements of $\Sigma$, including the empty string $\epsilon$. Thus $\delta(q, \epsilon) := q$ and inductively if $q' := \delta(q, s)!$, then

$$\delta(q, s.\sigma) := \delta(\delta(q, s), \sigma) := \delta(q', \sigma)$$

whenever $\delta(q', \sigma)!$. Graphically the strings $s = \sigma_1 \ldots \sigma_k \in \Sigma^*$ for which $\delta(q, s)!$ are precisely those for which there exists a path in the transition diagram starting from $q$ and having successive edges labeled $\sigma_1, \ldots, \sigma_k$.

We call any subset of $\Sigma^*$ (i.e., any set of strings of elements from $\Sigma$) a language over $\Sigma$ and accordingly speak of sublanguages of a language over $\Sigma$.

For **MACH**, the execution of a production cycle, namely, the event sequence (or string) $w.c$, or a work-breakdown-repair cycle, the string $w.b.r$, can be considered successful, and the corresponding string is said to be *marked*. States which are entered by marked strings are marked states and identified in a transition diagram by an outgoing arrow with no target. In Fig. 1, the only marked state happens to be the initial state, which is thus shown with a double arrow; in general there could be several marked states, which may or may not include the initial state. The marked states comprise a subset $Q_m \subseteq Q$, which may be empty (at one extreme) or equal to $Q$ (at the other). The case $Q_m = Q$ (all states marked) would imply that every string of events is considered as significant or successful as any other, while the case $Q_m = \emptyset$ (no state marked, so there are no successful strings) plays a technical role in computation.

In general a DES is a tuple $\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m)$ usually interpreted physically as for **MACH** above, but mathematically consisting merely of the finite state set $Q$; finite alphabet $\Sigma$; marked subset $Q_m \subseteq Q$, with initial state $q_o \in Q$; and (partial) transition function $\delta : Q \times \Sigma \to Q$. Additionally we bring in the closed behavior $L(\mathbf{G})$ of $\mathbf{G}$, defined as all the strings of $\Sigma^*$ which $\mathbf{G}$ can generate starting from the initial state, in the sense:

$$L(\mathbf{G}) := \{s \in \Sigma^* \mid \delta(q_o, s)!\}.$$

Of central importance also is the marked behavior of $\mathbf{G}$, namely, the sublanguage of $L(\mathbf{G})$ given by

$$L_m(\mathbf{G}) := \{s \in L(\mathbf{G}) \mid \delta(q_o, s) \in Q_m\}.$$

We need several definitions. A string $s'$ is a *prefix* of a string $s \in \Sigma^*$, written $s' \leq s$, if $s'$ can be extended to $s$, namely, there exists a string $w$ in $\Sigma^*$ such that $s'.w = s$. The *closure* of a language $M \subseteq \Sigma^*$ is the language $\overline{M}$ consisting of all prefixes of strings in $M$

$$\overline{M} := \{s' \in \Sigma^* \mid s' \leq s \text{ for some } s \text{ in } M\}$$

A language $N$ over $\Sigma$ is (prefix-)*closed* if it contains all its prefixes, namely, $N = \overline{N}$. In this notation $\mathbf{G}$ is said to be *nonblocking* if $L(\mathbf{G}) = \overline{L_m(\mathbf{G})}$, namely, any (generated) string in $L(\mathbf{G})$ is a prefix of, and so can be extended to, a marked string of $\mathbf{G}$.

The semantics of $\mathbf{G}$ (its mathematical meaning) is simply the pair of languages $L_m(\mathbf{G})$, $L(\mathbf{G})$. In general the latter may be infinite subsets of $\Sigma^*$, while $\mathbf{G}$ itself is a finite object, considered to represent an algorithm for the generation of its behaviors. Unless $\mathbf{G}$ is trivial (has empty state set), it is always true that (the empty string) $\epsilon \in L(\mathbf{G})$.

Transition labeling of $\mathbf{G}$ is deterministic: at every $q$, at most one transition is defined for each given event $\sigma$, namely,

$$\delta(q, \sigma) = q' \ \& \ \delta(q, \sigma) = q'' \text{ implies } q' = q''.$$

It is quite acceptable, however, that at distinct states $q$ and $r$, both $\delta(q, \sigma)!$ and $\delta(r, \sigma)!$ (where these evaluations may or may not be equal).

To formulate a control problem for $\mathbf{G}$, we first adjoin a control technology or mechanism by which $\mathbf{G}$ may be actuated to affect its temporal behavior, namely, determine the strings it is permitted to generate. To this end we assume that a subset of events $\Sigma_c \subseteq \Sigma$, called the *controllable* events, are capable of being enabled or disabled by an external controller. Think of a traffic light being turned green or red to allow or prohibit passage (vehicle transition) through an intersection. The complementary event subset $\Sigma_u := \Sigma - \Sigma_c$ is *uncontrollable*; events $\sigma \in \Sigma_u$ cannot be externally disabled but may be considered permanently enabled. For $\mathbf{G} = \mathbf{MACH}$ one might reasonably assume $\Sigma_c = \{w, r\}$, $\Sigma_u = \{c, b\}$. At a given state $q$ of $\mathbf{G}$, it will be true in general that $\delta(q,\sigma)!$ both for some (controllable) events $\sigma \in \Sigma_c$ and for some (uncontrollable) events $\sigma \in \Sigma_u$. Among the $\sigma \in \Sigma_c$, at a given time, some may be externally enabled and others disabled. So, $\mathbf{G}$ will nondeterministically choose its next generated event from the subset

$$\{\sigma \in \Sigma_u \mid \delta(q,\sigma)!\} \cup \{\sigma \in \Sigma_c \mid \delta(q,\sigma)! \;\&\; \sigma \text{ is externally enabled}\} \tag{1}$$

We formalize external enablement by a *supervisory control function* $V : L(\mathbf{G}) \to Pwr(\Sigma)$, where $Pwr(\cdot)$ stands for power set. For $s \in L(\mathbf{G})$, the evaluation $V(s)$ is defined to be the event subset

$$V(s) := \Sigma_u \cup \{\sigma \in \Sigma_c \mid \sigma \text{ is externally enabled following } s\} \tag{2}$$

In other words, the set (1) is expressible as

$$V(s) \cap \{\sigma \in \Sigma \mid s.\sigma \in L(\mathbf{G})\} \tag{3}$$

namely, the subset of events that, immediately following the generation of $s$ by $\mathbf{G}$, are either enabled by default (executable events in $\Sigma_u$) or else by the external controller's decision (a subset of executable events in $\Sigma_c$).

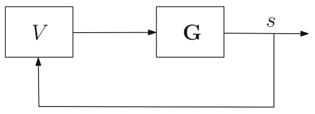
It is now easy to visualize how the generating action of $\mathbf{G}$ is restricted by the action of $V(\cdot)$. Initially (having generated the empty string) $\mathbf{G}$ chooses $\sigma_1 \in V(\epsilon) \cap L(\mathbf{G})$. Proceeding inductively, after $\mathbf{G}$ has generated $s = \sigma_1.\sigma_2 \ldots \sigma_k \in L(\mathbf{G})$, $s$ is fed back to the controller, which evaluates $V(s)$ according to (2), announcing the result to $\mathbf{G}$, which then chooses $\sigma_{k+1}$ in (3), and the process repeats. Of course the process would terminate any time the set (3) happened to become empty (although it need not). In any case, we denote the subset of $L(\mathbf{G})$ so determined as $L(V/\mathbf{G})$, called the *closed behavior* of $V/\mathbf{G}$, where the latter symbol (formally undefined) stands for $\mathbf{G}$ under the supervision of $V$. It is clear that supervision is a feedback process (Fig. 2), inasmuch as the choice of $\sigma_{k+1}$ in (3) is not, in general, known in advance, hence must be executed before the succeeding evaluation $V(s.\sigma_{k+1})$ can allow the generating process to continue. With the closed behavior of $V/\mathbf{G}$ now determined, we define the *marked behavior*

$$L_m(V/\mathbf{G}) := L(V/\mathbf{G}) \cap L_m(\mathbf{G}) \tag{4}$$

namely, those marked strings of $\mathbf{G}$ that survive under supervision by $V$. Thus supervisory control is nonblocking if $L(V/\mathbf{G}) = \overline{L_m(V/\mathbf{G})}$.

## Existence of Controls for DES: Controllability

Of fundamental interest is the question: what sublanguages of $L(\mathbf{G})$ qualify as a language $L(V/\mathbf{G})$ for some choice of supervisory control function $V$? In other words, what is the scope of controlled behavior(s) for a given $\mathbf{G}$? So far we know that $L(V/\mathbf{G})$ is a sublanguage of $L(\mathbf{G})$, but it is not usually the case that an arbitrary sublanguage would qualify. For instance, the empty string



**Supervisory Control of Discrete-Event Systems, Fig. 2** Feedback loop $V/\mathbf{G}$

language $\{\epsilon\} \neq L(V/\mathbf{G})$ for any $V$ as in (2) above, in case $\delta(q_o, \sigma)!$ for some $\sigma$ in $\Sigma_u$, for such $\sigma$ cannot be disabled.

Assume $\mathbf{G}$ is equipped with the technology of controllable events, hence uncontrollable events $\Sigma_u \subseteq \Sigma$. We make the basic definition: the language $K \subseteq \Sigma^*$ is *controllable* (with respect to $\mathbf{G}$) provided

$$\text{For all } s \in \overline{K} \text{ and for all } \sigma \in \Sigma_u,$$
$$\text{whenever } s.\sigma \in L(\mathbf{G}) \text{ then } s.\sigma \in \overline{K}. \qquad (5)$$

Informally, a string $s$ can never exit from $\overline{K}$ as the result of the execution by $\mathbf{G}$ of an uncontrollable event: $\overline{K}$ is invariant under the uncontrollable flow. In terms of $\mathbf{G} = \mathbf{MACH}$, above, the languages $\{\epsilon\}$, $\{w.b, w.c\}$ are controllable, but $\{w\}$, $\{w, w.c.w\}$ are not. For instance, $H := \{w, w.c.w\}$ has closure $\overline{H} = \{\epsilon, w, w.c, w.c.w\}$, which contains the string $s := w$, but $s.b = w.b$ can be executed in **MACH**, $b$ is uncontrollable, and $s.b$ has exited from $\overline{H}$. It is logically trivial from (5) that the empty language $\emptyset$ (with no strings whatever) is controllable.

We can now answer the fundamental question posed above.

$$\text{Given a nonempty sublanguage } K \subseteq L(\mathbf{G}),$$
$$\text{there exists a supervisory control function } V$$
$$\text{such that } \overline{K} = L(V/\mathbf{G}), \text{ if and only if } K$$
$$\text{is controllable.} \qquad (6)$$

This result exhibits the $L(V/\mathbf{G})$ property in a structured way; furthermore, both the containment $K \subseteq L(\mathbf{G})$ and the controllability property (5) (or its absence) can be effectively (algorithmically) decided in case $K$ itself is the closed or marked behavior of some given DES over $\Sigma$.

A key fact easily provable from (5) is that the family of all controllable languages (with respect to a fixed $\mathbf{G}$) is algebraically closed under union, namely,

$$\text{If } K_1 \text{ and } K_2 \text{ are controllable languages,}$$
$$\text{then so is } K_1 \cup K_2. \qquad (7)$$

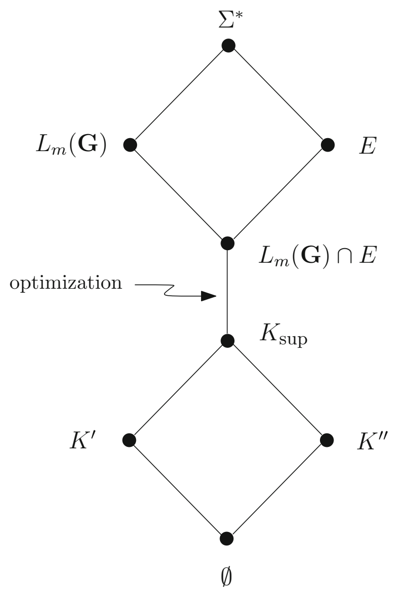
In fact (7) can be extended to an arbitrary finite or infinite union of controllable languages.

Given $\mathbf{G}$ as above, considered as the plant to be controlled, suppose a new (regular) language $E$ is specified, as the maximal set of strings that we are prepared to tolerate for generation by $\mathbf{G}$; for instance, $E$ could be considered the legal language for $\mathbf{G}$ (irrespective of what $\mathbf{G}$ is potentially capable of generating, namely, $L(\mathbf{G})$). Let us confine attention to the sublanguage of $E$ that contains only marked strings of $\mathbf{G}$, namely, $E \cap L_m(\mathbf{G})$. We now bring in the family $C(E \cap L_m(\mathbf{G}))$ of all controllable sublanguages of $E \cap L_m(\mathbf{G})$ (including the empty language). From (7) and its infinite extension, there follows the existence of the controllable language

$$K_{\text{sup}} := \cup\{K \mid K \in C(E \cap L_m(\mathbf{G}))\} \qquad (8)$$

We have $K_{\text{sup}} \subseteq E \cap L_m(\mathbf{G})$, and clearly if $K'$ is controllable and $K' \subseteq E \cap L_m(\mathbf{G})$, then $K' \subseteq K_{\text{sup}}$. $K_{\text{sup}}$ is therefore the supremal (largest) controllable sublanguage of $E \cap L_m(\mathbf{G})$. Furthermore, if $K_{\text{sup}}$ is nonempty, then by (6) there exists a supervisory control $V$ such that $K_{\text{sup}} = L(V/\mathbf{G})$; in this sense $V$ is optimal (maximally permissive), allowing the generation by $\mathbf{G}$ of the largest possible set of marked strings that the designer considers legal. We have thus established abstractly the existence and uniqueness of an optimal control for given $\mathbf{G}$ and $E$. This simple conceptual picture is displayed (Fig. 3) as a Hasse diagram, in which nodes represent sublanguages of $\Sigma^*$ and rising lines (edges) the relation of sublanguage containment.

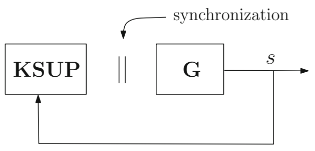
In a Hasse diagram it could be that $K_{\text{sup}}$ collapses to the empty language $\emptyset$. This means that



**Supervisory Control of Discrete-Event Systems, Fig. 3**
Hasse diagram

there is no supervisory control for the problem considered, either because the specifications are too severe and the problem is over-constrained or because the control technology is inadequate (more events need to be controllable).

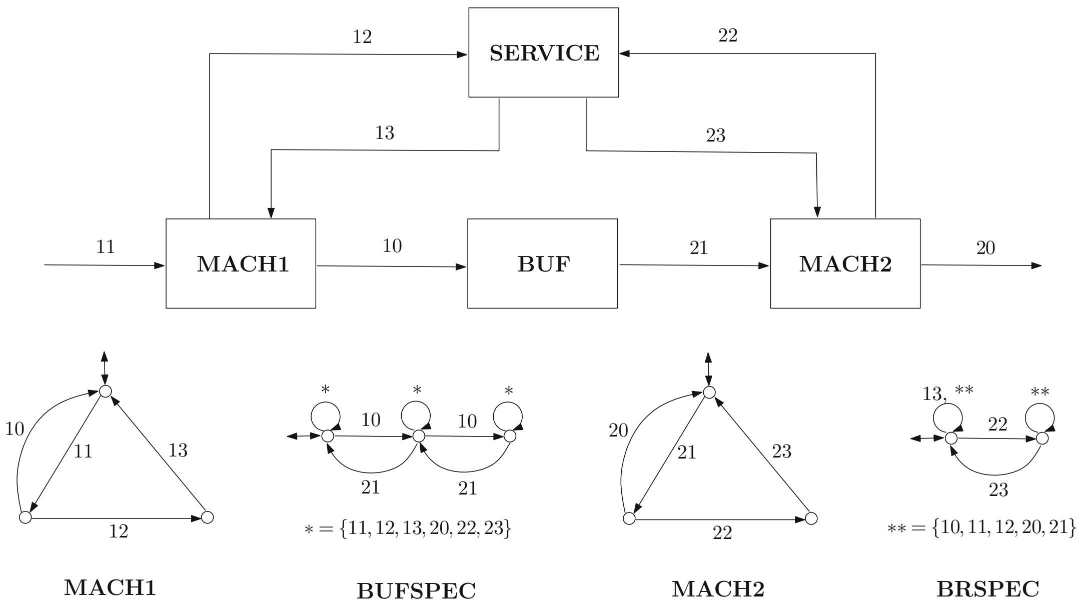
Under the finite-state assumption, $K_{\text{sup}}$ is effectively representable by a DES **KSUP**, which may serve as the optimal feedback controller, as displayed in Fig. 4. Here a string $s$ generated by **G** drives **KSUP**; at each state of **KSUP**, the events defined in its transition structure are exactly those available to **G** for nondeterministic execution (in its corresponding state) at the next step of the process. In this way the feedback control process is inductively well defined. The computational complexity of this design (cf. (8)) is $O(|\mathbf{E}|^2 \cdot |\mathbf{G}|^2)$ where **E** is a DES with $L_m(\mathbf{E}) = E$ and $|\cdot|$ denotes state size. The controller state size is $|\mathbf{KSUP}| \leq |\mathbf{E}| \cdot |\mathbf{G}|$, the product bound being of typical order.



**Supervisory Control of Discrete-Event Systems, Fig. 4**
Implementation of $V/\mathbf{G}$

## Supervisory Control Design: Small Factory

The following example, Small Factory (SF), is an illustration of supervisor design. As in Fig. 5, SF consists of two machines **MACH1** and **MACH2** each similar to **MACH** above, connected by a buffer **BUF** of capacity 2. In case of breakdown the machines can be repaired by a **SERVICE** facility as shown. Transition structures of the machines and design specifications are also displayed in Fig. 5. $\Sigma_c$ ($\Sigma_u$) are odd (even)-numbered events. When self-looped with all irrelevant events to form specification **BUFSPEC**, the latter declares that the machines must be controlled in such a way that **BUF** is not overflowed (an attempt by **MACH1** to deposit a workpiece in **BUF** when it is full) or subject to underflow (an attempt by **MACH2** to take a workpiece from **BUF** when it is empty). In addition, **SERVICE** must enforce priority of repair for **MACH2**: when the latter is down, repair of **MACH1** (if in progress) must be interrupted and only resumed after **MACH2** has been repaired; this logic is expressed by **BRSPEC**. To form the plant model **G** for the DES to be controlled, we compute the synchronous product of **MACH1** and **MACH2**. The result, say **G** = **FACT**, is a DES of which the components **MACHi** are free to execute their events independently except for synchronization on events that are shared (here, none). Similarly we form the synchronous product of **BUFSPEC** and **BRSPEC** to obtain the full specification DES **SPEC**. We now execute the optimization step in the Hasse diagram (Fig. 3); this yields the SF controller **KSUP**(21,47) with 21 states and 47 transitions. Online synchroniza-



**Supervisory Control of Discrete-Event Systems, Fig. 5** Small Factory

tion of **KSUP** with **FACT** will result in generation of the optimal controlled behavior $K_{\text{sup}}$ by the feedback loop. Since $K_{\text{sup}} \subseteq L_m(\mathbf{G})$ by (8), our marking conventions ensure that **KSUP** is nonblocking.

In general the language $K_{\text{sup}}$ will include in its structure not only the constraints required by control but also the physical constraints enforced by the plant structure itself (here, **FACT**). The latter are thus redundant in the online synchronization of the plant with the controller **KSUP**. A more economical controller is obtained if the plant constraints are projected out of **KSUP** to obtain a reduced controller, say **KSIM**. Mathematically, projection amounts to constructing a control congruence or dynamically (and control) consistent partition on the state set of **KSUP** and taking the cells of this partition, abstractly, as the new states for **KSIM**. In SF **KSUP**(21,47) is reduced to **KSIM**(5,18), which when synchronized with **FACT** yields exactly **KSUP** but is less than one-quarter the state size. In practice a state size reduction factor of ten or more is not uncommon.

## Distributed Control by Supervisor Localization

The projection technique above that yields a reduced controller can be extended further to create *local controllers* for individual component DES. In Small Factory, to create a local controller for machine **MACH1**, not only the constraints imposed by the plant structure but also those by control of machine **MACH2** are redundant or irrelevant. Projecting these constraints out of the monolithic controller **KSUP** generates a controller dedicated solely to control of machine **MACH1** and in this sense is "local." A symmetric projection generates another local controller for **MACH2**. This procedure of creating local controllers by decomposing the monolithic controller is called *supervisor localization*. It can be generalized as well to the case of multicomponent DES and guarantees that the local controllers when synchronized with the plant yield exactly the monolithic controller. Hence, equipped with its own private controller, each component DES may act autonomously without centralized supervision; meanwhile, the collective local controlled behavior is ensured to be the same as that achieved by the monolithic controller. This arrangement with only local controllers is called (purely) *distributed* control architecture; it is useful for systems comprised of multiple autonomous components, such as networked robots, vehicles, or sensors.

Typically, local controllers need to communicate certain events to one another to achieve critical synchronization. The identity between local controlled behavior and global controlled behavior is based on the assumption that event communication is instantaneous. In practice, however, communication is through physical channels, which are subject to (greater or less) time delay. Consequently, when a local controller sends an event $\sigma$ to another local controller, the latter can receive $\sigma$ only after some delay. This communication delay may be reflected by a channel model that treats $\sigma$ as sending the event, but a distinct $\sigma'$ as receiving the event, such that $\sigma'$ occurs after $\sigma$ with (variable but unbounded) delay. With this channel model, one may test if the local controllers are *robust* to communication delay, in the sense that the channeled behavior (including $\sigma'$) is *complete* and *correct* with respect to the original, idealized, zero-delay controlled behavior. Completeness means that every zero-delay behavior is some projected channeled behavior (where the delayed event $\sigma'$ is projected to $\epsilon$), while correctness means that every projected channeled behavior is some zero-delay behavior.

## Supervisor Architecture and Computation

As noted earlier, the state size |**KSUP**| of controller **KSUP** is on the order of the product of state sizes of the plant, |**G**|, and specification, |**E**|. As these in turn are the synchronous products of individual plant components or partial specifications, |**KSUP**| tends to increase exponentially with the numbers of plant components and specifications, the phenomenon of exponential state space explosion. The result is that centralized or monolithic controllers such as **KSUP** can easily reach astronomical state sizes in realistic industrial models, thereby becoming infeasible in terms of computer storage for practical design. This issue can be addressed in two basic ways: by decentralized and hierarchical architectures, possibly in heterarchical combination, and by symbolic DES representation and computation, where what is stored are not DES and their controller transition structures in extensional (explicit) form, but instead intensional or algorithmic recipes from which the required state and control variable evaluations are computed online only when actually needed.

## Supervisory Control Under Partial Observations

For a DES **G** over alphabet $\Sigma$, let $\Sigma_o \subseteq \Sigma$ be a subalphabet interpreted as the events that can be viewed by some external observer. A mapping $P : \Sigma^* \to \Sigma_o^*$ is called a *natural projection* if its action is simply to erase from a string $s$ in $\Sigma^*$ all the events in $s$ (if any) that do not belong to $\Sigma_o$ while preserving the order of events in $\Sigma_o$. By use of $P$ it is possible to carry over to DES the control-theoretic concept of observability. Two strings $s, s' \in \Sigma^*$ are *look-alikes* with respect to $P$ if $Ps = Ps'$, namely, are indistinguishable to an observer (or channel) modeled by $P$. Thus, given **G** and $P$ as above, a sublanguage $K \subseteq L(\mathbf{G})$ is *observable* if, roughly, look-alike strings in $\overline{K}$ have the same one-step extensions in $\overline{K}$ that are compatible with membership in $L(\mathbf{G})$ and also satisfy a consistency condition with respect to membership in $L_m(\mathbf{G})$. For control under observations through $P$, one defines a supervisory control function $V : L(\mathbf{G}) \to Pwr(\Sigma)$ to be *feasible* if it assumes the same value on look-alike strings, in other words respects the observation constraint enforced by $P$. It then turns out that a language $K \subseteq L_m(\mathbf{G})$ can be feasibly synthesized in a feedback loop including **G** and the feedback channel $P$ if and only if $K$ is both controllable and observable.

Although this result is conceptually satisfying, it is computationally inconvenient because, by contrast with controllability, the property of sublanguage observability is not in general closed under union. A substitute for observability is sublanguage *normality*, a property stronger than observability but one that is indeed closed under union. Since the family of controllable and normal sublanguages of a given specification language is nonempty (the empty language belongs) and is closed under union, a (unique) supremal (or optimal) element exists and can be computed; it therefore solves the problem of supervisory control under partial observations, albeit under the normality restriction. The latter has the feature that the resulting supervisor can disable a controllable event only if the latter is observable, i.e., belongs to $\Sigma_o$. In some applications this restriction might preclude the existence of a solution altogether; in others it could be harmless, or even desirable as a safety property, in that if the intended disablement of a controllable event happened to fail, and the event occurred after all, the fault would necessarily be observable and thus optimistically remediable in good time.

An intermediate property is known that is weaker than normality but stronger than observability, called *relative observability*. The family of relatively observable sublanguages of a given specification language is closed under union and thus does possess a supremal element, which in the regular case can be effectively computed. When combined with controllability, relative observability yields a solution to the problem of supervisory control under partial observations which places no limitation on the disablement of unobservable controllable events. Examples show that a nontrivial solution of this type may exist in cases where the normality solution is empty.

## Summary and Future Directions

Supervisory control of discrete-event systems, while relatively new, has reached a first level of maturity in that it is soundly based in a standard framework of (especially) finite-state machines and regular languages. It has effectively incorporated its own versions of control-theoretic concepts like stability (in the sense of nonblocking), controllability, observability, and optimality (in the sense of maximal permissiveness). Modular architectures and, on the computational side, symbolic approaches enable design of both monolithic and heterarchical/distributed controllers for DES models of industrial size. Major challenges remain, especially to develop criteria by which competing architectures can be meaningfully compared and to organize control functionality in ways that are not only tractable but also transparent to the human user and designer.

## Cross-References

- ▶ Applications of Discrete-Event Systems
- ▶ Models for Discrete Event Systems: An Overview

## Bibliography

Cai K, Wonham WM (2016) Supervisor localization: a top-down approach to distributed control of discrete-event systems. Lecture notes in control and information sciences, vol 459. Springer International Publishing

Cassandras CG, Lafortune S (2008) Introduction to discrete event systems. Springer, New York

Cieslak R, Desclaux C, Fawaz A, Varaiya P (1988) Supervisory control of discrete-event processes with partial observations. IEEE Trans Autom Control 33:249–260

Lin F, Wonham WM (1988) On observability of discrete-event systems. Inf Sci 44:173–198

Ma C, Wonham WM (2005) Nonblocking supervisory control of state tree structures. Lecture notes in control and information sciences, vol 317. Springer, Berlin

Ramadge PJ, Wonham WM (1987) Supervisory control of a class of discrete event processes. SIAM J Control Optim 25:206–230

Seatzu C et al (ed) (2013) Control of discrete-event systems. Springer, London/New York

Wonham WM, Cai K, Rudie K (2018) Supervisory control of discrete-event systems: a brief history. Ann Rev Control 45:250–256

Wonham WM, Cai K (2019) Supervisory control of discrete-event systems. Communications and Control Engineering, Springer International Publishing