Annual Reviews in Control 45 (2018) 250–256

Contents lists available at ScienceDirect

Annual Reviews in Control

journal homepage: www.elsevier.com/locate/arcontrol

Review article

Supervisory control of discrete-event systems: A brief history

W.M. Wonham [a], Kai Cai [b,*], Karen Rudie [c]

[a] Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada
[b] Department of Electrical and Information Engineering, Osaka City University, Osaka 558-8585, Japan
[c] Department of Electrical and Computer Engineering, Queen's University, Kingston, ON K7L 3N6, Canada

ARTICLE INFO

Article history:
Received 23 December 2017
Revised 11 March 2018
Accepted 21 March 2018
Available online 9 April 2018

Keywords:
Supervisory control theory
Discrete-event systems
History

ABSTRACT

This brief history summarizes the 'supervisory control of discrete-event systems' as it has evolved in the period 1980–2017. Overall, the trend has been from centralized or 'monolithic' control to more structured architectures, and from 'naive' to symbolic computation. Like any 'history' this one represents the perspective of the authors; in consequence some important contributions may have been overlooked or short-changed.

Contents

1. Background and early motivation c.1980

The year 1980 is a convenient point of departure for supervisory control of discrete-event systems (SCDES) in the current sense. By that time the broad field known as 'modern' systems control that had evolved over the previous 20 years was well-established, on the basis of the five fundamental concepts of feedback, stability, controllability, observability, and quantitative optimality (Athans & Falb, 1966).[1] Of particular relevance later to SCDES were geometric concepts for regulator synthesis by feedback in linear multivariable systems (Basile & Marro, 1991; Wonham, 1985), namely the lattice of subspaces of a linear vector space, including controlled invariant subspaces, controllability subspaces, supremal (unique maximal) elements in the sense of partial order by subspace inclusion, and the concomitant notion of qualitative optimality. The dynamical systems to which systems control applied were generally those described by ordinary or partial differential equations and their discrete-time sampled-data counterparts, the main application drivers being the industrial process industries and various national space programs.

By contrast, discrete-event systems (DES) was an area apart, concerned with systems usually discrete in time and state space, driven by instantaneous events other than (or in addition to) the tick of a clock, and 'nondeterministic' in the sense of making state-transitional choices by internal chance or other mechanisms not necessarily modeled by the system analyst. Such systems were generally not amenable to the differential techniques of systems control. The application drivers included manufacturing, traffic, database management, and logistic systems. Owing to model complexity and analytical intractability, sys-

* Corresponding author.
E-mail addresses: wonham@ece.utoronto.ca (W.M. Wonham), kai.cai@eng.osaka-cu.ac.jp (K. Cai), karen.rudie@queensu.ca (K. Rudie).

[1] For the reader's convenience, in place of original sources textbooks or monographs may be cited where references to the former can be found.

https://doi.org/10.1016/j.arcontrol.2018.03.002
1367-5788/

tem simulation played a major role in analysis and optimization; indeed the term 'DES' seems to have originated with the simulation community and computer languages like SIMULA and SIMSCRIPT (Fishman, 1978). Theoretical analysis rested on queues (Ephremides, Varaiya, & Walrand, 1980), Markov chains (Howard, 1960), Petri nets (Peterson, 1981), and boolean transition structures (Aveyard, 1974); while design approaches exploited semaphores (Dijkstra, 1965), path expressions (Shields, 1979), and computer program representation by pseudo-code along with 'by-hand' cut-and-try alternating with informal verification (Ben-Ari, 1982). Formal attacks evolved in response, especially in software science, either expressed in process algebras such as *communicating sequential processes* (CSP) (Hoare, 1985), the *calculus of communicating systems* (CCS) (Milner, 1989), and work of the emerging Dutch school (Baeten, 2004); or proposed from a language perspective in terms of process behaviors (Arnold & Nivat, 1980; Beauquier & Nivat, 1980).[2]

In this literature control problems were certainly implicit, but formal synthesis (in the style of systems control) was broadly lacking. No standard paradigm existed analogous to optimal control, and there was often no clear separation of controller and uncontrolled 'plant'. The need (or at least the interest) was therefore apparent of a DES control theory which would (1) be discrete in time and space, asynchronous, event-driven as well as (possibly) clock-driven, and nondeterministic (supporting autonomous transitional choices); (2) rest on a simple control 'technology' and exploit standard control concepts; (3) be amenable to computation and applicable to the DES drivers (such as manufacturing); and (4) be accessible to practitioners and students of control engineering.

2. Language controllability and monolithic supervisory control – 1981–1987

The DES control theory which appeared in response (Ramadge & Wonham, 1982) was not startling. The plant was modeled internally as a finite state machine (FSM) for ease of physical interpretation and explicit computation;[3] plant external behavior was thus a regular language, which could optionally be considered the conceptual starting point independently of representation. Control specification was also modeled as an FSM, the corresponding regular language providing an upper bound on acceptable controlled behavior. The key advantages of this setting were its flexibility, broad expressiveness, and the technical feature of regular language closure under the boolean operations. The proposed control technology was simply a partition of the language alphabet (of *events*) into *controllable* and *uncontrollable*, the former amenable to disablement (prevention from occurrence) by a hypothetical external agent dubbed *supervisor*, the latter events not capable of direct disablement but presumed liable to occur (by chance, or internal system 'volition') whenever defined at the system's current state.[4]

The *supervisory control synthesis problem* was then formalized as that of designing a finite-state supervisor which, on observing the string of events generated by the plant, would at each state disable a suitable subset of controllable events to ensure that the generated controlled behavior (regular language) continued to satisfy the control specification, namely remained within the specification or 'legal' language. As in standard systems control, the approach was thus to separate the issues of problem definition and explicit computation.

Inasmuch as disabling all controllable events (in a sense, allowing as little as possible) could often yield a (trivial) formal synthesis, a concept of qualitative optimality was introduced requiring that the controlled behavior be as rich as possible (*maximally permissive* or *minimally restrictive*) subject to the specification constraint. Therefore the need arose to identify the subclass of (regular) languages which, for given plant and specification, could be synthesized as just described, and within which an optimal behavior could be shown to exist. Thus the final and key ingredient of the theory (Ramadge & Wonham, 1982; 1987b) was the concept of *controllable language*, and the crucial fact that the controllable sublanguages of a given (specification) language admitted a *unique* maximal (or *supremal*) element. More technically, the legal controllable sublanguages form an upper semilattice under the partial ordering of language (i.e. string subset) inclusion. The solution of the formal problem of optimal control is thus precisely the *top* element of this semilattice, or *supremal controllable sublanguage* (of the legal language). The latter was shown to be effectively computable, indeed polynomial in the state size of the legal language, by an algorithm we shall later call 'Supcon'. The conceptual debt to previous geometric regulation theory was evident: while in geometric regulation theory one synthesizes subspaces, in SCDES one synthesizes languages; the synthesis technique is to use feedback.

Not uniquely in the annals of interdisciplinary research (Gold, 1989), the community response to this proposal ranged from indifference to hostility. Computer specialists dismissed the engineering application of FSM and regular languages as trivial and/or nothing new, while control specialists regarded FSM as impractical and/or irrelevant: "Finite automata," declared one anonymous reviewer for a leading journal, "have no place in control engineering." Eventually the archival paper was accepted by a third journal as a putative contribution to 'optimal control' (Ramadge & Wonham, 1987b).

3. Confronting the computational challenge

Attempts to apply the new theory to industrial problems encountered the barrier notorious as *exponential state space explosion*.[5] Thus a workcell with N machines each having k states would be modeled as a plant with *a priori* state size $\sim k^N$, so 10 machines each with 5 states would result in a global model with state count $5^{10} \sim 10$ million. Naive *extensional* representation of such systems (whereby the transitions are all listed and stored explicitly) rapidly becomes infeasible. As a first response, researchers therefore turned to 'smart architectures' involving horizontal and vertical modularity, or in systems terms decentralized (Ramadge & Wonham, 1987a) and hierarchical (Zhong & Wonham, 1990) decompositions, later including distributed control by supervisor localization (Cai & Wonham, 2010a) (described below).

The decentralized approach (Ramadge & Wonham, 1987a) adopts the view that the plant behavior is restricted by multiple local control specifications. The approach thus synthesizes a decentralized supervisor to enforce each of these specifications separately. In benign cases the synthesized decentralized supervisors work 'cooperatively', and their parallel operations achieve the same controlled behavior as the centralized, or 'monolithic', supervisor did (to enforce the overall specification). In general, however, decentralized supervisors are myopic' and unaware of their global interaction; the result may be conflict that leads to livelock (block-

[2] While in several ways (Arnold & Nivat, 1980) foreshadowed the publications (Ramadge & Wonham, 1982; 1987b) cited below, it was not until 1992 that the respective authors became aware of each others' work. We thank Prof. Arnold for helping to clarify this connection, and retrospectively Dr. Angelo Bean for mediating between our two communities which, at that time, were mutually rather isolated.

[3] That the two areas of control theory and automata theory shared ideas in common had been recognized for some time (see e.g. Arbib, 1965).

[4] The term *supervisor* for such a 'disabling agent' was adopted to distinguish it from *controller* (conventionally a 'forcing agent'). Nevertheless, forcing action can effectively be modeled within the theory when needed.

[5] Formal complexity studies were reported in Gohari and Wonham (2000) and Rohloff and Lafortune (2005).

ing) or even deadlock. The problem is thus to effectively coordinate the decentralized supervisors to ensure global nonblocking behavior, as the centralized supervisor would have guaranteed. Indeed, while control authority may ultimately be 'local', namely decomposable into specialized supervisors with authority over just a few plant components for enforcing a corresponding specification, to guarantee that these entities interact without mutual conflict means solving the global nonblocking problem, subject yet again to exponential computational effort. In other words, decentralized control typically requires global coordination, which threatens as before to be computationally infeasible.

On the other hand, the hierarchical approach (Zhong & Wonham, 1990) envisions that a 'manager' at the high level, being interested only in 'significant' dynamics of the plant at the low level, makes control decisions which are implemented through a hierarchical 'command-information' feedback loop. The approach first employs a causal (or prefix-preserving) map to create a high-level plant model (by aggregating significant dynamics of the low-level plant), which is optimistically much smaller in state size than the low-level model. Based on the high-level model, the approach then synthesizes a high-level supervisor (i.e. a manager) to enforce an imposed high-level specification. The expectation is that the controls of the manager, maximally permissive at the high level, will be faithfully implemented on the low-level plant. The manager may, however, be chagrined by the fact that the low-level controlled behavior is generally more conservative than expected. The cause lies in the possible inconsistency of information when creating the high-level plant using the causal map; as a cure, more refined dynamics must be included in the high-level model. This issue is called *hierarchical consistency* (Wong & Wonham, 1996a; Zhong & Wonham, 1990). To guarantee consistency, the price to be paid is increased state size of the high-level plant model, which could become even larger than the low-level model. Again this may render synthesis of the high-level supervisor computationally infeasible.

Several approaches, used singly or in combination, emerged to grapple with this issue. Essentially they sought to combine efficient system architecture with 'smart' computation. Thus the computational model of *state charts* (Harel, 1987) was adapted for control purposes in the version *state tree structures* (STS) (Ma & Wonham, 2005). These are layered (or hierarchical) models which make essential use of *intensional* (as distinct from extensional) representation of control functions using *binary decision diagrams* (BDDs) (Bryant, 1986).[6] With intensional representation a computable entity is stored not by tabling its values but instead providing an algorithm by which they are computed explicitly just when needed (as in the decimal representation of numbers, where '123' is stored instead of, say, a string of 1231's). Another efficient model class to be introduced was *extended state machines* (ESM) (Chen & Lin, 2000; Skoldstam, Akesson, & Fabian, 2007; Yang & Gohari, 2005), namely FSM parametrized by boolean and integer variables for logic elements and buffers, plus logic-based transition guards and variable assignments for succinct representation of state transitions. Other model types found useful included (bounded) *Petri nets* (Krogh, 1987) or *vector DES* (Li & Wonham, 1994) (either of these usually equivalent to a synchronous product of buffers),[7] especially when processed using FSM algorithms of Supcon type to achieve maximal permissiveness with nonblocking (Chen & Li, 2013). For a historical review of Petri nets, refer to Giua and Silva (2018).

4. Language observability and monolithic control with partial observations

Successful formalization of 'local' control structures rests on some notion of 'local observability'. In systems control *observability* is a property of a plant together with its output or observation structure, which guarantees that enough data about plant behavior (or current state) are available for implementation of a given class of controls (such as arbitrary state feedback controls). Specialized to our DES model, 'observation' has been modeled by a channel for transmission of the generated language strings from plant to supervisor. The simplest type of channel has zero memory, transmitting selected alphabet symbols one-by-one without change (in the case of *observable events*) or else erasing them altogether (the *unobservable events*). Formally, the channel is modeled by a *natural projection* from a language over a given alphabet to its image over a specified *observable subalphabet*.[8] A language is then said to be *observable*, with respect to a given plant and natural projection if, for every plant-generated string already in the language, its projection determines consistently whether a putative one-step (event) extension of the string remains a member of the language or not. It is shown that a controllable language can be synthesized (in a feedback loop with the plant) on observing only the projected generated strings (i.e. strings 'output by the channel') if and only if the language is observable (Lin & Wonham, 1988, Cieslak, Desclaux, Fawaz, & Varaiya, 1988). In the regular language framework observability is decidable in time polynomial in the state size of the targeted language (Rudie & Wonham, 1990; Tsitsiklis, 1989).

Conceptually, at least, the foregoing developments brought SCDES into the mainstream of systems control. Unfortunately, observability has turned out to be intractable for practical synthesis, the technical reason being that, unlike controllability, this property fails to be closed under language union; the upper semilattice algebraic structure that holds for controllable languages alone therefore fails; hence no optimal (unique maximally permissive) solution to the problem of *supervisory control under partial observations* (SCOP) need generally exist. Relaxing optimality to require only a 'maximal' solution is of no particular help inasmuch as a designer would generally have no idea where such a maximal element might be located in the landscape of solutions. Moreover, even when a meaningful observable sublanguage can be found, the construction of a supervisor that synthesizes that language cannot be achieved in polynomial time (Tsitsiklis, 1989).[9] These negative results related to control under partial observation are not unexpected: when an agent cannot determine which of two events occurred from the current state, it is forced to consider both paths that exit the state to examine all possible future outcomes.

In mitigation, conditions stronger than full observability have been proposed and, while a large catalog of realistic applications has yet to emerge, in many instances yield a useful result. The earliest and simplest was *normality* (Lin & Wonham, 1988), namely that a language is determined essentially by its inclusion in the plant and specification languages together with its image under the given natural projection for partial observation. It is shown that normality implies observability, and that the family of controllable normal languages admits a supremal element that is often feasibly computable despite exponential worst-case complex-

[6] For apparently the earliest application of symbolic computation to supervisory control see the article (Balemi, Hoffmann, Gyugyi, Wong-Toi, & Franklin, 1993).

[7] To model a bounded PN as a standard DES, determine the bound b_i on each place p_i, construct buffers B_i with capacity b_i, in the obvious way bring in state transitions σ_j among the B_i by inspection of the PN state transitions t_j, and finally take the synchronous product of the B_i.

[8] While a mapping called a *mask* is also used in the literature that appears more general than projection in that it maps the alphabet to a potentially different alphabet (and not merely a subalphabet), in fact, problems solvable using masks are equivalent to ones using projection.

[9] On the normal pessimistic assumption that NP $\neq$ P.

ity. The main shortcoming of this normality solution to SCOP is that an event can be considered controllable (i.e. subject to possible disablement) only if it is observable. In general this means that the supremal normality solution may be empty even though some observability solution is not, albeit whether or not the latter exists will in general be problematic. More recently, however, an improved condition of *relative observability* has been proposed (Cai, Zhang, & Wonham, 2015b), stronger than observability but strictly weaker than normality and with the same desirable property of closure under language union. Thus the family of controllable and relatively observable languages admits a supremal element, yielding a 'relatively' optimal solution to SCOP, which happily places no restriction on the disablement of unobservable controllable events. As with normality, this solution can be feasibly computable; examples have shown its practical utility as well as its generally greater permissiveness than the normality solution to SCOP.

Despite the computational challenges that partial observation entails, the more realistic setting of control under partial observation has led to the development of interesting concepts and research avenues.

In a loose analogy to 'detectability' in classical control systems, the notion of *detectability* was developed in SCDES to examine systems whose initial state is unknown and where observations are used to determine the state of the system (Shu, Lin, & Ying, 2007). A complementary body of work developed on failure diagnosis in DES (Sampath, Sengupta, Lafortune, Sinnamodhideen, & Teneketzis, 1995). In this model, faults or system failures are unobservable events and the goal is to determine, using subsequent observations, if and at what stage a failure has occurred. This work was motivated by the need for diagnostics in the automotive industry and for heating, ventilation and air conditioning systems.

The next stage in examining partial observation was to generalize the standard model to admit a dynamic notion of event observability, whereby instead of an event being observable or unobservable, individual occurrences of an event may be observable or unobservable (cf. Huang, Rudie, and Lin, 2008) where dynamic observation is defined for multiple supervisors—a special case of which is the partial observation under consideration here). This notion of *dynamic observation* is required for problems of *sensor activation* (Thorsley & Teneketzis, 2007) and system *opacity* (Bryans, Koutny, Mazaré, & Ryan, 2008; Mazaré, 2004)—a property needed in computer and network security.

Sensor activation involves dynamic control of event observability according to the current state and allowing for cost. The latter might be measured in battery charge usage, or sensor degradation with use, or the security risk if a sensor actually fails. A key challenge is activation policy implementation, which may entail an NFA-to-DFA conversion, with the attendant exponential state-space explosion. This familiar issue has led researchers to explore when special cases of partial observation problems do not, in fact, lead to an exponential number of states (e.g., NFA that satisfy the observer property (Wong & Wonham, 2004)) or to the need for NFA-to-DFA conversion (e.g., Sears & Rudie, 2013)—insights that could be of interest to the automata theory and languages community in theoretical computing.

5. Decentralized and distributed control with partial observations

Language observability was first applied to decentralized control via the extended concept of *co-observability* (Rudie & Wonham, 1992). The setup envisaged a global plant, together with a team of several isolated 'agents' each with an assigned subset of observable events (i.e. channel with corresponding natural projection) and an assigned subset of controllable events; an agent may share its observable and controllable events with other agents (i.e. agents' alphabets may possess elements in common). For a given controllable language meeting the (global) control specification, each agent is assumed independently to decide whether or not each 'next' controllable event should be enabled or disabled; it then communicates its decision to a central controlling authority. Employing one of several possible rules for 'decision fusion' (Yoo & Lafortune, 2002) the latter implements the collective control decision. The given language is defined to be *co-observable* if this decision is always correct; in the case of just one agent, co-observability reduces to observability. With several agents, a controllable language can then be synthesized in the described decentralized architecture; for this the co-observability property is both sufficient and necessary.

Unfortunately, just as with the monolithic setup, while co-observability is effectively decidable in time polynomial in the size of the target language (Rudie & Willems, 1995) it is not preserved by language union and in general a supremal (unique maximally permissive) solution to the decentralized SCOP fails to exist; nor is it obvious how any acceptable solution might be found, granting that one existed at all. Even when an acceptable co-observable sublanguage can be found, the construction of decentralized supervisors that synthesize that language cannot be achieved in polynomial time. In mitigation as before, co-observability may be relaxed to its counterpart *co-normality* (Dai & Lin, 2014), or more subtly to *relative co-observability* (Cai, Zhang, & Wonham, 2015a) provided the decision fusion rule is (severely) restricted to ensure the property of closure under union.

If the system fails to be co-observable, it is tempting to adjoin the stronger feature that agents be allowed to exchange information in accordance with an appropriate 'topology' of communication. Unfortunately again, design of the relevant protocols has turned out to be extremely difficult, owing to the interaction of agents' decisions due to the intertwining of communication and control. A fallback led to the simpler *state disambiguation* problem (Rudie, Lafortune, & Lin, 2003), though bringing with it the technical obstruction of non-monotonicity, namely that enhanced observation need not imply improved disambiguation (Wang, Lafortune, & Lin, 2008a). Here too, as in the setting of monolithic supervisors with partial observation, state-dependent (i.e., dynamic) observation with either or both observation and communication costs led to sensor activation problems and minimal communication problems where the relevant tradeoffs—with additional communication-specific concerns such as network bandwidth and network security—were optimized (cf. the survey (Sears & Rudie, 2016)).

A successful blend of architectural and observability concepts has been brought to bear in 'heterarchical' supervision, which exploits both decentralized and hierarchical control based on suitable system abstractions. First the given (large) system is split into smaller-scale subsystems, for which decentralized supervisors and coordinators (which enforce nonblocking) can be efficiently synthesized. Abstracted models of the resulting controlled subsystems are then computed by natural projections. These must be chosen to have the technical properties of being *natural observers* (Wong & Wonham, 2004; 1996a) and in some sense *control consistent* (Feng & Wonham, 2008; Schmidt & Breindl, 2011). The result is a hierarchical array of decentralized supervisors and coordinators that achieves global optimality with nonblocking. This approach has been demonstrated with the benchmark Production Cell, of state size 10^8 (Feng, Cai, & Wonham, 2009).

Another effective approach to distributed control has been introduced called *supervisor localization* (Cai & Wonham, 2010a). The latter envisages a plant composed (using *synchronous product*) of several modular components and a specification composed of several individual component specifications. By contrast with de-

centralized control, which usually means the allocation of separate specialized controls to separate component specifications, distributed control allocates separate controls to distinct plant components. This allocation (or *localization*) is achieved by decomposition of a given monolithic supervisor (or more generally each member of a given family of decentralized supervisors) by means of constructing suitable *control congruences* (Su & Wonham, 2004) (equivalence relations that respect both dynamics and control actions) on the relevant supervisor state sets. The result is to convert each plant component into a 'smart agent'. In general each agent communicates with an optimistically small number of (logical) 'neighbors' for exchange of information on event occurrence that is essential for control. This pattern of agent intercommunication is not assigned *a priori* but emerges as part of the problem solution. It is proved that the resulting distributed control behavior is identical with the monolithic or decentralized behavior adopted at the start, so if the latter is optimal then so is the localized result. There is thus no question of the latter's existence or in principle its feasible computation. In case the monolithic controller is too large to compute, localization may possibly be combined with the heterarchical approach described above (Cai & Wonham, 2010b). Further, in case some events are unobservable, localization of a partial-observation supervisor (synthesized by any effective method) may be carried out while respecting the requirement of feasible control actions under partial observation (Zhang, Cai, & Wonham, 2017).

Finally, as with any distributed control architecture, it has been deemed important to investigate its 'robustness' when inter-agent communication is subject to channel delay. The general problem of SCOP with several agents, no *a priori* architectural restrictions, and bounded or unbounded communication delay, has been proved to be undecidable (Tripakis, 2004). Nevertheless, more narrowly defined problems of this type have yielded useful insights (Lin, 2014; Zhang, Cai, Gan, Wang, & Wonham, 2016).

6. Extensions to SCDES with broader functionality

Several extensions of SCDES have been proposed for broader functionality and richer specifications. One of the earliest was based on temporal logic (Ramadge, 1989), allowing (among other things) the expression of 'eventuality', namely the occurrence or otherwise of some event 'in the long run' without regard to an *a priori* bound in time. This infinite behavior is found useful in modeling reactive and concurrent systems (Manna & Pnueli, 1992): for example, in concurrent programs each program will have access (upon request) to a computation resource eventually (*fairness*), or in multi-robot systems some robots will visit a critical area infinitely often (*liveness*). Similarly, specifications may be imposed on such infinite behavior to require that certain conditions be satisfied eventually, or some properties must hold infinitely often.

For supervisory control of DES with infinite behavior, the more technical setting is needed of *ω-languages* (Thistle & Wonham, 1994a; 1994b), which include infinite strings (as distinct from the regular languages of SCDES, whose strings are always finite albeit possibly of unbounded length). In this setting the plant is modeled again by an FSM but on infinite-length strings. Such FSM are called *ω-automata* (Thomas, 1990), for instance Büchi automata, Rabin automata, or Streett automata (which differ in acceptance conditions). In an ω-automaton, a subset of states is designated to be the *accepting criterion*, and a string is said to be accepted by the ω-automaton if it visits this state subset infinitely often.

In synthesizing a supervisor to enforce an imposed liveness specification, two language properties – ω-controllability and ω-closedness – are identified as central (Ramadge, 1989; Thistle & Wonham, 1994b). Like controllability of regular languages, ω-controllability enjoys the algebraic property that it is closed under arbitrary set unions. Unfortunately, ω-closedness turns out not to be closed under union (unlike closedness of regular languages). Consequently the supremal ω-controllable and ω-closed sublanguage of a given language need not exist in general, and the supremal ω-controllable sublanguage can be synthesized by a supervisor only if it also happens to be ω-closed. Despite the difficulty, a procedure is available (Thistle & Wonham, 1994b) to construct a supervisor (whenever it exists) that enforces liveness specifications on infinite-behavior DES while making control decisions only over finite-length strings. More recently, the issue of partial observation has also been addressed in this framework (Thistle & Lamouchi, 2009).

In a different direction and stimulated by earlier work (Ostroff & Wonham, 1985) on temporal logic, a timed version of DES, or TDES, was introduced (Brandin & Wonham, 1994), allowing the incorporation of event delays and deadlines as measured by a global digital clock, and including a notion of *forcible event* thought of as preempting the clock's tick. Specifically, in an FSM each event is considered to have a lower time bound and an upper time bound (possibly infinite). An event cannot occur before its timer has passed its lower bound (i.e. delay of occurrence), and must occur no later than the time of its upper bound (i.e. hard deadline). To explicitly model the temporal relations of events, the *timed transition graph* (TTG) is introduced, wherein a distinguished event 'tick' is employed to represent the tick of the global clock.

For a TDES, both logical and temporal specifications may be imposed. Logical specifications may be dealt with in the same fashion as in the untimed SCDES, namely by disabling suitable controllable events. Temporal specifications, however, generally require preempting the 'tick' event; for this a subset of *forcible* events is introduced which can be used, whenever available and as a last resort, to preempt occurrence of 'tick'. Thus a TDES supervisor is allowed not only to disable controllable events, but also to preempt 'tick' if there happen to be designated forcible events available. According to this new feature, timed controllability is introduced which, like its untimed counterpart, is closed under set unions. Hence the supremal (timed) controllable sublanguage of a given language exists, and can be synthesized with a TDES supervisor which is maximally permissive and nonblocking.

The TDES framework has been extended to the case of partial observation, where timed observability is introduced (Lin & Wonham, 1995). Like its untimed counterpart, timed observability is algebraically ill-behaved; consequently the same issues of partial-observation supervisory control are inherited by the timed case. A stronger property, timed relative observability, has been proposed (Cai, Zhang, & Wonham, 2016) and shown to be closed under set unions, and therefore to provide a relatively optimal solution to the timed SCOP. TDES were also extended to decentralized supervisory control, which includes multiple decentralized supervisors dedicated to enforcing imposed logical and temporal specifications. The decentralized supervisors are subject to possible global conflict (as in the untimed case); moreover, and unique to TDES, a decentralized supervisor might need to disable a controllable and forcible event (to enforce a logical specification), while another supervisor must use this event to preempt 'tick' (to enforce a temporal specification). The supervisors are called 'jointly coercive' if this problem does not arise, which is necessary for success of decentralized supervisory control of TDES.

An interesting generalization was to consider TDES in hierarchical control architecture, where the 'temporal frequencies' of different layers are different and become higher when descending down the hierarchy (Wong & Wonham, 1996b). To model this feature consider a two-layer hierarchy, where each layer is equipped with a distinct 'tick' event: $tick_{lo}$ and $tick_{hi}$, and these two 'tick' events are related by specifying that one occurrence of $tick_{hi}$ corresponds to k occurrences of $tick_{lo}$. Incorporating this hierarchical modeling, timed control structures are introduced and timed

hierarchical consistency is established. Finally distributed control of TDES was synthesized by the method of supervisor localization (Zhang, Cai, Gan, Wang, & Wonham, 2013). Here for each component agent, there is not only a local controller that disables controllable events belonging to this agent, but also a *local preemptor* that preempts the 'tick' event using forcible events of this agent. The collective behavior of these local controllers and local preemptors is identical to that of the global TDES supervisor.

An alternative approach via the notion of *timed automaton* (properly a generic term, but here with a specific definition) was proposed (Alur & Dill, 1990); this more technical setting admits multiple local clocks, analogous to the event 'timers' of TDES, but possibly measuring the 'real time' of physics.

Timed versions of some of the other DES representations noted above are an active area of current research.

7. Industrial applications

To conclude this historical overview we report that realistic industrial applications of SCDES are as yet few in number. This situation is due in part to a lack of experience among control engineers with modeling and specification in the framework of automata, but (more seriously) to the lack of software of industrial strength adapted to engineering design. While numerous applications have been proposed in the literature, relatively few have been demonstrated on actual hardware in a commercial environment. In any case, the widespread industrial technologies of *programmable logic controllers* (PLCs) and *sequential function charts* (SFCs) were utilized in experimental SCDES controllers at an early stage (Hellgren, Fabian, & Lennartson, 2001; Leduc & Wonham, 1995); and there is now convincing evidence that such a bridge between theory and practice is feasible. One of the first such applications was the testbed assembly process of the Atelier Interétablissement Productique (AIP) in Grenoble, France (Brandin & Charbonnier, 1994).

Another application with commercial implications has been the design of a telephone directory assistance call center (Seidl, 2006).

The power of SCDES in the control synthesis (as opposed to cut-and-try design) of a complex DES with over 6 billion states, namely the patient support system for a magnetic resonance image (MRI) scanner, has been impressively demonstrated in Theunissen, Petreczky, Schiffelers, van Beek, and Rooda (2013). A recent application of similar impressive realism is the control of a waterway (canal) lock system in Tilburg, the Netherlands (Reijnen, Goorden, van de Mortel-Fronczak, & Rooda, 2017).

Finally, interest in SCDES from the computer science community has led to DES researchers demonstrating that the tried-and-true techniques that characterize supervisory control—namely, the separation of plant from specifications and the synthesis of controllers that are correct-by-construction—can be applied to problems within the software engineering domain such as concurrency control (Auer, Dingel, & Rudie, 2014; Dragert, Dingel, & Rudie, 2008), deadlock avoidance in multi-threaded software (Liao et al., 2013; Wang, Kelly, Kudlur, Lafortune, & Mahlke, 2008b) (modeled in the framework of Petri nets instead of finite automata) and automated service composition (Atampore, Dingel, & Rudie, 2016).

Acknowledgment

This work was supported in part by the Natural Sciences and Engineering Research Council, Canada, Grant NSERC-DG-480599 and NSERC Discovery Grant RGPIN-2015-05699; JSPS KAKENHI Grant no. JP16K18122.

References

Alur, R., & Dill, D. (1990). Automata for modeling real-time systems. In *Proceedings 17th international colloquium on automata, languages and programming, Lecture notes on computer science (LNCS) no. 443* (pp. 322–335). Springer.

Arbib, M. A. (1965). A common framework for automata theory and control theory. *Journal Society of Industrial and Applied Mathematics (SIAM) Series A, Control, 3*(2), 206–222.

Arnold, A., & Nivat, M. (1980). Controlling behaviours of systems: some basic concepts and some applications. In *Theoretical Foundations of Computer Science, Lecture Notes on Computer Science (LNCS) No. 88, Springer-Verlag* (pp. 113–122).

Atampore, F., Dingel, J., & Rudie, K. (2016). Automated service composition via supervisory control theory. In *Proceedings 13th International Workshop on Discrete Event Systems* (pp. 28–35).

Athans, M., & Falb, P. (1966). *Optimal Control: An Introduction to the Theory and its Applications*. McGraw-Hill.

Auer, A., Dingel, J., & Rudie, K. (2014). Concurrency control generation for dynamic threads using discrete-event systems. *Science of Computer Programming, 82*, 22–43.

Aveyard, R. (1974). A boolean model for a class of discrete event systems. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-4*(3), 249–258.

Baeten, J. (2004). A brief history of process algebra. *Technical Report*. Rpt. CSR 04-02, Vakgroep Informatica, Technical University of Eindhoven, The Netherlands.

Balemi, S., Hoffmann, G. J., Gyugyi, P., Wong-Toi, H., & Franklin, G. F. (1993). Supervisory control of a rapid thermal multiprocessor. *IEEE Transactions Automatic Control, 38*(7), 1040–1059.

Basile, G., & Marro, G. (1991). *Controlled and Conditioned Invariants in Linear System Theory*. University of Bologna, Italy.

Beauquier, J., & Nivat, M. (1980). Application of formal language theory to problems of security and synchronization. In *R.V. Book (Ed.), Formal Language Theory - Perspective and Open Problems, Academic Press* (pp. 407–454).

Ben-Ari, M. (1982). *Principles of Concurrent Programming*. Prentice-Hall International.

Brandin, B., & Charbonnier, F. (1994). The supervisory control of the automated manufacturing system of the AIP. In *Proceedings 4th International Conference on Computer Integrated Manufacturing and Automation Technology* (pp. 319–324). Troy, NY, USA

Brandin, B., & Wonham, W. M. (1994). Supervisory control of timed discrete-event systems. *IEEE Transactions Automatic Control, 39*(2), 329–342.

Bryans, J., Koutny, M., Mazaré, L., & Ryan, P. (2008). Opacity generalised to transition systems. *International Journal of Information Security, 7*(6), 421–435.

Bryant, R. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions Computers, C-35*(8), 677–691.

Cai, K., & Wonham, W. M. (2010a). Supervisor localization: a top-down approach to distributed control of discrete-event systems. *IEEE Transactions Automatic Control, 55*(3), 605–618.

Cai, K., & Wonham, W. M. (2010b). Supervisor localization for large discrete-event systems – case study production cell. *International Journal Advanced Manufacturing Technology, 50*(9–12), 1189–1202.

Cai, K., Zhang, R., & Wonham, W. M. (2015a). On relative coobservability of discrete-event systems. In *Proceedings American Control Conference* (pp. 371–376). Chicago, IL, USA

Cai, K., Zhang, R., & Wonham, W. M. (2015b). Relative observability of discrete-event systems and its supremal sublanguages. *IEEE Transactions Automatic Control, 60*(3), 659–670.

Cai, K., Zhang, R., & Wonham, W. M. (2016). Relative observability and coobservability of timed discrete-event systems. *IEEE Transactions Automatic Control, 61*(11), 3382–3395.

Chen, Y., & Li, Z. (2013). *Optimal Supervisory Control of Automated Manufacturing Systems*. CRC Press.

Chen, Y.-L., & Lin, F. (2000). Modeling of discrete event systems using finite state machines with parameters. In *Proceedings 2000 IEEE International Conference on Control Applications* (pp. 941–946). Anchorage, AL, USA

Cieslak, R., Desclaux, C., Fawaz, A. S., & Varaiya, P. (1988). Supervisory control of discrete-event processes with partial observations. *IEEE Transactions Automatic Control, 33*(3), 249–260.

Dai, J., & Lin, H. (2014). A learning-based synthesis approach to decentralized supervisory control of discrete event systems with unknown plants. *Control Theory and Technology, 12*(3), 218–233.

Dijkstra, E. (1968). Cooperating sequential processes. Technical Report EWD-123, Technological University, Eindhoven, The Netherlands, 1965; reprinted in *Programming Languages*, F. Genuys, Eg., Academic Press, New York, 1968, 43–112.

Dragert, C., Dingel, J., & Rudie, K. (2008). Generation of concurrency control code using discrete-event systems theory. In *Proceedings 16th ACM Sigsoft International Symposium on Foundations of Software Engineering* (pp. 146–157).

Ephremides, A., Varaiya, P., & Walrand, J. (1980). A simple dynamic routing problem. *IEEE Transactions Automatic Control, 25*(4), 690–693.

Feng, L., Cai, K., & Wonham, W. M. (2009). A structural approach to the nonblocking supervisory control of discrete-event systems. *International Journal Advanced Manufacturing Technology, 41*(11), 1152–1167.

Feng, L., & Wonham, W. M. (2008). Supervisory control architecture for discrete-event systems. *IEEE Transactions Automatic Control, 53*(6), 1449–1461.

Fishman, G. (1978). *Principles of Discrete Event Simulation*. Wiley.

Giua, A., & Silva, M. (2018). Petri nets and automatic control: a historical perspective. *Journal of Annual Reviews in Control*.

Gohari, P., & Wonham, W. M. (2000). On the complexity of supervisory control design in the RW framework. *IEEE Transactions Systems, Man, and Cybernetics-Part B: Cybernetics, Special Issue on DES, 30*(5), 643–652.

Gold, T. (1989). New ideas in science. *Journal of Scientific Exploration, 3*(2), 103–112.

Harel, D. (1987). Statecharts: a visual formalism for complex systems. *Science of Computer Programming, 8*(3), 231–274.

Hellgren, A., Fabian, M., & Lennartson, B. (2001). Modular implementation of discrete event systems as sequential function charts applied to an assembly cell. In *Proceedings 2001 IEEE International Conference on Control Applications* (pp. 453–458). Mexico City, Mexico

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

Howard, R. (1960). *Dynamic Programming and Markov Processes*. The MIT Press.

Huang, Y., Rudie, K., & Lin, F. (2008). Decentralized control of discrete-event systems when supervisors observe particular event occurrences. *IEEE Transactions on Automatic Control, 53*(2), 384–388.

Krogh, B. (1987). Controlled Petri nets and maximally permissive feedback logic. In *Proceedings 25th Annual Allerton Conference on Communication, Control, and Computing* (pp. 317–326). Allerton, IL, USA

Leduc, R., & Wonham, W. M. (1995). Discrete event systems modeling and control of a manufacturing testbed. In *Proceedings 1995 Canadian Conference on Electrical and Computer Engineering* (pp. 793–796). Montreal, QC, Canada

Li, Y., & Wonham, W. M. (1994). Control of vector discrete-event systems, I-the base model; II-controller synthesis. *IEEE Transactions on Automatic Control, 38*(8). 1214–1227, 1993; 39(3), 512–531

Liao, H., Wang, Y., Stanley, J., Lafortune, S., Reveliotis, S., Kelly, T., et al. (2013). Eliminating concurrency bugs in multithreaded software: a new approach based on discrete-event control. *IEEE Transactions on Control Systems Technology, 21*(6), 2067–2082.

Lin, F. (2014). Control of networked discrete event systems: dealing with communication delays and losses. *SIAM Journal Control and Optimization, 52*(2), 1276–1298.

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

Lin, F., & Wonham, W. M. (1995). Supervisory control of timed discrete-event systems under partial observation. *IEEE Transactions Automatic Control, 40*(3), 558–562.

Ma, C., & Wonham, W. M. (2005). Nonblocking Supervisory Control of State Tree Structures. *Lecture Notes in Control and Information Sciences (LNCIS) No. 317*. Springer.

Manna, Z., & Pnueli, A. (1992). *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag.

Mazaré, L. (2004). Using unification for opacity properties. *Technical Report*. Verimag Technical Report.

Milner, R. (1989). *Communication and Concurrency*. Prentice-Hall.

Ostroff, J., & Wonham, W. M. (1985). A temporal logic approach to real time control. In *Proceedings 24th IEEE Conference on Decision and Control* (pp. 656–657). New York, NY, USA

Peterson, J. (1981). *Petri Net Theory and Modeling of Systems*. Prentice-Hall.

Ramadge, P. J. (1989). Some tractable supervisory control problems for discrete-event systems modeled by büchi automata. *IEEE Transactions on Automatic Control, 34*(1), 10–19.

Ramadge, P. J., & Wonham, W. M. (1982). Supervisory control of discrete event processes. In *Joint Workshop on Feedback & Synthesis of Linear & Nonlinear Systems, Istituto di Automatica, University di Roma, June 1981* (pp. 202–214). In D. Hinrichsen, A. Isidori (Eds.), Feedback Control of Linear and Nonlinear Systems. Lecture Notes on Control and Information Sciences (LNCIS), No. 39, Springer-Verlag

Ramadge, P. J., & Wonham, W. M. (1987a). Modular feedback logic for discrete event systems. *SIAM Journal Control and Optimization, 25*(5), 1202–1218.

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

Reijnen, F. F. H., Goorden, M. A., van de Mortel-Fronczak, J. M., & Rooda, J. E. (2017). Supervisory control synthesis for a waterway lock. In *Proceedings 1st Conference on Control Technology and Applications* (pp. 1562–1568). Big Island, HI

Rohloff, K., & Lafortune, S. (2005). PSPACE completeness of modular supervisory control problems. *Discrete Event Dynamic Systems, 15*(2), 145–167.

Rudie, K., Lafortune, S., & Lin, F. (2003). Minimal communication in a distributed discrete-event system. *IEEE Transactions Automatic Control, 48*(6), 957–975.

Rudie, K., & Willems, J. (1995). The computational complexity of decentralized discrete-event control problems. *IEEE Transactions Automatic Control, 40*(7), 1313–1319.

Rudie, K., & Wonham, W. (1990). The infimal prefix-closed and observable superlanguage of a given language. *Systems & Control Letters, 15*, 361–371.

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

Sampath, M., Sengupta, R., Lafortune, S., Sinnamodhideen, S., & Teneketzis, D. (1995). Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control, 40*(9), 1555–1575.

Schmidt, K., & Breindl, C. (2011). Maximally permissive hierarchical control of decentralized discrete event systems. *IEEE Transactions Automatic Control, 56*(4), 723–737.

Sears, D., & Rudie, K. (2013). Efficient computation of sensor activation decisions in discrete-event systems. In *Proceedings 52nd IEEE Conference on Decision and Control* (pp. 6966–6971). Florence, Italy

Sears, D., & Rudie, K. (2016). Minimal sensor activation and minimal communication in discrete-event systems. *Discrete Event Dynamic Systems, 26*(2), 295–349.

Seidl, M. (2006). Systematic controller design to drive high-load call centers. *IEEE Transactions Control Systems Technology, 14*(2), 216–223.

Shields, M. (1979). COSY train journeys. *Technical Report*. Rpt. ASM/67, Computing Laboratory, Univ. of Newcastle-upon-Tyne.

Shu, S., Lin, F., & Ying, H. (2007). Detectability of discrete event systems. *IEEE Transactions on Automatic Control, 52*(12), 2356–2359.

Skoldstam, M., Akesson, K., & Fabian, M. (2007). Modeling of discrete event systems using finite automata with variables. In *Proceedings 46th IEEE Conference Decision and Control* (pp. 3387–3392). New Orleans, LA

Su, R., & Wonham, W. M. (2004). Supervisor reduction for discrete-event systems. *Discrete Event Dynamic Systems, 14*(1), 31–53.

Theunissen, R., Petreczky, M., Schiffelers, R., van Beek, D., & Rooda, J. (2013). Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner. *IEEE Transactions Automation Science and Engineering, 11*(1), 20–32.

Thistle, J., & Wonham, W. M. (1994a). Control of infinite behavior of finite automata. *SIAM Journal Control and Optimization, 32*(4), 1075–1097.

Thistle, J., & Wonham, W. M. (1994b). Supervision of infinite behavior of discrete-event systems. *SIAM Journal Control and Optimization, 32*(4), 1098–1113.

Thistle, J. G., & Lamouchi, H. M. (2009). Effective control synthesis for partially observed discrete-event systems. *SIAM Journal on Control and Optimization, 48*(3), 1858–1887.

Thomas, W. (1990). Automata on infinite objects. In *Handbook of Theoretical Computer Science, vol. B: Formal Models and Semantics, J. van Leeuwen, eds., Elsevier, The MIT Press, Cambridge, MA* (pp. 134–191).

Thorsley, D., & Teneketzis, D. (2007). Active acquisition of information for diagnosis and supervisory control of discrete event systems. *Discrete Event Dynamic Systems, 17*(4), 531–583.

Tripakis, S. (2004). Decentralized control of discrete-event systems with bounded or unbounded delay communication. *IEEE Transactions Automatic Control, 49*(9), 1489–1501.

Tsitsiklis, J. N. (1989). On the control of discrete-event dynamical systems. *Mathematics of Control, Signals, and Systems, 2*(2), 95–107.

Wang, W., Lafortune, S., & Lin, F. (2008a). On the minimization of communication in networked systems with a central station. *Discrete Event Dynamic Systems, 18*, 415–443.

Wang, Y., Kelly, T., Kudlur, M., Lafortune, S., & Mahlke, S. (2008b). Gadara: dynamic deadlock avoidance for multithreaded programs. In *Proceedings 8th USENIX Symposium on Operating Systems Design and Implementation* (pp. 281–294).

Wong, K., & Wonham, W. (2004). On the computation of observers in discrete-event systems. *Discrete Event Dynamic Systems, 14*(1), 55–107.

Wong, K. C., & Wonham, W. M. (1996a). Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems, 6*(3), 241–273.

Wong, K. C., & Wonham, W. M. (1996b). Hierarchical control of timed discrete-event systems. *Discrete Event Dynamic Systems, 6*(3), 275–306.

Wonham, W. M. (1985). *Linear Multivariable Control: a Geometric Approach*. 3rd ed., Springer.

Yang, Y., & Gohari, P. (2005). Embedded supervisory control of discrete-event systems. In *Proceedings 2005 IEEE International Conference on Automation and Engineering* (pp. 410–415).

Yoo, T. S., & Lafortune, S. (2002). A general architecture for decentralized supervisory control of discrete-event systems. *Discrete Event Dynamic Systems, 12*(3), 335–377.

Zhang, R., Cai, K., Gan, Y., Wang, Z., & Wonham, W. M. (2013). Supervision localization of timed discrete-event systems. *Automatica, 49*(9), 2786–2794. Figures 1–9 are reprinted with kind permission from Elsevier

Zhang, R., Cai, K., Gan, Y., Wang, Z., & Wonham, W. M. (2016). Distributed supervisory control of discrete-event systems with communication delay. *Discrete Event Dynamic Systems, 26*(2), 263–293.

Zhang, R., Cai, K., & Wonham, W. M. (2017). Supervisor localization of discrete-event systems under partial observation. *Automatica, 81*(7), 142–147.

Zhong, H., & Wonham, W. M. (1990). On the consistency of hierarchical supervision in discrete-event systems. *Proceedings 8th Usenix Symposium on Operating Systems Design and Implementation, 35*(10), 1125–1134.