Chapter 3

Discrete Event Systems: The State of the Art and New Directions

Christos G. Cassandras
Stéphane Lafortune

ABSTRACT The goal of this paper is to present some perspectives on current and future research directions in the area of discrete event systems. For the benefit of readers unfamiliar with this field, we start with a brief survey of the modeling of discrete event systems along with some key results from the last decade that define the state of the art. We then comment on some new challenges for the "next generation" of discrete event system theory, highlighting issues of complexity and uncertainty, the emergence of hybrid systems, and the need for optimal control. Recent research on some of these issues is then discussed in more detail in four sections treating decentralized control and optimization, failure diagnosis, nondeterministic supervisory control, and hybrid systems and optimal control.

3.1 Introduction

The development of the theory of Discrete Event Systems (DES) is largely motivated by a wide range of application domains fostered in part by the rapid proliferation of computer technology. In particular, many 'human-made' systems, such as those encountered in manufacturing, communication networks, transportation, logistics, or the execution of software, all fall within the class of DES. There are two key features that characterize these systems. First, their dynamics are *event-driven* as opposed to *time-driven*, i.e., the behavior of a DES is governed only by occurrences of different types of events over time rather than by ticks of a clock. In fact, unlike conventional time-driven systems, the evolution of time between event occurrences has no visible effect on the system. Second, at least some of the natural variables required to describe a DES are discrete. Examples of *events* include the pushing of a button or an unpredictable computer failure. Examples of discrete variables involved in modeling a DES are descriptors of the state of a resource (e.g., UP, DOWN, BUSY, IDLE) or (integer-valued) counters for the number of users waiting to be served by a resource. From a mathe-

matical standpoint, a key implication of these two features is the following: much of the conventional differential/difference equation frameworks and associated analytical techniques (developed for the analysis of time-driven systems) can no longer be used for the purpose of modeling DES. To meet this major challenge, contributions from various disciplines have had to be brought together, including systems theory, operations research, computer science, and industrial engineering. Before proceeding, it is also worth mentioning that the acronym DEDS, for Discrete Event Dynamic System, rather than DES, is also often used to emphasize the critical role played by *dynamics* in studying these systems.

Historically, since the early 1980's, the development of DES theory has proceeded along three basic directions. First, one is interested in the 'logical behavior' of the system, i.e., ensuring that a precise *ordering of events* takes place which satisfies a given set of specifications. In this context, the actual timing of events is not relevant. Next, one does become interested in *event timing* when this is a crucial part of the design specifications (e.g., certain events are required to occur within certain critical time intervals). More generally, event timing is important in assessing the performance of a DES often measured through quantities such as 'throughput' or 'response time'. Finally, one cannot ignore the fact that DES frequently operate in a stochastic setting, hence necessitating the development of probabilistic models and related analytical methodologies for design and performance analysis. Although research along these directions has been concurrently and largely independently pursued, it is noteworthy that in recent years more and more interrelationships have been discovered, bringing forward some of the salient properties and inherent limitations of various modeling and analysis techniques developed to date. For example, Glasserman and Yao in [48] identify monotone structures as a unifying theme in the study of DES.

The first direction above, focusing on the logical behavior of DES, was launched by the work of Ramadge and Wonham [96, 94, 95], where a DES is modelled as the generator of a formal language of finite (possibly arbitrarily long) strings of event symbols, where the set of all possible events includes both controllable and uncontrollable events. Control features are introduced in this setting by means of a 'supervisor' mechanism which may enable/disable the controllable events in order to satisfy a set of qualitative specifications on the admissible orderings of the events that can be executed by the system. Furthermore, the supervisor may not be able to 'see' all of the events generated by the system due to the presence of unobservable events in the event set. This basic set-up has been extended to include modular control architectures (decentralized, distributed, and hierarchical) as well as system modeling using formal languages of infinite strings of events. The systems and control theory for DES developed in the above setting is known as *Supervisory Control Theory*; two excellent survey papers on supervisory control theory are [95, 109]. In addition to supervisory

control theory, we mention references [64, 108, 20, 88, 71] which provide a sample of other control-theoretic results related to the logical behavior of DES.

The second direction aims at DES models that explicitly incorporate event timing. There is a wide range of research that falls into this category, including work on programming languages with formal underlying mathematical models for the study of timed DES (such as the language SIGNAL [11]), work on control synthesis problems for various modeling formalisms of timed DES (cf. section 5 of [109] and [85]), and work on the development of analytical techniques for timed DES that parallel the success story of linear systems and control in the time-driven setting. The result of this last line of work is a framework known as the *Max-Plus Algebra*; in this framework, DES models are developed which are 'linear' not in the sense of the conventional algebra, but in the sense of an algebra based on the two operations: 'maximum' (or 'minimum') and 'addition' (or 'subtraction'). The roots of the max-plus algebra can be traced to the work of Cuninghame-Green in [37], which was used as the basis for the DES modeling setting proposed by Cohen et al. [36]. Subsequent work has identified connections to Petri nets and provided a variety of extensions; for a comprehensive source on the study of DES through the max-plus algebra, see [3].

The third direction is also motivated by the need to study event timing in DES and is driven by the fact that the state trajectory (sample path) of a DES observed under a given set of conditions contains a surprising amount of information about the behavior of the system under a spectrum of different conditions one might be interested in. This is particularly important when analytical models are unavailable or simply inadequate for DES of considerable complexity, especially in a stochastic environment. The roots of this direction may be found in the work of Ho *et al.* [59] and Ho and Cassandras [58], from which emerged the theory of *Perturbation Analysis* (PA) for DES [57, 47, 24]. More recently, this direction has concentrated on the development of efficient learning schemes based on information extracted from sample paths, with the ultimate goal of satisfying a set of quantitative specifications and optimizing the performance of a DES.

The objective of this paper is not so much to present an introduction to DES theory (the reader is referred to a recent paper [23] which attempts to accomplish this task), but rather to draw on the accomplishments of the past decade and the knowledge that has been gained, which, combined with recent technological developments, allow us to define what might be regarded as the 'next generation' of DES theory: new challenges, research directions, and key goals to meet over the next few years. With this objective in mind, the organization of the paper is as follows. Sections 3.2 and 3.3 provide some basic background on DES theory. In section 3.2 we review one of the modeling frameworks for DES to be used for the purposes of this paper, while section 3.3 highlights some important results obtained over the past decade. In section 3.4, we outline what we view as important

problems and new directions to pursue for DES research. The remainder of the paper is devoted to a presentation of some of these new research topics. In particular, section 3.5 focuses on decentralized approaches aimed at overcoming the tremendous computational complexity obstacles encountered when developing explicit control and optimization schemes for DES. Section 3.6 addresses the failure diagnosis problem for DES, where the objective is to diagnose the occurrence of "unobservable failure events", i.e., to resolve some uncertainty in the system behavior. In section 3.7 we examine the problem of uncertainty from the point of view of supervisory control theory, which needs to be extended to accommodate nondeterministic models of DES. Finally, in section 3.8 we discuss the emergence of hybrid systems and related issues in the context of optimal control.

3.2 DES Modeling Framework

As was mentioned in the introduction, the two defining characteristics of a DES are that the state space X of the system is a discrete set and the dynamics are event-driven, as opposed to time-driven. The set of possible events is denoted by Σ. Events occur asynchronously (in general) and cause transitions between the discrete states of the system. A "sample path" or "system trajectory" of a DES is a sequence of events with their times of occurrence. Several modeling formalisms are being used to represent the behavior of a DES, namely the set of all possible sample paths. The modeling formalism that we consider in this paper is that of *automata*. Other formalisms include Petri nets [83], process algebras [4, 68], and logic-based models.

A *Deterministic Automaton*, denoted by G, is a six-tuple

$$G = (X, \Sigma, f, \Sigma_G, x_0, X_m)$$

where

- X is the set of states of G.
- Σ is the set of events associated with the transitions in G.
- $f : X \times \Sigma \to X$ is the *partial* transition function of G: $f(x, e) = x'$ means that there is a transition labeled by event e from state x to state x'.
- $\Sigma_G : X \to 2^{\Sigma}$ is the active event function (or feasible event function): $\Sigma_G(x)$ is the set of all events e for which $f(x, e)$ is defined. $\Sigma_G(x)$ is called the *active event set* (or *feasible event set*) of G at x.
- x_0 is the *initial* state of G.

- $X_m \subseteq X$ is the set of *marked states* of X.

We make the following remarks about this definition.

1. The words *generator* (which explains the notation G) and *state machine* are also extensively used in the literature to describe the above object.

2. If X is finite, we call G a deterministic *finite* automaton, or DFA.

3. The automaton is said to be *deterministic* because f is a function from $X \times \Sigma$ to X. In contrast, the transition structure of a *nondeterministic* automaton is defined by means of a relation on $X \times \Sigma \times X$ or, equivalently, a function from $X \times \Sigma$ to 2^X, the power set of X.

4. The fact that we allow the transition function f to be partially defined over its domain $X \times \Sigma$ is a variation over the standard definition of a DFA in automata theory that is quite important in DES theory.

5. By designating certain states as marked, we record that the system, upon entering these states, has completed some operation or task. Marked states are used to study the issue of *blocking* (deadlock or livelock) in DES control problems (e.g., see [29]).

The automaton G operates as follows. It starts in the initial state x_0 and upon the occurrence of an event $e \in \Sigma_G(x_0)$ it will make a transition to state $f(x_0, e) \in X$, or to one of the states in $f(x_0, e)$ if the automaton is nondeterministic. This process then continues based on the transitions for which f is defined.

Let us extend the above definition of automaton in order to include timing information about event occurrences; we will call the resulting object a *timed automaton*. We associate to every event $e \in \Sigma_G(x)$ a *clock* value (or *residual lifetime*) y_e, which represents the amount of time required until event e occurs, with the clock running down at unit rate. The clock value of event e always starts with a *lifetime*, which is an element of an externally provided clock sequence $\mathbf{v}_e = \{v_{e,1}, v_{e,2}, \ldots\}$; we view this as an input to the automaton G. In other words, our model is endowed with a *clock structure*, defined as the set of event lifetime sequences $V = \{\mathbf{v}_e, e \in \Sigma\}$. A timed automaton is a seven-tuple $G_t = (G, V) = (X, \Sigma, f, \Sigma_G, x_0, X_m, V)$.

Let us informally describe how a timed automaton operates (for details, see [24]). If the current state is x, we look at all clock values y_e, $e \in \Sigma_G(x)$. The *triggering event* e' is the event which occurs next at that state, i.e., the event with the smallest clock value:

$$e' = \arg\min \{y_e, \quad e \in \Sigma_G(x)\} . \tag{3.1}$$

Once this event is determined, the next state, x', is specified by $x' = f(x, e')$. More generally, f can be replaced by the transition probabilities

$p(x'; x, e)$ where $e \in \Sigma_G(x)$. The amount of time spent at state x defines the *interevent time* (between the event that caused a transition into x and the event e'):

$$y^* = \min \{y_e, \quad e \in \Sigma(x)\} \ . \tag{3.2}$$

Thus, time is updated through $t' = t + y^*$, where t is the time when the system entered state x. Clock values are updated through $y'_e = y_e - y^*$, except for e' and any other events which were not feasible in x but become feasible in x'. For any such event, the clock value is set to a new lifetime obtained from the next available element in the event's clock sequence.

The final addition that we do to our model is to replace the clock structure V by a set of probability distribution functions $F = \{F_e, \quad e \in \Sigma\}$. The resulting object is called a *stochastic timed automaton* and is denoted by $G_{st} = (G, F) = (X, \Sigma, f, \Sigma_G, x_0, X_m, F)$. In this case, whenever a lifetime for event e is needed, we obtain a sample from F_e. The state sequence generated through this mechanism is a stochastic process known as a *Generalized Semi-Markov Process* (GSMP) (see also [47, 57]).

The stochastic timed automaton G_{st} provides the framework for generating sample paths of stochastic DES. A sample path is a sequence $\{e_k, t_k\}, \quad k = 1, 2, \ldots$, where e_k is the kth event taking values from Σ and t_k is its occurrence time. We will call this sequence a *timed trace* of events of system G_{st}. Let us call a *timed language* a set of timed traces of events of the above form and a *stochastic timed language* a timed language together with associated probability distribution functions characterizing the lifetime process for each event. The stochastic timed language represents all possible sample paths of the system together with statistical information about this set. The stochastic timed automaton G_{st} then serves as a *representation* of the stochastic timed language of interest. This type of modeling is the most detailed as it contains event information in the form of event occurrences and their orderings, timing information about the exact times at which the events occur (and not only their relative ordering), and statistical information about the probabilities of sample paths. If we omit the statistical information, then the corresponding timed language enumerates all the possible sample paths of the DES (with timing information). Finally, if we project out the timing information from a timed language we obtain an *untimed language*, or simply *language*, which is the set of all possible orderings of events (i.e., traces) that could happen in the given system. The language of the system is represented by the automaton G.

The above modeling process establishes a precise connection between the two important concepts of automata and GSMP. Researchers have also considered other means of introducing timing information into automata than the event-based clock structure V used to build G_t above. We will not discuss these other timed models here; the interested reader is referred to section 5 of [109] for more discussion on this issue.

Languages, timed languages, and stochastic timed languages represent

the three levels of abstraction at which DES are modeled and studied: untimed (or logical), timed, and stochastic. The choice of the appropriate level of abstraction clearly depends on the objectives of the analysis, namely: are we interested in purely logical properties of the behavior of the system, in properties that involve timing information, or in properties that involve expected behavior? These three levels of abstraction complement one another as they address different issues about the behavior of the DES. In fact, the literature in DES is quite broad and varied as extensive research has been done on modeling, analysis, control, optimization, and simulation at all three levels of abstraction.

We now discuss further the notion of (untimed) languages and its connection with automata. When formalizing the notion of language, we must distinguish between *finite* traces of events and *infinite* traces of events. In this brief review, we will restrict attention to languages of finite (albeit possibly arbitrarily long) traces, as this will suffice for our discussion of analysis and control problems at the logical level of abstraction in the following sections. The framework of finite traces is used for the study of problems of analysis and control involving *safety* properties (i.e., avoiding illegal states or illegal subtraces) as well as for the issue of *blocking* (e.g., deadlock and certain forms of livelock) in DES control (cf. [23]). The framework of infinite traces is considered when *liveness* properties are of concern (cf. [95, 109]).

Denote by Σ^* the set of all finite traces of elements of Σ, including the empty trace ϵ; the * operation is called the Kleene closure. For example, if $\Sigma = \{a, b, c\}$, then

$$\Sigma^* = \{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, \ldots\} .$$

For the sake of convenience, the transition function f of G is extended from domain $X \times \Sigma$ to domain $X \times \Sigma^*$ in the following recursive manner:

$$f(x, \epsilon) = x,$$

$$f(x, s\sigma) = f(f(x, s), \sigma) \quad \text{for } s \in \Sigma^* \quad \text{and } \sigma \in \Sigma .$$

This leads us to the notions of the languages *generated* and *marked* by automaton G.

- The language *generated* by G is

$$\mathcal{L}(G) \ := \ \{s \in \Sigma^* : f(x_0, s) \text{ is defined } \} .$$

- The language *marked* by G is

$$\mathcal{L}_m(G) \ := \ \{s \in \mathcal{L}(G) : f(x_0, s) \in X_m\} .$$

The purpose of the marked language is to represent those traces in the system behavior that correspond to a "completed" task in the sense that they bring G to a marked state.

Given any language $K \subseteq \Sigma^*$, we can always construct an automaton – albeit not necessarily a finite-state automaton– that marks K: for instance, simply build the automaton as an infinite tree whose root is the initial state and where the nodes at layer n are entered by the traces of length n. However, it is well-known that not all subsets of Σ^* can be represented by *finite* automata. A language K is said to be *regular* if there exists a DFA G that marks it, i.e., $\mathcal{L}_m(G) = K$. We often need to represent languages with *finite* automata, e.g., when the representation has to be stored in memory for performing calculations on it or simply for storage of a control policy. For this reason, the class of regular languages is of special interest in the study of DES.

We conclude this section by presenting an operation called the *synchronous composition* (or *parallel* composition) that captures the joint operation of two interconnected automata (possibly with common events). This operation is denoted by $||$. Consider $G_1 = (X_1, \Sigma_1, f_1, \Sigma_{G_1}, x_{01}, X_{m1})$ and $G_2 = (X_2, \Sigma_2, f_2, \Sigma_{G_2}, x_{02}, X_{m2})$. Then:

$$G_1 \,||\, G_2 := (X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f, \Sigma_{G_{1||2}}, (x_{01}, x_{02}), X_{m1} \times X_{m2})$$

where

$$f((x_1, x_2), e) := \begin{cases} (f_1(x_1, e), f_2(x_2, e)) & \text{if } e \in \Sigma_{G_1}(x_1) \cap \Sigma_{G_2}(x_2) \\ (f_1(x_1, e), x_2) & \text{if } e \in \Sigma_{G_1}(x_1) \setminus \Sigma_2 \\ (x_1, f_2(x_2, e)) & \text{if } e \in \Sigma_{G_2}(x_2) \setminus \Sigma_1 \\ \text{undefined} & \text{otherwise,} \end{cases}$$

and thus

$$\Sigma_{G_{1||2}}(x_1, x_2) = [\Sigma_{G_1}(x_1) \cap \Sigma_{G_2}(x_2)] \cup [\Sigma_{G_1}(x_1) \setminus \Sigma_2] \cup [\Sigma_{G_2}(x_2) \setminus \Sigma_1] \,.$$

In a synchronous composition of two automata, a common event, i.e., an event in $\Sigma_1 \cap \Sigma_2$, can only be executed if the two automata both execute it simultaneously. Thus the two automata are "synchronized" on the common events. The other events, i.e., those in $(\Sigma_2 \setminus \Sigma_1) \cup (\Sigma_1 \setminus \Sigma_2)$, are not subject to such a constraint and can be executed whenever possible. The definition of $||$ is extended to more than two automata in a straightforward manner.

3.3 Review of the State of the Art in DES Theory

3.3.1 Supervisory Control

Supervisory control is the situation where control is exerted on a given uncontrolled DES in order to satisfy the given specifications on the languages

generated and marked by the controlled system. The point of view adopted is that some of the uncontrolled behavior is *illegal* and thus must not be allowed under control. The control task is complicated by the fact that the controller, or *supervisor*, has limited capabilities in the sense that in general it cannot prevent by control all of the events in Σ nor observe the occurrence of all of these events. We denote the set of *uncontrollable* events by Σ_{uc} and the set of *observable* events by Σ_o. This leads to the feedback loop depicted in Figure 3.1.

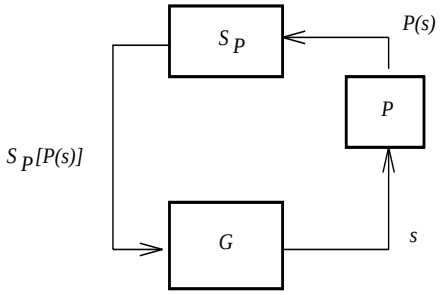

FIGURE 3.1. The feedback loop of supervisory control

In the figure, G is the system under the control of the supervisor S_P while P represents a "projection" operation that simply *erases* all of the *unobservable* events (i.e., those in $\Sigma \setminus \Sigma_o$) executed by G. The control action of the supervisor, $S_P[P(s)]$, is the set of events enabled by the supervisor after it has observed the trace of events $P[s]$. Only events in $\Sigma \setminus \Sigma_{uc}$, i.e., *controllable* events, can be excluded from this set. At any time, G can only execute an event that is currently enabled by the supervisor. The control action of the supervisor can be updated whenever G executes an observable event.

This general control paradigm was initiated by Ramadge & Wonham in 1982 [96] and studied extensively since then by themselves, their co-workers, and several other researchers; this body of work is known as *Supervisory Control Theory*. We will briefly mention some basic results of this theory for the case of untimed models of DES, namely languages represented by automata. We note that the system-theoretic results of supervisory control theory are more naturally stated in the language domain, while the synthesis and computational results are more naturally stated in the (finite) automaton domain. For further details on supervisory control theory as well as for many extensions of the basic results discussed here, including extensions to modular control architectures and to timed language models, we refer the reader to [95, 23, 72, 109] and the references therein. We emphasize that a large body of control-theoretic results for DES has been developed in the context of other modeling formalisms, most notably Petri nets (cf. the excellent survey paper [64]) and logic models (e.g., the COCOLOG theory

[20]). The choice of a different modeling formalism means, among other things, that the system structure will be exploited in a different manner in the ensuing analysis and synthesis operations, as compared with the finite automata employed either directly (e.g., as in [116]) or indirectly through symbolic representations (e.g., as in [5]) in the computational results of supervisory control theory.

The closed-loop system depicted in Figure 3.1 is denoted by S_P/G, for "S_P controlling G", or simply S/G when all of Σ is observable by G. The subset of $\mathcal{L}(G)$ that is possible under the control of S_P is the language $\mathcal{L}(S_P/G)$. The language *marked* by S_P/G is defined as follows:

$$\mathcal{L}_m(S_P/G) := \mathcal{L}(S_P/G) \cap \mathcal{L}_m(G) \ ,$$

i.e., it consists exactly of the marked traces of G that survive under the control of S_P. S_P is said to be *nonblocking* if

$$\mathcal{L}(S_P/G) = \overline{\mathcal{L}_m(S_P/G)} \ ;$$

otherwise, S_P is said to be *blocking*. Since marked traces represent completed tasks or record the completion of some particular operation (by choice at modeling), blocking means that the controlled system cannot terminate the execution of the task at hand.

A basic result in supervisory control theory is the characterization of what sublanguages of a given system language are achievable under control.

Basic Controllability and Observability Theorem: Consider DES G where $\Sigma_{uc} \subseteq \Sigma$ is the set of uncontrollable events and $\Sigma_o \subseteq \Sigma$ is the set of observable events. Let P be the projection from Σ^* to Σ_o^*. Consider also the language $K \subseteq \mathcal{L}_m(G)$, where $K \neq \emptyset$.

There exists a *nonblocking* supervisor S_P for G such that

$$\mathcal{L}_m(S_P/G) = K$$

iff the three following conditions hold:

1. K is *controllable* w.r.t. $\mathcal{L}(G)$ and Σ_{uc};
2. K is *observable* w.r.t. $\mathcal{L}(G)$, P, and Σ_c;
3. K is $\mathcal{L}_m(G)$-closed.

We briefly discuss the three conditions in this theorem.

Marking. Condition 3 is technical in nature and has to do with the marking of the traces in K as compared with that in $\mathcal{L}_m(G)$. It requires that

$$K = \overline{K} \cap \mathcal{L}_m(G)$$

where the notation $\overline{K}$ denotes taking the prefix-closure of the language K.

Controllability. The notion of controllability in supervisory control is defined as follows: K is *controllable* w.r.t. $\mathcal{L}(G)$ and Σ_{uc} if

$$\overline{K}\Sigma_{uc} \cap \mathcal{L}(G) \subseteq \overline{K} \ .$$

This condition is very intuitive. It means that a given language can be achieved by control iff there are no continuations, by uncontrollable events, of traces in the language to traces outside the language but in the uncontrolled behavior.

Observability. The notion of observability is also intuitive. In words, observability means that if two traces "look" the same from the point of view of the supervisor (i.e., after their unobservable events are erased by P), then the required control actions after these two traces should not be "inconsistent". The formalization of observability is slightly more involved than that of controllability and proceeds as follows.

- *Nextact Relation:* Given languages M ($M = \mathcal{L}(G)$ in our context) and K over the event set Σ, where $K \subseteq M$, $nextact_{K,M}$ is a ternary relation on $\Sigma^* \times \Sigma \times \Sigma^*$ that is defined as follows:

$$(s, \sigma, s') \in nextact_{K,M} \quad \text{if}$$
$$s\sigma \in \overline{K} \wedge s' \in \overline{K} \wedge s'\sigma \in M \Rightarrow s'\sigma \in \overline{K} \ .$$

- *Observability:* Let K and M be languages over an event set Σ. Let Σ_c be a designated subset of Σ. Let Σ_o be another designated subset of Σ with P as the corresponding projection from Σ^* to Σ_o^*. K is said to be *observable* w.r.t. M, P, and Σ_c if for all $s, s' \in \Sigma^*$,

$$P(s) = P(s') \Rightarrow$$
$$(\forall \sigma \in \Sigma_c)[(s, \sigma, s') \in nextact_{K,M} \ \wedge \ (s', \sigma, s) \in nextact_{K,M}] \ .$$

Controllability and observability are fundamental notions in supervisory control, and they have been studied extensively in the literature. Of particular interest is the computation of controllable and observable *sublanguages* of a given language that does not possess these properties. Such computations are central to the solution of supervisory control problems. One such problem is the *Basic Supervisory Control Problem - Nonblocking Version*, or BSCP-NB. In this problem, all of the events are observable and thus observability is not an issue. The formulation of BSCP-NB is as follows.

BSCP-NB: Given DES G, $\Sigma_{uc} \subseteq \Sigma$, and legal marked language $L_{am} \subseteq \mathcal{L}_m(G)$, with L_{am} assumed to be $\mathcal{L}_m(G)$-closed, build *nonblocking* supervisor S such that:

1. $\mathcal{L}_m(S/G) \subseteq L_{am}$;

2. $\mathcal{L}_m(S/G)$ is *as large as possible*, i.e., for any other nonblocking S' such that $\mathcal{L}_m(S'/G) \subseteq L_{am}$, $\mathcal{L}(S'/G) \subseteq \mathcal{L}(S/G)$.

Solution of BSCP-NB: Due to requirement 2, we call the desired solution S the *minimally restrictive nonblocking solution*. The solution is to choose S such that

$$\mathcal{L}(S/G) = \overline{L_{am}^{\uparrow}}$$

as long as $L_{am}^{\uparrow} \neq \emptyset$. The notation $\uparrow$ means the *supremal controllable sublanguage* of the given language with respect to $\mathcal{L}(G)$ and Σ_{uc}. The two seminal references on BSCP-NB and the notion of supremal controllable sublanguage are [94, 116].

Note that it can be shown that $L_{am}^{\uparrow}$ is $\mathcal{L}_m(G)$-closed whenever L_{am} is $\mathcal{L}_m(G)$-closed (an assumption in BSCP-NB). This fact guarantees that choosing S as above indeed results in a nonblocking closed-loop system.

If $L_{am}^{\uparrow}$ is regular, then the desired supervisor S can be *realized* by building a DFA representation of the language $\overline{L_{am}^{\uparrow}}$; if we denote this automaton by R, then the feedback loop of Figure 3.1 is then formally equivalent to the composition $R||G$ (see, e.g., [23]). This provides a useful interpretation of the feedback loop of Figure 3.1 in terms of the widely-used $||$ operation.

While the above discussion has only touched a very small part of the large body of work in supervisory control theory, it should allow the reader unfamiliar with this area of research to understand the related discussion in the remainder of this paper.

3.3.2 Max-Plus Algebra

A close look at the timed automaton model presented in section 3.2 reveals that there are two mathematical operations that play a key role in DES theory: 'maximum' (or 'minimum') and 'addition' (or 'subtraction'). In (3.1) and (3.2), for instance, we can see that the minimum operation is required to determine the event responsible for the next state transition, followed by a simple addition operation to update time through $t' = t + y^*$. Letting $x_i(k)$ denote the time when the kth occurrence of an event of type i takes place, we can obtain the following general form for state equations characterizing the event time dynamics of a large class of DES known as "event graphs" [3]:

$$\begin{aligned} x_i(k+1) &= \max(a_{i1} + x_1(k), a_{i2} + x_2(k), \ldots, a_{in} + x_n(k)) \\ &= \max_j(a_{ij} + x_j(k)), \quad i = 1, \ldots, n \ . \end{aligned} \tag{3.3}$$

The coefficients a_{ij} may be interpreted as the amount of time required for an event of type j to trigger the next occurrence of an event of type i. For instance, if i represents processing completions at the ith workstation of a manufacturing system, then a_{ii} is the processing time at i, while a_{ij}, $j \neq i$, is the processing time at other workstations that can affect i; if they

cannot affect it, then we set $a_{ij} = -\infty$. This representation gives rise to the 'max-plus' algebra, in which it is conventional to represent addition $+$ by $\otimes$ and max by $\oplus$. This choice is motivated by the fact that the equation above can be rewritten in a form resembling a standard linear difference equation

$$x_i(k+1) = \bigoplus_j (a_{ij} \otimes x_j(k)), \quad i = 1, \ldots, n \ , \tag{3.4}$$

or, in vector notation,

$$x(k+1) = A \otimes x(k), \tag{3.5}$$

where the symbol $\otimes$ is usually omitted once the meaning of "multiplication" in the max-plus algebra is established.

More generally, a linear system with inputs and outputs in the max-plus algebra is described by

$$\left.\begin{array}{rcl} x(k+1) & = & Ax(k) \oplus Bu(k) \ , \\ y(k) & = & Cx(k) \ , \end{array}\right\} \tag{3.6}$$

which is short-hand notation for

$$\begin{array}{rcll} x_i(k+1) & = & \max(a_{i1} + x_1(k), \ldots, a_{in} + x_n(k), & \\ & & \quad b_{i1} + u_1(k), \ldots, b_{im} + u_m(k)), & i = 1, \ldots, n \ ; \\ y_i(k) & = & \max(c_{i1} + x_1(k), \ldots, c_{in} + x_n(k)), & i = 1, \ldots, p \ . \end{array}$$

A generalization of (3.5) is

$$x(k+1) = A_0 x(k+1) \oplus A_1 x(k) \oplus \cdots \oplus A_{l+1} x(k-l), \tag{3.7}$$

which can also be rewritten as

$$x(k+1) = A_0^* A_1 x(k) \oplus \cdots \oplus A_0^* A_{l+1} x(k-l) \tag{3.8}$$

where

$$A_0^* := I \oplus A_0 \oplus A_0^2 \oplus A_0^3 \oplus \cdots$$

The notation I refers to the identity matrix in the max-plus algebra: it has zeros on the main diagonal and $\varepsilon = -\infty$ elsewhere. This equation can be rewritten as a set of first-order difference equations by simply augmenting the state space as in conventional linear system theory.

Periodic Behavior. As in standard linear systems, a critical goal of the max-plus algebra is to study the existence of eigenvalues and eigenvectors, i.e., the existence of λ and $v \neq \varepsilon$ such that:

$$Av = \lambda v \ . \tag{3.9}$$

where A is a square matrix. Interpreted in the max-plus algebra sense, v corresponds to an initial state resulting in a solution with 'period' 1, while λ corresponds to an interevent time.

In order to state one of the main results of the max-plus algebra, let $\mathcal{G}(A)$ denote the *precedence graph* of an $n \times n$ matrix A, defined as a weighted digraph with n nodes and an arc (j,i) if $a_{ij} \neq \varepsilon$, in which case the weight of this arc receives the numerical value of a_{ij}. Any weighted digraph $\mathcal{G} := (\mathcal{V}, \mathcal{E})$, with $\mathcal{V}$ being the set of nodes and $\mathcal{E}$ being the set of arcs, is the precedence graph of an appropriately defined square matrix. If a path is denoted by ρ, then the *mean weight of the path* is defined as the sum of the weights of the individual arcs of this path (denoted by $|\rho|_{\mathrm{w}}$), divided by the length of this path (denoted by $|\rho|_{\mathrm{l}}$). If such a path is a circuit one talks about the mean weight of the circuit, or simply the *cycle mean*. The *maximum cycle mean* is the maximum over all the cycle means in such a graph. If the cycle mean of a circuit equals the maximum cycle mean, then the circuit is called *critical*. The graph consisting of all critical circuits (if there happen to be more than one) is called the *critical graph* and denoted by $\mathcal{G}^{\mathrm{c}}$. Finally, a graph is said to be *strongly connected* if there exists a path from any node to any other node; the corresponding matrix is called *irreducible*.

With these definitions in mind, the following is a key result pertaining to a square matrix A:

Theorem: If $\mathcal{G}(A)$ is strongly connected, then there exists one and only one eigenvalue and at least one eigenvector. The eigenvalue is equal to the maximum cycle mean of the graph:

$$\lambda = \max_{\zeta} \frac{|\zeta|_{\mathrm{w}}}{|\zeta|_{\mathrm{l}}} \quad ,$$

where ζ ranges over the set of circuits of $\mathcal{G}(A)$.

A matrix A is said to be cyclic if there exist scalars M, λ and d such that $\forall m \geq M$, $A^{m+d} = \lambda^d A^m$. The least such d is called the cyclicity of A. The quantity λ equals the maximum cycle mean of A. It can then be shown that

Theorem: Any irreducible matrix is cyclic. The cyclicity of the irreducible matrix A equals the cyclicity of $\mathcal{G}^{\mathrm{c}}(A)$, being the critical graph corresponding to matrix A.

Computational Issues. In practice, we are obviously interested in developing efficient numerical procedures to obtain the eigenvalue and eigenvector of a matrix A in (3.9). An important result in this respect is known as Karp's theorem [69], which allows us to compute the maximum cycle mean of an $n \times n$ matrix A with corresponding precedence graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The issue of numerical approaches for such calculations remains a crucial one that is still attracting considerable research, including studying the zero(s) of the characteristic equation in the max-plus algebra [3]; using linear programming techniques [82]; and "power algorithms" as in [15].

3.3.3 Sample Path Analysis and Performance Optimization

The fact that a large part of DES theory developed to date is related to the study of sample paths of these systems is motivated by two main observations. First, when it comes to evaluating the performance of DES as a function of design or control parameters of interest, expressed as $J(\theta)$, closed-form expressions for $J(\theta)$ are seldom available; this is due to the complexity of these systems, which makes accurate modeling through analytical techniques very difficult. Therefore, it is natural to resort to simulation or direct observation, which involves generating and studying state trajectories or sample paths. Second, it turns out that sample paths of DES observed under a particular parameter setting θ contain a surprising amount of information about the performance of the system under different settings $\theta' \neq \theta$.

The second observation above has been greatly exploited in the context of *sensitivity estimation* for DES, which is a natural first step towards the development of optimization schemes. We shall briefly overview below the main accomplishments associated with these aspects of DES theory.

Sensitivity Analysis. Assuming a scalar parameter θ takes real values and $J(\theta)$ is differentiable, an obvious way to estimate a sensitivity of the form $dJ/d\theta$ is to estimate finite-difference approximations of the form $\Delta J/\Delta\theta$ through either simulation or direct observation of a sample path as follows. Let $J(\theta) = E[L(\theta)]$, where $L(\theta)$ is the performance obtained over a specific sample path observed under θ. Next, perturb θ by $\Delta\theta$ and observe $L(\theta+\Delta\theta)$. Finally, $[L(\theta+\Delta\theta)-L(\theta)]/\Delta\theta$ is an estimate of $dJ/d\theta$. However, this approach involves $(n+1)$ sample paths if there are n parameters of interest, and it quickly becomes prohibitively time-consuming. In addition, accurate gradient estimation requires "small" $\Delta\theta$; however, division by a small number in $\Delta J/\Delta\theta$ leads to a host of numerical problems. This has motivated the effort towards derivative estimation based on information from a single observed sample path, and has led to *Perturbation Analysis* (PA) and *Likelihood Ratio* (LR) techniques (see [47, 57, 24, 22, 98]).

In what follows, let us denote a *sample path* of a DES by (θ, ω) where θ is a vector of parameters characterizing the state transition function and/or the clock structure V (as defined in the stochastic timed automaton model presented in section 3.2), and ω represents all random occurrences in the system of interest. In particular, we let the underlying sample space be $[0,1]^\infty$, and ω a sequence of independent random variables uniformly distributed on $[0,1]$. Given a sample path (θ, ω), we can evaluate a sample performance function $L(\theta, \omega)$ via a statistical experiment, i.e., a discrete-event simulation run or data collected from an actual system in operation. This, then, serves as an estimate of the actual performance measure of interest, which we consider to be the expectation $J(\theta) = E[L(\theta, \omega)]$. In Perturbation Analysis (PA), we are interested in questions of the form "what would the

effect of a perturbation $\Delta\theta$ be on the system performance?" Thus, we use the adjectives *nominal* and *perturbed* to qualify the performance measures $L(\theta)$ and $L(\theta+\Delta\theta)$, as well as the corresponding sample paths (θ,ω) and $(\theta+\Delta,\omega)$.

The first important result in sample path analysis of DES is that $L(\theta+\Delta\theta)$ can often be easily evaluated from data obtained along the nominal sample path (θ,ω) alone. One can then also obtain the finite difference $\Delta L(\theta,\Delta\theta,\omega)$ which serves as an estimate of $\Delta J(\theta,\Delta\theta)$. If $\Delta\theta$ is sufficiently small, then one can also estimate a sensitivity of the form $\Delta J/\Delta\theta$ without having to implement any perturbation $\Delta\theta$. This is referred to as *Finite Perturbation Analysis* (FPA) [60].

Often, however, of more interest to sensitivity analysis is the derivative $dJ/d\theta$. Viewing the sample function $L(\theta,\omega)$ as a function of θ over a fixed ω, allows us to define sample derivatives $dL/d\theta$, which are also relatively easy to evaluate along a nominal sample path. A critical question in the field of PA then is whether $dL/d\theta$ may be used as an estimate of $dJ/d\theta$ that possesses desirable properties such as unbiasedness and consistency. The fundamental issue of unbiasedness boils down to the interchangeability of the derivative and expectation, i.e., whether:

$$E\left[\frac{dL(\theta,\omega)}{d\theta}\right]=\frac{dE[L(\theta,\omega)]}{d\theta}. \tag{3.10}$$

One might immediately suspect that this interchange may be prohibited when $L(\theta,\omega)$ exhibits discontinuities in θ. Such discontinuities may arise when a change in θ causes various event order changes. However, a very important result in PA is based on the realization that some event order changes may in fact occur without violating the continuity of $L(\theta,\omega)$. A precise characterization of the condition under which such event order changes are allowed was provided by Glasserman [47] and is known as the *commuting condition*:

Commuting condition: Let $x,y,z_1\in X$ and $\alpha,\beta\in\Sigma_G(x)$ such that $p(z_1;x,\alpha)p(y;z_1,\beta)>0$. Then, there exists $z_2\in X$, such that: $p(z_2;x,\beta)=p(y;z_1,\beta)$ and $p(y;z_2,\alpha)=p(z_1;x,\alpha)$. Moreover, for any $x,z_1,z_2\in X$ such that $p(z_1;x,\alpha)=p(z_2;x,\alpha)>0$, we have: $z_1=z_2$.

In words, the commuting condition requires that if a sequence of events $\{\alpha,\beta\}$ takes state x to state y, then the sequence $\{\beta,\alpha\}$ must also take x to y. Moreover, this must happen in such a way that every transition triggered by α or β in this process occurs with the same probability. The last part of the commuting condition also requires that if an event α takes place at state x, the next state is unique, unless the transition probabilities to distinct states z_1,z_2 are not equal.

Under mild technical conditions, it is then possible to prove that (3.10) indeed holds [47]. In this case, the method for obtaining the sensitivity estimate $dL/d\theta$ along the nominal sample path is referred to as *Infinitesimal*

Perturbation Analysis (IPA). Under some additional conditions, it has also been established that IPA estimates are strongly consistent. Unfortunately, the commuting condition limits IPA to a class of DES and performance sensitivity estimation problems that does not allow for several important DES features such as job prioritization schemes or "blocking" due to finite capacities of resources such as job queues. If, for example, the parameter θ affects the state transition mechanism of the DES, rather than just the event lifetime distributions, then IPA will generally not yield unbiased estimates.

The limitations of IPA have motivated a number of significant extensions beyond the confines of the commuting condition. For a summary of these extensions see [23] and for a more comprehensive presentation see several recent books [47, 57, 24, 22, 98].

Sample Path Constructability. The discussion above was limited to parameters θ that take real values and the related issue of sensitivity estimation. A more general problem is one where we are interested in a *discrete* parameter set $\Theta = \{\theta_1, \ldots, \theta_m\}$. Let some value from this set, θ_1, be fixed. A sample path of a DES depends on θ_1 and on ω, an element of the underlying sample space Ω, which (as in PA) is taken to be $[0,1]^\infty$. Let such a sample path be $\{e_k^1, t_k^1\}$, $k = 1, 2, \ldots$ Then, assuming all events and event times e_k^1, t_k^1, $k = 1, 2, \ldots$, are directly observable, the problem is to construct a sample path $\{e_k^j, t_k^j\}$, $k = 1, 2, \ldots$, for any θ_j, $j = 2, \ldots, m$, as shown in Figure 3.2. We refer to this as the *constructability problem*. Ideally, we would like this construction to take place on line, i.e., while the observed sample path evolves. Moreover, we would like the construction of all $(m-1)$ sample paths for $j = 2, \ldots, m$ to be done concurrently.

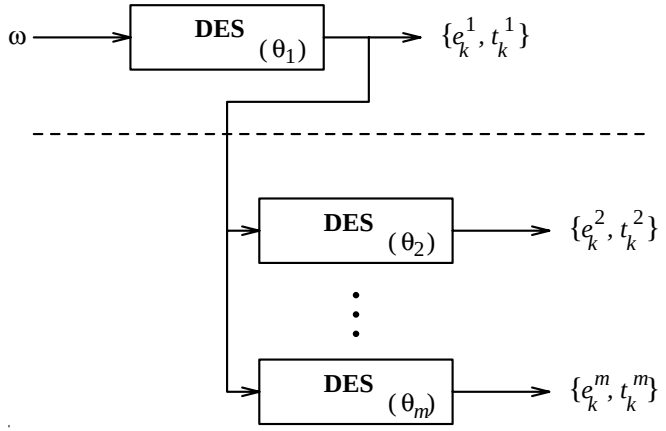

FIGURE 3.2. The sample path constructability problem

A natural question that arises is "under what conditions can one solve

the constructability problem?" This question was addressed by Cassandras and Strickland in [28] and [27], and the constructability condition presented in [27] is briefly reviewed next. First, we say that a sample path $\{e_k^j, t_k^j\}$, $k = 1, 2, \ldots$, is *observable* with respect to $\{e_k, t_k\}$ if $\Sigma_G(x_k^j) \subseteq \Sigma_G(x_k)$ for all $k = 0, 1, \ldots$. Intuitively, this condition guarantees that all feasible events required at state x_k^j to proceed with the sample path construction are "observable", since they are a subset of the feasible events at the observed state x_k. Unfortunately, this condition alone does not guarantee constructability, since the clock values in the constructed sample path are generally not identical to those of the observed sample path, not even in distribution. To address this issue, let v_σ be the lifetime of some event σ currently feasible, and let z_σ be its age, i.e., if y_σ is its clock value, we have $z_\sigma = v_\sigma - y_\sigma$. We can then define the conditional probability distribution of the kth clock value $y_{\sigma,k}$ in a sample path given the event age $z_{\sigma,k}$ (an observable quantity), and denote it by $H(\cdot, z_{\sigma,k})$. Using similar notation for the constructed sample path $\{e_k^j, t_k^j\}$, $k = 1, 2, \ldots$, the constructability condition presented in [27] is as follows.

Constructability condition: A sample path $\{e_k^j, t_k^j\}$, $k = 1, 2, \ldots$, is *constructable* with respect to $\{e_k, t_k\}$ if:

$$\Sigma_G(x_k^j) \subseteq \Sigma_G(x_k) \quad \text{for all } k = 0, 1, \ldots \tag{3.11}$$

$$\text{and} \quad H^j(\cdot, z_{\sigma,k}^j) = H(\cdot, z_{\sigma,k}) \quad \text{for all } \sigma \in \Sigma_G(x_k^j) \tag{3.12}$$

One can then easily establish the following two results:

Theorem: If all event processes are Markovian (generated by Poisson processes), then observability implies constructability.

Theorem: If $\Sigma_G(x_k^j) = \Sigma_G(x_k)$ for all $k = 0, 1, \ldots$, then constructability is satisfied.

The constructability condition (3.11)-(3.12) is naturally satisfied for some simple DES, but, similar to the commuting condition discussed earlier, it limits our ability to analyze most DES of interest. However, to solve the constructability problem when one or both of (3.11)-(3.12) are violated, several techniques have been proposed, which define the state-of-the-art in this particular area. The reader is referred to [110] for more information on the *Standard Clock* approach and to [28, 27] for details on *Augmented System Analysis*. More recently, Cassandras and Panayiotou [26] have proposed a general procedure for the solution of the constructability problem at the expense of additional computational effort for processing nominal sample path data.

Performance Optimization of DES. The ultimate goal of sensitivity

estimation and sample path constructability techniques is to optimize the performance of DES. Returning to the case where θ takes real values and $J(\theta)$ is differentiable, the ability to efficiently estimate sensitivities of the form $\partial J/\partial\theta$ leads to the development of a variety of gradient-based optimization schemes, typically of the form:

$$\theta(n+1) = \theta(n) + \epsilon(n)Y(\theta(n)), \quad n = 0, 1, \cdots \tag{3.13}$$

where $Y(\theta(n))$ is usually an estimate of the gradient of $J(\cdot)$ with respect to $\theta(n)$. The factor $\epsilon(n)$ is referred to as the *step size* or *gain* or *learning rate* parameter. Such schemes are commonly referred to as Stochastic Approximation (SA) algorithms and they have been thoroughly studied in the literature (see [75, 74, 70, 97]). Obviously, the efficacy of such schemes relies on the availability of a gradient estimate $Y(\theta(n))$, which is precisely where PA and similar methods have played a crucial role. Examples of gradient-based optimization algorithms which have been analyzed with gradients estimated through PA techniques may be found in [31, 45]. These algorithms can be used to optimize the performance of complex DES such as communication networks or distributed processing systems by controlling real-valued parameters such as routing probabilities or link-traffic capacities; specific references may be found in [24].

On the other hand, it has become increasingly clear that many of the most interesting parameters in designing and controlling DES are discrete (usually integer-valued) in nature (e.g., buffer capacities, thresholds, integer numbers of resources). This is where the framework of sample path constructability is more appropriate. From an optimization standpoint, if performance estimates $L(\theta_1), \ldots, L(\theta_m)$ over a discrete parameter set $\{\theta_1, \ldots, \theta_m\}$ can be concurrently obtained by observing a single sample path, then we can immediately select a candidate optimal parameter $\theta^* = \arg\min\{L(\theta), \quad \theta \in \Theta\}$. This is potentially the true optimal choice, depending on the statistical accuracy of the estimates $L(\theta_1), \ldots, L(\theta_m)$ of $J(\theta_1), \ldots, J(\theta_m)$. It is important to point out that what is of interest in this process is accuracy in the *order* of the estimates $L(\theta_1), \ldots, L(\theta_m)$, not their actual *cardinal* value, a fact that is exploited in some recent ideas pointing towards what appears to be a promising theory for *Ordinal Optimization* [56]. In general, however, the development of iterative optimization procedures over potentially very large search spaces of the form $\{\theta_1, \ldots, \theta_m\}$ is an area where limited progress has been made to date.

3.4 New Directions in DES Theory

As mentioned earlier, the purpose of this paper is to provide a limited overview of the accomplishments made in the field of DES to date to the extent necessary to bring the reader up to speed with the current state

of the art and to describe some of the new research directions that are emerging. Most of the accomplishments of the past decade have focused on defining fundamental design, control, and optimization problems, and on developing appropriate modeling frameworks and methodologies leading to the solution of these problems. The progress made to date has contributed to the identification of new challenges that lay ahead and set the stage for new approaches required to face these challenges.

Complexity. A key feature of DES is *complexity* and it manifests itself in a variety of ways. For example, the discrete nature of state and event spaces results in an inevitable combinatorial explosion for many basic problems in the analysis of DES, such as state reachability and deadlock avoidance. To make matters worse, most interesting DES are physically large, contributing to this explosion. In addition, the dynamics of these systems are often dependent on human-imposed operational rules that may involve arbitrary conditions, adding to the computational complexity. This is frequently encountered in scheduling or routing decisions ubiquitous in manufacturing systems or communication networks. In short, it is not uncommon in practice to deal with discrete state spaces whose dimensionality exceeds 10^{20} (in chess, the number of board positions is of the order of 10^{43}). Clearly, to tackle problems of such complexity, computational hardware power alone is not sufficient. Thus, one of the goals in the DES research agenda has become the development of analytical devices aimed at tackling this problem.

Three techniques that are being used to tackle complexity are: aggregation, limited lookahead, and symbolic representations. Aggregation techniques constitute one means of reducing the dimensionality of the state spaces involved in DES; they are often used in the context of hierarchical models. In this regard, we mention the control of DES modeled by hierarchical state machines studied in [17] and hierarchical logic control, in the COCOLOG framework, developed in [21, 114]. The strategy in limited lookahead supervisory control is to calculate control actions "on the fly" (i.e., on-line) based on a limited projection into the future of the system behavior [33, 34, 9, 52]. The use of symbolic methods, together with efficient data structures such as "binary decision diagrams", tackles complexity by representing the state transition function of a DES symbolically (i.e., as logical formulas) rather than by explicit enumeration. Symbolic methods have been applied to model checking in computer science (cf. [19]) and have been used for the synthesis of supervisors in several works, including [5, 50, 80].

One opportunity for tackling complexity arises from the observation that many DES naturally consist of distinct components, which suggests the use of *decentralized* control and optimization. For instance, a manufacturing system is naturally decomposed into workcenters or cells, while a communication network consists of switches or nodes. This implicit "modularity"

can be exploited to develop effective decentralized methodologies, which not only simplify many control and optimization problems, but also present advantages in terms of issues such as system reliability and robustness. We mention that many works have dealt with decentralized schemes in supervisory control, among them [79, 35, 99]. Recently, researchers have been looking at "distributed" schemes, i.e., decentralized schemes where some communication is allowed between supervisors located at different "sites"; cf. [10, 115]. In this paper, we will discuss decentralization in section 3.5 in the context of stochastic models of DES.

Uncertainty. Another major challenge in the study of DES comes from the presence of *uncertainty* (which may, in fact, be viewed as one form of "complexity"). Uncertainty may manifest itself as the inability to predict the next state entered as a result of an event, in which case models involving *nondeterministic* features need to be used. Section 3.7 is devoted to supervisory control of nondeterministic systems with this kind of uncertainty.

Uncertainty also arises in the timing of event occurrences and hence the time spent at a particular state, which can drastically affect the performance of a DES. In such cases, the use of *stochastic* models becomes necessary, as in describing the demand for a product or message traffic in a communication network. Dealing with uncertainty is yet another major challenge for the future of DES. While good modeling frameworks have been developed to describe different forms of uncertainty, technological advances require increasingly higher performance requirements that translate into increasingly lower tolerance of uncertainty. For example, demand for higher quality in manufacturing is intrinsically connected to the requirement for better control of variability in different processes. The same is true for the transmission of packetized voice and video in communication networks, where one often talks of packet "loss probabilities" of the order of 10^{-9} or less, implying the need to analyze the effect of "rare events" in a DES.

An additional form of uncertainty is the result of frequent changes in the system itself. Such changes may be due to random equipment failures, uncontrollable changes in system parameters (e.g., traffic rates varying with time), or structural changes that are a natural part of new system designs (e.g., the addition and removal of nodes over time in a wireless network). This creates an urgent need for at least two types of mechanisms for combating uncertainty:

1. *Mechanisms for failure diagnosis*, i.e., determining if a given system is in a failed state or if some prior failure event (which was not observable) has happened. This problem will be discussed in section 3.6.

2. *Mechanisms for adaptive control and optimization.* Coupled with the

complexity problem already mentioned, it is becoming increasingly apparent that different forms of *learning* approaches will have to be integrated with existing DES analysis methodologies. In this paper, we will limit ourselves to the discussion of some recent developments along these lines in section 3.5.

Hybrid Systems. The past few years have seen the emergence of a class of dynamic systems in which a time-driven and an event-driven component are closely connected, giving rise to the term *hybrid system*. Typically, one views a conventional plant as the time-driven component, with an event-driven supervisory control structure present (representing the "control computer"). The plant is equipped with the ability to generate events which are input to the supervisor; this in turn supplies control events back to the plant. This, however, may be an overly narrow viewpoint as far as the hybrid nature of systems of interest is concerned. In a manufacturing process, for instance, a typical production part at any point in time is characterized by two kinds of information: its *physical state*, which describes the extent to which the part is close to a finished product (i.e., the target desired state); and its *temporal state*, which describes how long the part has been in the system. Both states are subject to time and event-driven dynamics which may in fact be closely coupled. Moreover, the controller itself may be either time or event-driven.

Given the increasing importance of these hybrid systems, the need to consolidate existing models or develop new ones is self-evident. The obvious next step is to develop analytical tools for design, control, and optimization. In this paper, we will limit ourselves to some control issues motivated by manufacturing problems, as discussed in section 3.8.

Optimization and Optimal Control. As in the study of other classes of dynamic systems, the phase of modeling and addressing fundamental design, control synthesis, and performance evaluation problems is normally followed by a desire to optimize system performance.

In the case of untimed models, a theory of optimal control of DES has recently been proposed in [106]; this theory has conceptual similarities with "classical optimal control" in the sense that it captures, in a dynamic setting, the fundamental trade-off between costs on the system behavior and costs on control.

In the case of timed models, one can draw upon the classical theory of optimal control with state dynamics represented through max-plus equations such as those we saw in section 3.2. The difficulty here is dealing with the discontinuous nature of these equations due to the max (or min) operator. Nonetheless, some first steps along these lines may be found in [46, 91].

In the case of stochastic models of DES, most problems considered to date have been limited to parametric optimization as discussed in section 3.3.

However, the enormous complexity we encounter calls for radically new approaches needed to overcome the combinatorial explosion in many common problems, the lack of structure which often prohibits the use of "special purpose" devices for the efficient exploration of search spaces, and the presence of uncertainties which requires the use of stochastic optimization methods. A variety of new, often "exotic", ideas have emerged in recent years opening up some promising new directions in this area (for a brief survey, see [56]) and a few success stories are beginning to be reported. In the area of stochastic *dynamic* optimization, i.e., the ability to process information in real time for the purpose of optimization, little progress has been made. An optimal control theory that parallels the successes of the Linear Quadratic Gaussian (LQG) framework for classical time-driven systems is still lacking. In this paper, we will present some new ideas aiming at this goal in the context of hybrid systems in section 3.8.

3.5 Decentralized Control and Optimization

As argued in the previous section, the tremendous complexity of DES poses serious computational obstacles to even the most elegant and efficient analytical techniques developed to date for design, control, and optimization purposes. These computational obstacles are not likely to be overcome through mere computing power, especially when it comes to applications with real-time requirements. It is therefore reasonable to seek approaches allowing us to decompose a DES into more manageable parts. Fortunately, by their very nature, many DES do tend to consist of a number of distributed components, with each component operating autonomously and contributing to the overall function of the system. Examples include the switches of a communication network, the processors in a distributed computer system, or workstations in a manufacturing system. While this decomposition conceptually provides opportunities for efficient control and optimization of the system, the price to pay is the need for careful coordination and transfer of information among components. Thus, a desirable objective for ongoing and future research is the development of decentralized schemes which permit individual components to take control actions that contribute towards systemwide performance targets and design specifications.

In this section, we will discuss one approach for achieving decentralized control of DES in a stochastic environment, the objective being to optimize a systemwide performance criterion. Let u denote a real-valued controllable parameter vector and $J(u)$ a given performance measure (or cost) to be optimized. The DES under consideration consists of K components. Thus, the parameter vector is of the form $u = [u_1, \cdots, u_K]$, where u_i corresponds to the ith component, $i = 1, \cdots, K$, and may itself be a vector. Our objec-

tive is to determine a vector u^* that maximizes the performance criterion $J(u)$. When the DES operates in a stochastic environment, this criterion is usually of the form $J(u) = E[L(u)]$, where $L(u)$ is the cost obtained over a specific sample path. This problem is particularly hard due to the fact that closed-form expressions for $J(u)$ are seldom available for complex DES. As a result, one must resort to various techniques for estimating $J(u)$ over all (or as many as possible) values of u in order to seek u^*.

3.5.1 Some Key Issues

For control purposes, the most common approach for determining u^* is based on iterative schemes of the general form (3.13), i.e.,

$$u(n+1) = u(n) + \epsilon(n)Y(u(n)), \quad n = 0, 1, \cdots \tag{3.14}$$

where, due to the decomposition of the DES we are considering, $Y(u(n)) = [Y_1(u(n)), \cdots, Y_K(u(n))]$ is an estimate of the gradient of $J(\cdot)$ (or its negative) with respect to $u(n)$. As pointed out in section 3.3, such Stochastic Approximation (SA) schemes have been thoroughly studied in the literature; however, less attention has been paid to the use of SA algorithms for systems consisting of many components [76, 111]. When this is the case, there are a number of issues, some not so obvious, related to the simple scheme (3.14) above.

1. Gradient Estimation. Usually, $Y(u(n))$ is an estimate of the gradient of $J(u(n))$ (or its negative) with respect to $u(n)$. Thus, the first issue to consider is that of determining appropriate gradient estimates based on observable system data. This is one aspect of DES theory where much progress has been made over the past decade, with a substantial arsenal of efficient and well-understood gradient estimation techniques such as Perturbation Analysis (PA) (e.g., [57, 47]) and the Likelihood Ratio (LR) methodology (e.g., [98]. However, the efficacy of these techniques is usually limited to "small" systems, which is why decomposing a DES into such "small" components can take advantage of the techniques.

2. Convergence. Under a number of conditions on the set of admissible control parameter vectors $u(n)$, the step size sequence $\{\epsilon(n)\}$, and the estimates $Y(u(n))$, convergence w.p. 1 of the sequence $\{u(n)\}$ to a global optimum u^* can be established for the basic SA scheme (3.14). In addition, the "weak convergence" framework of [74] can also be used, as in [76]. However, when using (3.14) for decentralized optimization, the issue of convergence becomes significantly more complicated.

3. Adaptivity. Convergence to a global optimum u^* is normally established for (3.14) by allowing the step size sequence $\{\epsilon(n)\}$ to go to zero over time. If, however, (3.14) is used on line as an adaptive control mechanism, then the scheme can obviously not respond to changes in the operating environment after the step size has reached zero. We are, therefore,

often interested in the limiting behavior of SA schemes with some constant (normally small) step size, which would permit the control vector to track various changes on line, usually at the expense of some oscillatory behavior.

4. Distributed Estimation. In many DES, such as large communication networks, it is infeasible to transfer instantaneous state information from the ith system component to other components or to a central controller. Thus, it is highly desirable to develop *distributed* algorithms, whereby at least part of the necessary computation is carried out locally at each component. In the SA scheme (3.14), the main computational burden involves the gradient estimation process. One of our objectives, therefore, is to have each component locally evaluate an estimate of the derivative of $J(u)$ with respect to the local control parameter u_i.

5. Decentralized Control. Once the gradient estimates are evaluated, the simplest approach for executing an update in (3.14) is to have a central controller who collects all estimates and performs control updates. This approach, however, requires significant coordination among components, as well as the transfer of state information; this involves substantial communication overhead and delays which often render state information useless. More importantly, failure of a central controller implies failure of the entire system, which cannot sustain its proper operation without it. Therefore, a desirable alternative is to allow each individual component to separately update the global control vector $u(n)$ and transfer this information to all other components.

6. Synchronization. In a fully synchronized scheme, there is an a priori mechanism based on which the updates of $u(n)$ take place. For instance, a central controller periodically requests estimates from all components in order to perform a control update. If the procedure is decentralized, however, a natural question is whether any component can be allowed to take a control action at *any random point in time* without any synchronizing mechanism. Such a feature is obviously highly desirable, since it requires virtually no coordination among components and it minimizes the amount of information that is transferred from one component to another.

7. Full utilization of system state history. A problem that frequently arises in SA schemes is that the estimator $Y(u(n))$ may not use all data collected over the history of the process. This typically arises in an asynchronous control update scheme, when a component being informed of a control update from another component may have to discard a partial local computation that it is in the process of performing. It is, therefore, desirable to develop a scheme using as much of the complete system history as possible and avoid having to re-initialize estimators, which essentially discards past history information.

In the remainder of this section we will review a class of problems for which a fully decentralized asynchronous optimization scheme based on distributed gradient estimation can be shown to converge to a global optimum in the framework of [74]. This decentralized scheme, proposed in [111], has

the added property of making use of *all* past state information. It is interesting, if not somewhat surprising, that such a scheme indeed converges, despite this very loose coordination among system components, opening up the possibility for similar control approaches applied to broader classes of problems.

3.5.2 Decentralized Optimization Problem Formulation

The optimization problem we are interested in is the determination of a vector u^* that maximizes a performance criterion $J(u) = E[L(u)]$, where $L(u)$ is the sample performance function. The DES is assumed to be too complex for any analytical expression for $J(u)$ or sufficient approximation to be available, and we resort to an optimization scheme of the general form (3.14), where $Y(u(n))$ is an estimate of the negative of the gradient of $J(u)$ with respect to $u(n)$. Moreover, we assume that each system component only has access to local state information and can estimate a local performance criterion $J_i(u_i)$ and its gradient. For ease of notation, let us limit ourselves here to the case where u_i is a scalar. Given the structure of the system, the optimization problem we face is as follows:

$$\max_{u \in U} \sum_{i=1}^{K} \beta_i J_i(u_i) \qquad s.t. \;\; g_1(u_1, \cdots, u_K) = c_1, \ldots, g_r(u_1, \cdots, u_K) = c_r$$

where β_i, $i = 1, \cdots, K$, are weights associated with the system components and $g_j(u_1, \cdots, u_K) = c_j$, $j = 1, \cdots, r$ are r *linear* constraints. Note that there may also be additional inequality constraints associated with the problem above; these can be taken into account by appropriately defining the admissible set U. Finally, let us assume that each system component has knowledge of all weights β_i and all linear constraints present. An unconstrained version of this optimization problem can be obtained by solving the r linear equations above so at to eliminate some of the K control parameters and solve for $q < K$ of them. Let C_q denote the reduced set of q system components. For any component $k \notin C_q$, we then have

$$u_k = a_k + \sum_{j \in C_q} b_{kj} u_j \tag{3.15}$$

for some constant coefficients a_k and b_{kj}, $j \in C_q$. It is then straightforward to show that the optimization scheme for all remaining $i \in C_q$ reduces to

$$u_i(n+1) = u_i(n) + \epsilon(n) \sum_{k=1}^{K} \gamma_{ki} D_k(n) \qquad \text{for all } i \in C_q \tag{3.16}$$

where $D_k(n)$ is an estimate of the negative of the derivative dJ_k/du_k available at the end of the nth iteration and the coefficients γ_{ki} are given by

$$\gamma_{ki} = \begin{cases} \beta_i & k = i \\ \beta_k b_{ki} & k \notin C_q \\ 0 & \text{otherwise} \end{cases} \tag{3.17}$$

We will model the DES we consider as a stochastic timed automaton (see section 3.2). Since our DES consists of K distributed components, the event set Σ_G is partitioned into $K+1$ subsets $\Sigma_G^0, \Sigma_G^1, \cdots, \Sigma_G^K$ so that Σ_G^0 contains all events (if any) directly observable by all system components, and Σ_G^k contains events which are directly observable by the kth component alone. Under a control parameter vector u, let $X_m(u)$ denote the state entered after the mth event occurrence; thus, we obtain a Markov chain with transition probability $P_u(x,\cdot)$ defined on $\mathcal{B}$, the σ-algebra of subsets of the state space:

$$P_u(x,B) = P_u\{X_{m+1}(u) \in B | X_m(u) = x\}, \quad \forall B \in \mathcal{B}$$

We shall denote by E_u the expectation with respect to the measure P_u of the Markov chain $\{X_m(u)\}$. We assume that for each value of u, where u is defined over a set U, the invariant measure $\mu_u(\cdot)$ of the corresponding process exists and is unique. In this setting, we focus on the problem of finding a control value $u^* \in U$ that maximizes $J(u) = \int L(x,u)\mu_u(dx)$. Within the framework of gradient based methods, we shall assume that $J(u)$ is differentiable and that all $\frac{\partial J(u)}{\partial u_k}, k = 1, \ldots, K$ are bounded and continuous.

Example: To illustrate our modeling framework, consider an optimal scheduling problem where K nodes compete for a single server/resource. This is motivated by the well-known "transmission scheduling problem" arising in packet radio networks, where the resource is the communication channel, fixed length packets arrive at node k according to an arbitrary interarrival time distribution with rate λ_i, and a slotted time model is considered (with slot size equal to the packet transmission time δ). At each scheduling epoch, i.e., at the start of each time slot, the channel is assigned to a particular node (see Figure 3.3). The assignment is based on a *random polling* policy: the current time slot is allocated to the kth class with probability u_k. The objective is to determine the optimal slot assignment probabilities so as to minimize the weighted average packet waiting time. The constrained optimization problem is then stated as:

$$\min_u \frac{1}{\sum_{j=1}^K \lambda_j} \sum_{k=1}^K \lambda_k J_k(u_k) \quad s.t. \quad \sum_{i=1}^K u_k = 1 \qquad \textbf{(P1)}$$

where $J_k(\cdot)$ is the average node k packet waiting time and $u = [u_1, \cdots, u_K]$ is assumed to be in the set of probability vectors such that $u_k > \lambda_k/\delta$,

which ensures stability of each queue. This defines the set U over which the control vector u is defined. In the absence of any a priori information on the arrival processes, closed form expressions for this performance measure are unavailable. Thus, the gradual on-line adjustment of u is one attractive approach.

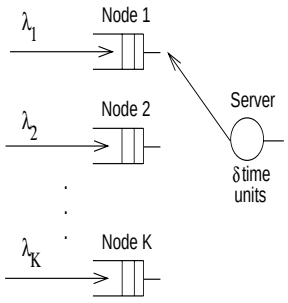

FIGURE 3.3. The transmission scheduling problem

Let a_k denote a packet arrival event at node k and τ_k a packet transmission event when node k is assigned a time slot. The event set of this DES is then $\Sigma_G = \{a_1, \cdots, a_K, \tau_1, \cdots, \tau_K\}$. Note that events a_k, τ_k are observed only by node k. The only way that a node $j \neq k$ can become aware of these events is if k explicitly transmits such information to j; this, however, not only entails communication overhead, but the information reaching j is also delayed. A natural partition of Σ_G consists of K sets, $\Sigma_G^1, \cdots, \Sigma_G^K$ with $\Sigma_G^k = \{a_k, \tau_k\}$.

In what follows, we shall use the notation $\mathbf{u}_m^\epsilon$ to denote the vector-valued control parameter in effect at the epoch of event m and ϵ is the value of the step size parameter in (3.16). Let us next introduce the two main time scales we shall use and associated notation. We have a "fast" global time scale, defined by all events that drive the DES, and a "slow" time scale, defined by instants in time when control updates are performed according to (3.16). We will use the following notation to distinguish these two time scales: m = global event index, and n = iteration index over global control updates. Thus, $\mathbf{u}_m^\epsilon$ is updated at selected events (at which time the index n is incremented) according to the decentralized control structure to be described later. As we will see, the global control updates are performed asynchronously by individual system components and they coincide with the instants when any component completes a local estimation interval.

3.5.3 Distributed Estimation

To exploit the modularity of a DES consisting of K components, the derivative estimators required in the control update mechanism (3.16) are ob-

tained in *distributed* fashion, i.e., each system component separately performs all estimation computation required, using only locally available state information. We emphasize that the issue of distributed estimation is distinct from that of control implementation, which can be centralized or decentralized.

Let $j = 1, 2, \ldots$ index the sequence of *local* events at component k, i.e., all events in the set Σ_G^k. Let $m_k(j)$ be the corresponding *global* event index (when no ambiguity arises, we will also use $m(j)$). We define $\rho_k(u)$ to be the invariant average rate of events at k under some fixed u. By the ergodicity assumption:

$$\rho_k(u) = \lim_{j \to \infty} \frac{j}{m_k(j)} \tag{3.18}$$

A derivative estimator of the objective function $J(u)$ with respect to the control parameter u_k at component k is calculated from local observations of the state values over a period of time. Let j be some local event index and Δ a number of local events. Then, we define $d_k(j, \Delta)/\Delta$ to be the estimator of dJ_k/du_k obtained for the fixed-u process over the Δ local events $\{j, \ldots, j + \Delta - 1\}$. We can then see that all system components can evaluate their estimators by dividing the computation into *estimation intervals*. We construct these intervals by choosing an appropriate increasing sequence $L_k(l)$, $l = 1, 2, \ldots$, of random stopping times with independently distributed increments:

$$\Delta_k(l+1) = L_k(l+1) - L_k(l)$$

for each component k. Thus, the l-th estimation interval at component k contains all local events $j \in \{L_k(l), \ldots, L_k(l+1) - 1\}$. The resulting lth estimator at component k is

$$\tilde{d}_k(l) \equiv d_k(L_k(l), \Delta_k(l+1)) \tag{3.19}$$

In other words, we view the time line associated with component k as being partitioned into intervals defined by the sequence $\{L_k(l)\}$, the lth interval containing $\Delta_k(l+1)$ local events. Hence, a sequence of estimates $\{\tilde{d}_k(l)\}$ is defined. Before proceeding, it is important to note that even if the control values change within an estimation interval, the local estimators use the same functional form and continue the computation of the current estimate.

Example (continued): Returning to the system of Figure 3.3, a convenient way of defining an estimation interval at any node k is to consider one local busy period. Then, an estimator $\tilde{d}_k(l)$ based on PA (discussed in section 3.3) has the following general form:

$$\tilde{d}_k(l) = \frac{1}{u_k} \sum_{j \in A_l} f_k(X_{m(j)}, u_k) \tag{3.20}$$

where A_l is a particular set of local event indices and $f_k(X_{m(j)}, u_k)$ is a function of the of the state when the jth local event occurs (equivalently: the $m(j)$th global event occurs) and of the control parameter u_k. We will not discuss here the precise nature of A_l or of f_k (which may be found in [25, 112]), but only point out that f_k depends on the state through those entries associated with the kth component.

The Decentralized Asynchronous Control Structure. We shall now present a fully decentralized and asynchronous control scheme where each component imposes a *global* control update on the entire system at the end of its individual local estimation intervals. Because of the global and local time scales involved, it is important to carefully define the event indexing notation we will use:

$$\begin{aligned} L_k(l) &= \text{local event index at } k \text{ when the } l\text{th estimation interval ends} \\ G_k(l) &= \text{global event index corresponding to } L_k(l), \text{ i.e., } G_k(l) = m(L_k(l)) \\ G(n) &= \text{global event index indicating the } n\text{th global control update} \end{aligned}$$

The decentralized structure we will use is described as follows. Each system component k of the DES becomes a *global* controller and can change the value of the local variable u_k, as well as the values of all u_i $i \neq k$ (as long as the constraints in (3.15) hold at all times). In particular, at the end of the lth estimation interval, when the global event index is $G_k(l) = m+1$, k becomes a controller and changes the value of the ith component of the vector $\mathbf{u}_m^\epsilon$ by adding to it an amount dependent on $\tilde{d}_k(l)$ as follows:

$$\mathbf{u}_{m+1,i}^\epsilon = \mathbf{u}_{m,i}^\epsilon + \epsilon Y_{m,i}^\epsilon(\mathbf{u}_m^\epsilon), \qquad i \in C_q \tag{3.21}$$

where:

$$Y_{m,i}^\epsilon(\mathbf{u}_m^\epsilon) = -\sum_{k=1}^{K} \frac{\gamma_{ki}}{\rho_k(\mathbf{u}_m^\epsilon)} \sum_{l=1}^{\infty} \tilde{d}_k(l) \mathbf{1}_{\{G_k(l)=m+1\}} \tag{3.22}$$

Assuming that $1 \leq \Delta_k(l)$ a.s, it follows that, for any fixed component k, for every m at most one value of l is such that $G_k(l) = m+1$, that is, $\sum_{l=1}^{\infty} \mathbf{1}_{\{G_k(l)=m+1\}} = 0$ or 1 a.s. Thus, whenever the $(m+1)$th global event coincides with the end of an update interval (say, l) at some component (say, k), the expression above yields $\gamma_{ki}\tilde{d}_k(l)/\rho_k(\mathbf{u}_m^\epsilon)$. This is the amount by which the ith control parameter is changed at that time. Notice that in this scheme two or more controllers may simultaneously update the same components of the control vector.

This asynchronous decentralized control scheme can be summarized as follows:

Each Component k:

1. Evaluates a *local* estimator $\tilde{d}_k(l)$ over an interval of local events $j \in \{L_k(l), \ldots, L_k(l+1)-1\}$, $l = 0, 1, \ldots$
2. At epochs of *local* events $L_k(l)$ [equivalently, global events $m = G_k(l)$], $l = 1, 2, \ldots$, updates all control parameters by evaluating $\mathbf{u}^\epsilon_{m+1,i}$ for all $i \in C_q$ through (3.21)-(3.22); and for all $i \notin C_q$ through (3.15).
3. Sends the complete updated control vector $\mathbf{u}^\epsilon_{m+1}$ to all other system components.

An alternative and more convenient way to rewrite (3.21)-(3.22) is obtained by introducing auxiliary variables, denoted by $v^\epsilon_{ki}(n)$, with the following interpretation: $v^\epsilon_{ki}(n)$ is the cumulative control change imposed by component k on component i by the instant when k becomes a global controller for the nth time. In other words, at the epoch of a control update event $G_k(n)$, the current value of u_i has experienced a total change from the action of controller k given by $v^\epsilon_{ki}(n)$. The dynamics of these auxiliary variables are specified as follows. Let $v^\epsilon_{ki}(0)$ be such that $\sum_k v^\epsilon_{ki}(0) = u^\epsilon_i(0) = u_i$. Then define:

$$v^\epsilon_{ki}(n+1) = v^\epsilon_{ki}(n) + \epsilon Y^\epsilon_{ki}(n) \tag{3.23}$$

where

$$Y^\epsilon_{ki}(n) = -\frac{\gamma_{ki}}{\rho_k(\mathbf{u}^\epsilon_{G_k(n+1)-1})}\tilde{d}_k(n) \tag{3.24}$$

It should be clear that $Y^\epsilon_{ki}(n)$ is the amount by which component k imposes a change on the control parameter value at i at the $(n+1)$th time that k becomes a global controller (i.e., at global event index $G_k(n+1) = m(L_k(n+1))$).

3.5.4 Weak Convergence Analysis

In this section, we address the issue of convergence of the scheme (3.16). To do so, we have to carefully consider how varying the control parameter vector u after selected events affects the underlying system state and hence the derivative estimates which drive these SA schemes. The first step in this process is to enlarge the original state vector X_m by defining an appropriate "augmented" state, denoted by ξ^ϵ_m. The choice of ξ^ϵ_m must be such that the resulting process $(\xi^\epsilon_m, \mathbf{u}^\epsilon_m)$ is a Markov Decision Process (MDP). While this may not always be possible for arbitrary systems and derivative estimators, the structure of our particular DES derivative estimators allows us to accomplish this goal with relatively little effort (see [111]).

The Interpolation Processes. When dealing with the notion of convergence, we implicitly assume a concept of a norm. The approaches that study a.s. convergence of the sequence $\{\mathbf{u}^\epsilon_m\}$ generally use the norm in $\mathbb{R}^K$.

The approach taken here is the study of the behavior of the updates by taking a global view of the processes and establishing that it gets closer to the trajectory of the solution of an ordinary differential equation (ODE), as $\epsilon \to 0$. The limiting process shall therefore be a *continuous time* process. The first step in analyzing weak convergence of SA schemes is to define "continuous time" processes from the event-driven sequence of control values. We shall therefore begin by defining two important continuous time processes, as follows.

Let us start by considering $\{S^\epsilon(n), n \geq 0\}$ to be a sequence of random stopping event indices, measurable with respect to the filtration $\{\mathcal{F}_m^\epsilon\}$ of a MDP process $(\xi_m^\epsilon, \mathbf{u}_m^\epsilon)$. Then, set

$$\Delta^\epsilon(n) = S^\epsilon(n+1) - S^\epsilon(n)$$

In addition, let $(\chi^\epsilon(n), w^\epsilon(n)) = (\xi^\epsilon_{S^\epsilon(n)}, \mathbf{u}^\epsilon_{S^\epsilon(n)})$ be a random sampling of the state of the process.

We now define the *ladder interpolation process* associated with $w^\epsilon(n) = \mathbf{u}^\epsilon_{S^\epsilon(n)}$:

$$\zeta^\epsilon(t) = w^\epsilon(n) \quad \text{for } t \in [n\epsilon, (n+1)\epsilon) \tag{3.25}$$

and the *natural interpolation process*:

$$\tilde{\zeta}^\epsilon(t) = \mathbf{u}_m^\epsilon \quad \text{for } t \in [m\epsilon, (m+1)\epsilon) \tag{3.26}$$

The first interpolation scales time with respect to *control update* intervals, and the second with respect to *global event* epochs. We begin with the piecewise constant process describing control updates as a function of the global event index m. This defines the natural interpolation process $\tilde{\zeta}^\epsilon(t)$. This process is then sampled at a subset of event indices $\{S(0), S(1), \ldots\}$ with corresponding values $w^\epsilon(0), w^\epsilon(1), \ldots$. The ladder interpolation process $\zeta^\epsilon(t)$ is finally simply obtained by redrawing this piecewise constant function as a function of the control update index n on a $n\epsilon$ scale.

These interpolation processes possess a number of important properties when $\{S^\epsilon(n)\}$ is related to the control update sequences corresponding to $\{G_k(l)\}$ for the fully decentralized structure presented earlier. We shall omit these properties (however, see [111]) and limit ourselves to a statement of the main result that establishes the fact that the decentralized and asynchronous (3.21)-(3.22) weakly converges to a solution of our optimization problem.

Main Convergence Result. Consider the control update scheme (3.21) and for each k, the associated update scheme (3.23) for the vector $v_k^\epsilon(n)$ (with components $v_{ki}^\epsilon(n)$, $i = 1, \ldots, K$.) Recall that the values of $v_k^\epsilon(n)$ are updated only at the local update epochs at component k corresponding to global event indices $G_k^\epsilon(n)$. For any fixed k, let $\nu_k^\epsilon(t)$ and $\tilde{\nu}_k^\epsilon(t)$ be the

ladder and natural interpolation processes related to $v_k^\epsilon(n)$, as follows:

$$\begin{aligned} \nu_k^\epsilon(t) &= v_k^\epsilon(n) \quad \text{for } t \in [n\epsilon, (n+1)\epsilon) \\ \tilde{\nu}_k^\epsilon(t) &= v_k^\epsilon(n) \quad \text{for } t \in [G_k(n)\epsilon, G_k(n+1)\epsilon) \end{aligned}$$

where $\nu_k^\epsilon(\cdot)$ is a vector with components $\nu_{ki}^\epsilon(\cdot)$, and similarly for $\tilde{\nu}_k^\epsilon(\cdot)$. Note that the natural interpolation process $\zeta^\epsilon(\cdot)$ defined in (3.26) is given by

$$\tilde{\zeta}^\epsilon(t) = \sum_{k \in C_q} \tilde{\nu}_k^\epsilon(t)$$

for all t, ϵ, since $\tilde{\nu}_k^\epsilon(\cdot)$ are piecewise constant and only change at the epochs corresponding to local updates, so that the actual control value at the epoch of event m is the initial control value $\mathbf{u}_0$ plus the total changes effected at the control updates. Moreover, $\tilde{\nu}_k(\cdot) - \tilde{\nu}_k(0)$ contains the cumulative changes performed at component k, and $\tilde{\nu}_k(0) = v_k^\epsilon(0)$ with $\sum_k v_k^\epsilon(0) = \mathbf{u}_0$.

Under several technical conditions (for details, including the case of constraints on the control vector, see [111]), it is possible to show that the processes $\{\tilde{\zeta}^\epsilon(t)\}$ converge weakly as $\epsilon \to 0$ to a solution of the ODE:

$$\frac{d\tilde{\zeta}(t)}{dt} = \nabla_u J(\tilde{\zeta}(t)) \tag{3.27}$$

If this ODE has a unique solution for each initial condition, then the sequence $\tilde{\zeta}^\epsilon(\cdot)$ converges weakly to $\tilde{\zeta}(\cdot)$. Furthermore, if (3.27) has a unique stable point u^* such that $\nabla_u J(u^*) = 0$, then $\lim_{t\to\infty} \tilde{\zeta}(t) = u^*$.

Example (continued): Let us return to the system of Figure 3.3 and problem (**P1**) with $K = 3$ and $\lambda_1 = \lambda_2 = \lambda_3$. Because of this symmetry, it is obvious that the optimal control vector is $u^* = [1/3, 1/3, 1/3]$, allowing us to numerically verify the performance of the decentralized control scheme implemented. Following the scheme (3.21)-(3.22), each node k asynchronously performs a control update at the end of its local estimation interval $[L_k(l), L_k(l+1))$ $l = 0, 1, ...$, where, for simplicity, the interval length may be chosen to be a deterministic number of service completions at node k. Let m be the global index and let node k be the node that initiates its l-th update at event $m+1 = G_k(l)$. Then, node k updates the ith component of the control vector according to (3.21) as follows:

$$\begin{aligned} \mathbf{u}_{m+1,1} &= \mathbf{u}_{m,1} + \epsilon \frac{\gamma_{k1} \tilde{d}_k(l)}{\rho_k} \\ \mathbf{u}_{m+1,2} &= \mathbf{u}_{m,2} + \epsilon \frac{\gamma_{k2} \tilde{d}_k(l)}{\rho_k} \end{aligned}$$

and $\mathbf{u}_{m+1,3} = 1 - \mathbf{u}_{m+1,1} - \mathbf{u}_{m+1,2}$, where $\gamma_{11} = 1 = \gamma_{22}, \gamma_{12} = \gamma_{21} = 0, \gamma_{31} = \gamma_{32} = -1$ and $\tilde{d}_k(l)$ is the estimate at node k over the local interval

$[L_k(l), L_k(l+1))$. Finally, the complete updated control vector $\mathbf{u}_{m+1}$ is sent to all other system components $j \neq k$. Note that ϵ is kept fixed in this example. The procedure therefore updates as follows: every time node 1 (or node 2) has an estimate $\tilde{d}_1(l)$ (or $\tilde{d}_2(l)$), it adds to u_1 (or u_2) the corresponding term weighted by the factor ρ_k and adjusts u_3. When node 3 has an estimate $\tilde{d}_3(l)$, it subtracts it from both u_1 and u_2, and adjusts u_3. The compound effects yield convergence of the natural interpolation $\tilde{\zeta}^\epsilon(t)$ to the solution of the ODE:

$$\frac{du_1(t)}{dt} = \frac{dJ_1(u_1(t))}{du_1} - \frac{dJ_3(u_3(t))}{du_3} \tag{3.28}$$

$$\frac{du_2(t)}{dt} = \frac{dJ_2(u_2(t))}{du_2} - \frac{dJ_3(u_3(t))}{du_3} \tag{3.29}$$

$$u_3(t) = 1 - u_1(t) - u_2(t) \tag{3.30}$$

which, in the limit as $t \to \infty$, has an asymptotic value $u(t) \to u^*$ that satisfies the Kuhn-Tucker conditions for optimality. Explicit numerical results for this example may be found in [111].

3.6 Failure Diagnosis

In this section, we consider the problem of failure diagnosis for DES that are modeled using automata. Our goal is to survey some of the recent work in this area and then present one approach in some more detail using a simple example. Finally, we will discuss some trends in this area of research.

3.6.1 Statement of the Problem

Failure diagnosis is an important and widely researched problem in system and control engineering (cf. [93]). Many different approaches have been proposed and developed to deal with failure diagnosis; among them, we mention fault trees, methods based on analytical redundancy, expert systems, model-based reasoning methods in Artificial Intelligence, and more recently approaches based on DES theory.

From a DES viewpoint, the problem of failure diagnosis is to determine if the system is in a failed state or if some (unobservable) failure event has happened based on the available observations of the system behavior and using model-based inferencing; thus, failure diagnosis is an issue of *analysis* of the behavior of a dynamic system.

DES approaches to failure diagnosis are most appropriate for diagnosing abrupt yet non-catastrophic failures, i.e., failures that cause a distinct change in the behavior of the system but do not necessarily bring it to a

halt. Such sharp failures occur in a wide variety of technological systems including automated manufacturing systems, communication networks, heating, ventilation, and air-conditioning units, process control, and power systems; examples of failures are equipment failures (e.g., stuck failures of valves, stalling of actuators, bias failures of sensors, and controller failures) as well as many types of process failures (e.g., overflow of buffers in manufacturing and communication networks, contamination in semiconductor manufacturing, and control software faults). "Finer" types of failures (e.g., slow drifting of sensors and gradual fouling of reheat coil in heating system) are typically handled by continuous-variable methods based on analytical redundancy.

3.6.2 Survey of Recent Literature

In the last several years, many different approaches to the problem of failure diagnosis of DES have appeared in the literature. Each approach was developed to address failure diagnosis in a specific application area, and this explains why the approaches differ in many respects. Nevertheless, the approaches that we discuss below, being model-based, are generic and thus suitable for many application areas.

We will mostly focus on works that use automata as the modeling formalism for the DES at hand. We note that diagnostic techniques using different modeling formalisms have been proposed; for instance, Petri net models are used in [113, 12].

Automotive applications, in particular mixed digital and analog circuits and exhaust gas recirculation, motivated the work in [78, 77] where a state-based approach to off-line and on-line diagnosis is proposed. The state space of the system is partitioned into normal and failed states, and the objective is to identify which element of the partition the state is in based on measurements of the system outputs (where outputs are associated with states). Two scenarios are considered. In the first scenario, as in the testing of a circuit, the system is assumed to be in a test-bed and test commands are sent in order to draw inferences about the state of the system. This is a problem of off-line diagnosability or *testability*. In the second scenario, as in the diagnostics of the exhaust gas recirculation system, the system is assumed to be operating when test commands are sent, and thus uncontrollable events may occur during the execution of the test commands. The focus of these papers is on the design of appropriate test commands for diagnosing failed states. The problem of testability was further considered in [8] where the focus was on sensor configuration for testability purposes. A manufacturing application (piston manufacturing cell) motivated this latter extension.

The diagnosis of equipment failures in heating, ventilation, and air conditioning (HVAC) units motivated the development of a different approach for failure diagnosis of DES. This approach, described in [102, 103, 104,

105, 100], is language-based in the sense that failures are modeled as unobservable events. The automaton model of the system accounts for the normal behavior of the system as well as for the behavior of the system in failure modes. The goal is to detect the occurrence of failure events and identify which failure events have occurred based on on-line observations of the system behavior. We discuss this approach further in the next section.

The area of communication networks is one where many different problems of failure diagnosis arise. We mention the problem of conformance testing of communication protocols, where the objective is to determine, by the use of test sequences, if the actual "black box" implementation of a protocol indeed realizes the specification automaton for this protocol. This problem is conceptually related to that of testability mentioned above. The types of faults considered involve incorrect state transitions in the implementation of the automaton. We refer the reader to [81] for an example of work on "fault coverage" of conformance tests. Researchers have also considered the problem of failure diagnosis in network management, i.e., during the operation of the communication network (a problem of on-line failure diagnosis). Automata models have been used to detect and then identify failures of the dynamic behavior of a communication system. The work in [13, 14] is concerned with failures that cause changes to the transitions of the automaton model of the system; an extension that also considers additions to the transition structure can be found in [90].

The work described so far deals with untimed models of DES. In many applications, timing can be an important consideration in failure diagnosis. In such cases, one needs to resort to timed language models. A generic manufacturing application involving an assembly line with conveyors and proximity switches has motivated two approaches to diagnosing timed DES. A distributed strategy based on "time templates" has been proposed and studied in [61, 63, 62]. The idea is to observe the timed trace of events generated by the system and compare it to a set of expected templates that capture timing and sequencing information among observed events. This technique attempts to avoid building a complete model of the system, a step which may lead to combinatorial explosion of the state space. Recently, an extension of the approach in [104] to timed models was proposed in [30]. The timed model used in that work differs from the notion of timed automaton presented in section 3.2 and is based instead on the formalism of [16]. The work in [30] shows that the approach in [104] can be used to diagnose failures in a timed DES after the timed model is suitably "untimed" by the introduction of new transitions that model the passage of time.

This completes our brief (and admittedly incomplete) survey of some recent work on failure diagnosis of DES.

3.6.3 Presentation of One Approach to Failure Diagnosis

Our objective in this section is to give more details on the approach presented in [102, 103, 104, 105, 100] for failure diagnosis. A small example will be used to focus the discussion.

The system to be diagnosed is represented by a DFA G; here, G accounts for the normal and the failed behavior of the system. The event set Σ is partitioned into the set Σ_o of observable events and the set Σ_{uo} of unobservable events. Typically, the observable events are one of the following: commands issued by the controller, sensor readings immediately after the execution of the above commands, and changes of sensor readings. The unobservable events are failure events or other events which cause changes in the system state not recorded by sensors.

To illustrate our discussion, consider a small mechanical system consisting of a pump, a valve, and a controller. For simplicity, we assume that the pump and the controller do not fail while the valve has two failure modes, a stuck-open failure mode and a stuck-closed failure mode; the stuck-open failure occurs only from the open state of the valve and the stuck-closed failure occurs only from its closed state. In order to build the automaton G modeling the system, we proceed in three steps. First, we obtain the individual models of the components of this system; these are shown in Figure 3.4. The initial states of the components are VC (valve closed), POFF (pump off), and C1 for the controller. The only unobservable events are the stuck failure events of the valve.

Suppose that the system is equipped with just one sensor, a flow sensor whose output is discretized to two possible values: F, indicating that there is a flow and NF, indicating that there is no flow. The second step of model-building is to list the sensor output for all relevant states of the system. This sensor map, denoted by the function h, is listed in Table 3.1. The $\bullet$s in the table stand for the state of the controller and are used to indicate that the sensor map is independent of the controller state.

h(VC, POFF, $\bullet$)	=	NF
h(VO, POFF, $\bullet$)	=	NF
h(SC, POFF, $\bullet$)	=	NF
h(SO, POFF, $\bullet$)	=	NF
h(VO, PON, $\bullet$)	=	F
h(VC, PON, $\bullet$)	=	NF
h(SC, PON, $\bullet$)	=	NF
h(SO, PON, $\bullet$)	=	F

TABLE 3.1. The sensor map for the pump-valve-controller system

The final step of model-building is to combine the sensor map in Table 3.1 with the components models in Figure 3.4 to obtain the "complete" system model G. The details of this step are omitted here (the interested

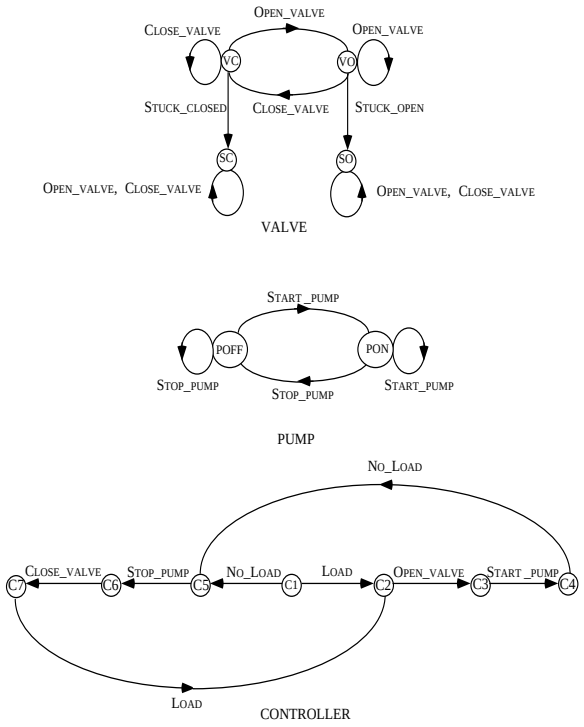

FIGURE 3.4. Component models for the pump-valve-controller system

reader may consult [105]); the resulting G is depicted in Figure 3.5. For convenience, the states of G are denoted by numbers in the figure instead of triples (valve state, pump state, controller state). Note that the observable transitions in the component models have been renamed to include the sensor map information at the state entered by the transition; also note that unobservable transitions are denoted by dashed arcs in Figure 3.5.

Central to the methodology in [104] is an automaton, built from G, called the *diagnoser*. The diagnoser, denoted by G_d, serves two purposes: (i) to verify off-line the diagnosability properties of the system G and (ii) to perform diagnostics when it observes on-line the behavior of G. G_d has for event set the set of observable events Σ_o and for state space, denoted by X_d, a set related to the power set of X. More precisely, a state $x_d \in X_d$ is of the form

$$x_d = \{(x_1, \ell_1), \ldots, (x_n, \ell_n)\}$$

where $x_i \in X$ and ℓ_i is of the form $\ell_i = \{N\}$, or $\ell_i = \{Fi_1 F_{i_2}, \ldots, F_{i_k}\}$, where in the latter case $\{i_1, i_2, \ldots, i_k\} \subseteq \{1, 2, \ldots, m\}$. The label N is to be interpreted as meaning "normal" and the label F_i, $i \in \{1, \ldots, m\}$, as meaning that a failure of the type F_i has occurred. Here, we have partitioned the set of failure events into m distinct types of failures, F_1 to

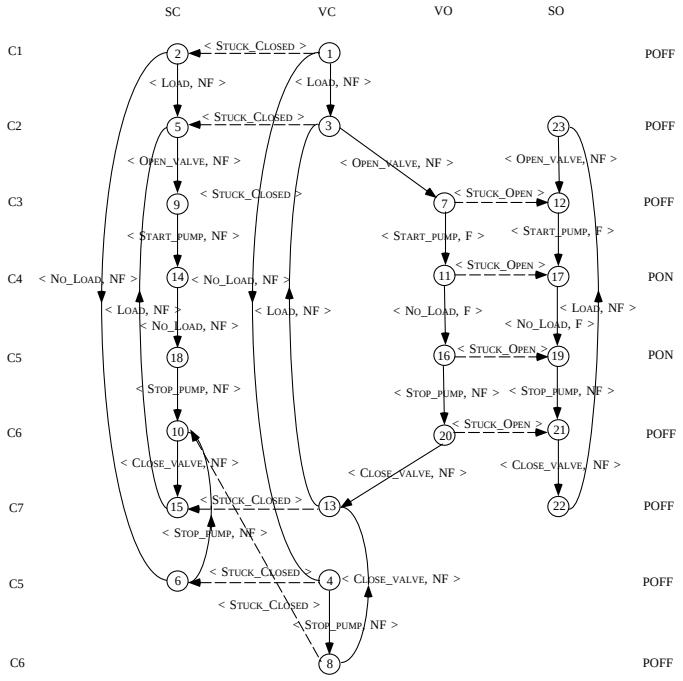

FIGURE 3.5. The system model G for the pump-valve-controller system

F_m.

The diagnoser G_d can be thought of as an "extended observer" for G that gives (i) an estimate of the current state of the system after the occurrence of an observable event and (ii) information on potential past failure occurrences in the form of labels of failure types. In our pump-valve-controller example, let the partition of the set of failure events be chosen as follows: $F_1 = \{\text{Stuck\_Closed}\}$ and $F_2 = \{\text{Stuck\_Open}\}$. Recall that the initial state x_0 of G is state 1, i.e., the state (VC, POFF,C1). Figure 3.6 illustrates the diagnoser G_d for this system. In the figure we represent (x, ℓ) pairs simply as $x\ell$ for clarity.

Informally, the construction of the diagnoser can be summarized as follows. We assume that the system G is normal to start with, hence we define $x_{d,0} = \{(x_o, \{N\})\}$. Next, suppose that the current state of the diagnoser (i.e., the set of estimates of the current state of G with their corresponding labels) is x_1 and let the next observed event be e. The new state x_2 of the diagnoser is computed by (i) determining the set of all possible states the system could be in after the occurrence of the event e, accounting for all possible unobservable events that could *precede* the occurrence of e, and (ii) propagating the labels associated with the state estimates in x_1 to the state estimates in x_2 following certain rules of label propagation. For in-

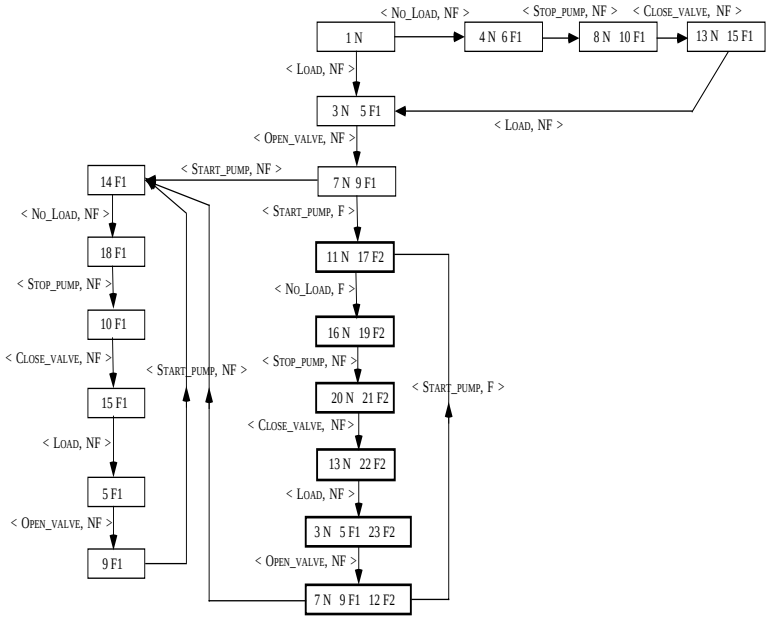

FIGURE 3.6. The diagnoser G_d for the pump-valve-controller system

stance, a state x of G appearing in a state of G_d carries the label N if no prior failure has occurred in the system when it enters x while it receives the label F_i if a failure of type F_i has occurred when the system enters x. Further, the failure labels propagate from state to state, i.e., if a state x receives the label F_i, then so do all of its successors.

Inspection of the diagnoser G_d in Figure 3.6 reveals the presence of a cycle of states where we are unsure if the system is "normal" or if a failure of type "F_2" has happened; this cycle is highlighted in Figure 3.6. Furthermore, we note that the system G can be in cycles involving states 11, 16, 20, 13, 3, 7, and 11 or states 17, 19, 21, 22, 23, 12, and 17, for the *same* sequence of *observable* events (cf. Figure 3.6): <No_Load, F> <Stop_Pump, NF> <Close_Valve, NF> <Load, NF> <Open_Valve, NF> <Start_Pump, F>. When this happens, we say that we have an F_2-*indeterminate* cycle in the diagnoser. It has been shown in [104] that failure types can be diagnosed iff the diagnoser contains no indeterminate cycles. We conclude, therefore, that our system is not diagnosable. More precisely, it is not possible to diagnose occurrences of the valve stuck-open failure in this system, as the system behavior could be a trace of events whose observable part cycles forever in the F_2-indeterminate cycle in G_d.

It is interesting to observe that a small change in the controller model can alter the diagnosability properties of the system. Suppose that the controller of Figure 3.4 is replaced with the controller of Figure 3.7. These

controllers respond to the presence and absence of a load on the system by taking the same control actions; however, they differ in the order in which the pump and the valve are activated.

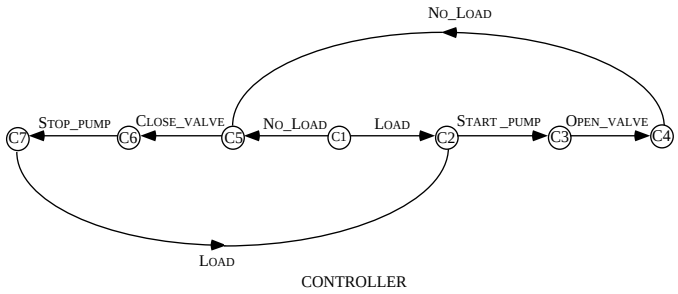

FIGURE 3.7. New Controller model for the pump-valve-controller system

Figure 3.8 depicts the new diagnoser G_d^{new} for this modified system. Inspection of G_d^{new} reveals that there are no F_1 or F_2-indeterminate cycles. Therefore, we conclude that the modified pump-valve-controller system is diagnosable.

To illustrate how the diagnoser can be used on-line to perform failure diagnosis, let us follow a few sample traces in the diagnoser of Figure 3.8. Suppose that the system generates the sequence of observations: <Load, NF> <Start_Pump, NF> <Open_Valve, NF>. Since the flow sensor reads a "no flow" value following the open valve command, we can immediately conclude that the valve is stuck closed. This information can be obtained from the diagnoser G_d^{new} by noting that the state of the diagnoser following the above event sequence, $\{(13, \{F_1\})\}$, tells us that a failure of type F_1 (stuck closed failure) must have happened. Suppose, on the other hand, that the observed event sequence is: <Load, NF> <Start_Pump, NF> <Open_Valve, F>. If we observe a flow following the open valve command, then we can conclude that the valve is normally open. The diagnoser, in this case, is in the state $\{(11, \{N\})\}$. Suppose that the next observable event is <No_Load, F>. At this point, the valve could be in one of two possible states, the normally-open state or the stuck-open state, since either of these two states could give rise to a "flow" value for the flow sensor. This information is provided by the diagnoser by the fact that it transitions into the state $\{(15, \{N\}), (18, \{F_2\})\}$ following the event <No_Load, F>. Finally, suppose that the diagnoser next sees the event <Close_Valve, F>. It concludes immediately that the valve is stuck open since it reaches the state $\{(21, \{F_2\})\}$ following this event.

We note that due to the simplicity of the illustrative example used throughout this section, all the states where we are sure of failures in the two diagnosers presented in Figures 3.6 and 3.8 are actually single-

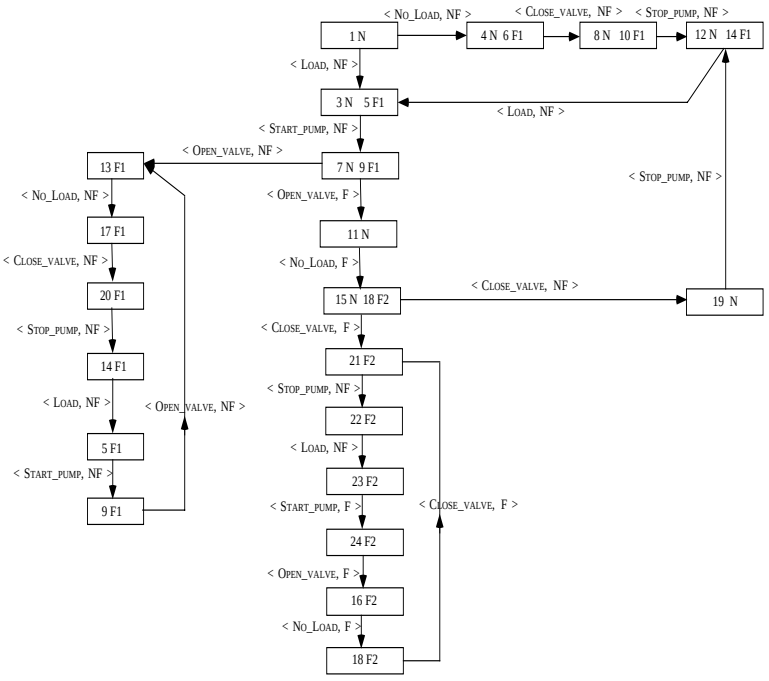

FIGURE 3.8. The diagnoser G_d^{new} for the modified pump-valve-controller system

ton states, i.e., the state of the system is known exactly; this is of course not true in general, as demonstrated by the more comprehensive examples in [105].

The purpose of the discussion in this section was to illustrate how a model-based approach allows for the analysis of the diagnosability properties of a DES, as well as simple on-line implementation of failure diagnosis, namely, by storing the diagnoser automaton and tracking its current state based on observed events.

3.6.4 Some Issues for Future Research

The work that has been done so far on the problem of failure diagnosis in the context of DES is complementary to other approaches to diagnosis such as quantitative methods based on analytical redundancy (concerned with "finer" types of failures) and expert systems (used for systems that are difficult to model). DES approaches are most closely related to fault trees and AI model-based reasoning methods, even though the modeling process and the ensuing analysis proceed quite differently for each approach. DES approaches typically model normal and failed behavior, using the notion of unobservable failure events, and then represent the complete system

model in the form of an automaton, a class of models amenable to analysis using results from automata theory and DES theory. Fault trees are constructed using variable-based models where nodes represent either variables or failure events. Fault tree models do not enjoy the same properties as automata in terms of model composition (when constructing system models from component models) and analysis (e.g., building observers or diagnosers). In [38], a recently-proposed model-based reasoning method for diagnosing DES, propositional temporal logic is used as the modeling formalism. The model is further constrained by exploiting the structure in the system topology. A close examination of the relationship between the work in [38] and other DES work using automata models would be worthwhile and would provide further insight into the merits of these different, yet related, approaches.

Another avenue of investigation that we believe to be promising concerns the problem of diagnosis by *probing*, with specially designed "test sequences", as opposed to diagnosis by means of a "passive observer" as we have discussed so far in this section. We mentioned earlier that probing is at the core of conformance testing of communication protocols (cf. [81]). Probing, in the form of *self-testing*, is also becoming more and more important in microelectronic circuits, where logic for self-testing of a chip is actually becoming part of the logic of the chip itself (cf. [84]). Some DES work has been done recently on the problem of *active diagnosis* [101, 32], where the goal is to design a controller for a DES such that the controlled system enjoys desirable diagnosability properties; however much more remains to be done in this area, and in particular in establishing a connection between current techniques for generating test sequences in protocol engineering and microelectronics and how DES approaches might contribute to the design of such sequences.

We mentioned earlier that current DES work in failure diagnosis has dealt with logical and timed models of DES. The next step is to employ *stochastic* models of DES and study the diagnosability of a given system in that context. Obviously, everything can potentially fail and thus tradeoffs are encountered when logical DES models are built, in the sense of which failure events are to be included in the model. A possible approach would be to attach probabilities to the occurrence of the various failure events along the traces of the system; one could then imagine studying the diagnosability properties in the context of "this failure event (or this failure type) can be diagnosed with probability p." This is certainly an area of research that deserves serious consideration.

Finally, one should not underestimate the many issues that arise in the *implementation* of failure diagnosis systems. For example: (i) at what level of detail should one build the discrete event model of the system (e.g., how many states to include in the model of a valve?)? (ii) how does one build the interface between the discrete level where diagnosis is implemented and the continuous-valued sensor measurements (e.g., how to discretize the

sensor outputs?)? (iii) how does one deal with unmodeled dynamics at the discrete event level (e.g., unmodeled events due to sensor noise)? (iv) how does one deal with *intermittent failures*? (v) how does one build *distributed implementations* of failure diagnosis systems (as would be desirable in a large automated manufacturing system for example)? and (vi) how does one deal with the computational complexity of DES approaches when the system consists of a large number of interacting components (i.e., how to decompose the system into a set of tractable and partially decoupled subsystems)?

3.7 Nondeterministic Supervisory Control

3.7.1 Nondeterminism and semantics of untimed models

Dealing with uncertainty often means dealing with models that include some form of nondeterminism. In this section, we will review some recent work that attempts to extend the theory of supervisory control to (untimed) nondeterministic models of DES. The type of nondeterminism that we wish to capture is best illustrated by considering automaton models of DES. For example, consider the two automata in Figure 3.9. (The initial states are indicated by the short arrows pointing into the states.)

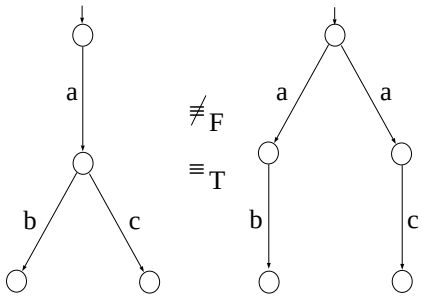

FIGURE 3.9. Comparison of failure equivalence and trace equivalence

The automaton on the left in Figure 3.9 is deterministic while the one on the right is nondeterministic because $f(x_0, a)$ is a set of two states (recall the definition of automaton in section 3.2). These two automata generate the same language, consisting of traces ab, ac, and their prefixes. However, the nondeterministic automaton on the right carries further information in the sense that after the occurrence of event a, the system modeled by this automaton will be able to execute either event b only or event c only, depending on which of the two arcs labeled a is "followed" upon the occurrence of a. This is different from the situation on the left, where both b and c are possible after the occurrence of a.

The theory of supervisory control that we briefly reviewed in section 3.3 is set in the semantics of languages, i.e., two systems (automata) are equivalent if they generate and mark the same languages. Deterministic finite-state automata are used to represent these (regular) languages and for the ensuing analysis and controller synthesis tasks. We say that the notion of system equivalence for the existing theory is *trace equivalence* (or language equivalence), denoted by $\equiv_T$. It is well-known that finite-state nondeterministic automata have the same expressive power for languages as finite-state deterministic automata, namely the class of regular languages [65]. Therefore, any extension of supervisory control whose objective is to allow some form of nondeterminism, either in the uncontrolled system model or the desired system behavior, must adopt a different notion of equivalence than trace equivalence, i.e., a different semantics. The semantics chosen will determine the amount of additional information, beyond the language, required to compare two DES. We observe that Inan has considered in [66] a supervisory control problem where the *supervisor* is allowed to be nondeterministic, while the system and desired behavior are deterministic; the notion of equivalence used in [66] remains trace equivalence. Inan has applied this problem formulation to the gateway synthesis problem in communication networks in [67].

A thorough discussion of many DES semantics that have been proposed and studied in the field of process algebra in computer science can be found in [4]. We will restrict our discussion to four semantics beyond trace, namely failure, trajectory, bisimulation, and isomorphism; these are the semantics that have been considered so far in DES control. Differences between these semantics will be shown by comparing automaton models.

3.7.2 The failure semantics

Returning to the automaton on the right in Figure 3.9, we can associate with each trace s in the language generated by this automaton events that can be *refused* after s, i.e., that cannot occur after s. The event sets that can be refused are called *refusals* and the *refusal set* following trace s is the union of all the possible refusals after s. For example, the refusal set of $s = a$ is

$$\{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}\} .$$

We note here that the set $\{b, c\}$ is not in the refusal set since the automaton cannot refuse both b and c after a. One should think of "refusing" as meaning that the system *can* "deadlock"; for example, in the context of a synchronous composition with another system, if the set of events presented by the other system is in the refusal set of the system, then deadlock will occur since the system cannot execute any of these events.

The failure semantics, introduced in [18] to handle the modeling of deadlock for nondeterministic DES, states that two DES are *failure equivalent*,

denoted by $\equiv_F$, if they are trace equivalent and moreover they have the same refusal set after each trace in the language. Thus the two automata in Figure 3.9 are not failure equivalent since they have different refusal sets after trace a.

Motivated by supervisory control problems that arise in layered network architectures, Overkamp considered in [86, 87] the control of DES where the notion of equivalence is failure equivalence. The plant G (uncontrolled system) and the specification E are both given in the failure semantics. The objective is to design a supervisor S (also in the failure semantics) such that the controlled system $G||S$ "reduces" E in the sense that $\mathcal{L}(G||S) \subseteq \mathcal{L}(E)$ and the refusal set of $G||S$ after any trace $s \in \mathcal{L}(G||S)$ is a subset of that of E after s. The first requirement is as in the basic supervisory control problem in the trace semantics (cf. section 3.3) while the second requirement stipulates that the controlled system may only refuse what the specification can also refuse; in the case of deterministic systems, the second requirement is always satisfied whenever the first is. Furthermore, a requirement is imposed that the supervisor S should be *complete* with respect to the system G in the sense that S should never refuse an uncontrollable event offered by G when doing the composition $G||S$. This is equivalent to saying that S should never disable an uncontrollable event that is possible in G. In this context, [86] contains algorithms for controller synthesis for fully-observed and partially-observed systems and it deals also with partial specifications, i.e., specifications E whose sets of events are subsets of the set of events of the system G.

3.7.3 The trajectory semantics

So far, we have been concerned with DES whose interactions were modeled by the synchronous composition operation. There are situations where a more general form of interaction, where one DES may have priority over another DES for certain events, is desirable. This has motivated the introduction of the *prioritized synchronous composition* by Heymann in [51, 55]. The *prioritized synchronous composition* (PSC) of two automata G_1 and G_2 is

$$G_1 \ {}_A||_B \ G_2 := (X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f, \Sigma_{G_{1,2}}, (x_{01}, x_{02}), X_{m1} \times X_{m2})$$

where

$$f((x_1, x_2), e) := \begin{cases} (f_1(x_1, e), f_2(x_2, e)) & \text{if } e \in \Sigma_{G_1}(x_1) \cap \Sigma_{G_2}(x_2) \\ (f_1(x_1, e), x_2) & \text{if } e \in \Sigma_{G_1}(x_1) \setminus \Sigma_{G_2}(x_2) \setminus B \\ (x_1, f_2(x_2, e)) & \text{if } e \in \Sigma_{G_2}(x_2) \setminus \Sigma_{G_1}(x_1) \setminus A \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The two sets A and B that appear in the definition are subsets of $\Sigma_1 \cup \Sigma_2$; these sets are the sets of priority (one could also say "blocking") events

in the composition of G_1 and G_2. Heymann has shown in [51] that the failure semantics is not adequate when one is dealing with the prioritized synchronous composition of nondeterministic systems. This is because two failure-equivalent nondeterministic systems, when composed in PSC with the same system, may yield results that are not language equivalent. This undesirable property means that a stronger form of equivalence than failure equivalence is necessary when studying nondeterministic DES that are to be composed using PSC. An appropriate semantics in this situation is the *trajectory semantics* of [55]; we denote equivalence under this semantics as $\equiv_{TY}$. The *failure-trace semantics* of [92] is closely related to the trajectory semantics (cf. [107] for further details); in the context of this paper, their difference is irrelevant.

Roughly speaking, in the trajectory semantics, one needs to account for the refusal set not only at the end of trace s, but after each prefix of s. The notion of equivalence then requires that the refusal sets be the same after each event along the trace. The two nondeterministic automata in Figure 3.10 are failure equivalent, but not trajectory equivalent. This is because for trace $s = abd$, the refusal set after the prefix a in the left automaton is not the same as the refusal set in the right automaton after the same prefix of abd.

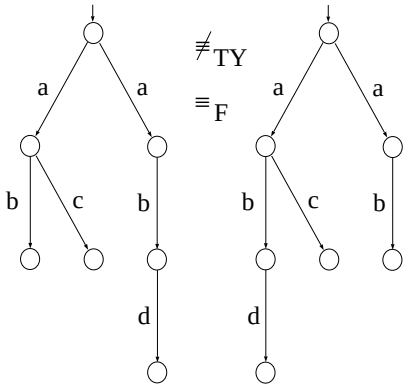

FIGURE 3.10. Comparison of failure equivalence and trajectory equivalence

Shayman and Kumar, in [107, 73], have considered supervisory control problems where the plant and supervisor are allowed to be nondeterministic and are coupled via the PSC operation; thus, both models are to be interpreted in the trajectory semantics. The specification however is given as a language (trace semantics). The motivation for their work comes from the observations that (i) nondeterminism arises naturally in modeling systems when certain internal events are suppressed or when there are unmodeled dynamics, and (ii) PSC is a more powerful form of interaction between the

plant and supervisor than the standard synchronous composition. The priority set of the plant G is the set of controllable and uncontrollable events and the priority set of the supervisor S is the set of controllable events union the set of *driven* events. Driven events are used by the supervisor to "force" the plant to execute a certain command. By the rules of PSC, driven events require the participation of the supervisor, while the plant will follow whenever possible. In nondeterministic systems, the fact that the plant may be unable to execute a driven event command from the supervisor can be used to resolve some of the uncertainty due to nondeterminism (from the point of view of the supervisor), a fact which may allow improved performance in control. The work in [107, 73] considers control problems under full observation, partial observation, as well as nonblocking control issues (in the context of the trajectory semantics).

The control of nondeterministic systems in the trajectory semantics has also been considered by Heymann and Lin in [53, 54]. Their viewpoint is different though, as they approach the control problem from the angle of partial observation. Namely, the nondeterministic system is "lifted" to a deterministic one by the addition of hypothetical unobservable events. This lifted system is consistent with the original model in the sense that its projection retrieves the original nondeterministic model. The nondeterministic specification is also lifted in a manner consistent with the lifting of the system. The power of this approach is that once the problem has been lifted to an equivalent one involving a deterministic system and specification, the results of supervisory control theory in the trace semantics can be used. In this sense, the results in [53, 54] establish a precise relationship between nondeterminism (in the trajectory semantics) and partial observation (in the trace semantics).

3.7.4 The bisimulation semantics

Let us consider a stronger notion of equivalence than trajectory equivalence. Figure 3.11 shows two nondeterministic automata that are trajectory equivalent. Suppose that the notion of equivalence that we are interested in stipulates that any pair of states reached in the two automata after a given trace of events should have the same future behavior in terms of post-language. Then the two automata in Figure 3.11 are not equivalent in that sense since after trace a, if the left automaton is not in the middle state, then it will not have both bc and bd as possible future behaviors, in contrast to the automaton on the right.

This notion of equivalence is formalized by introducing the notion of *bisimulation relation* between two (possibly nondeterministic) automata, termed H and G hereafter.

A *bisimulation relation* is a binary relation, Φ, between subsets of states $S_H \subseteq X_H$ and $S_G \subseteq X_G$ with respect to a set of events Σ_R satisfying [2, 4]:

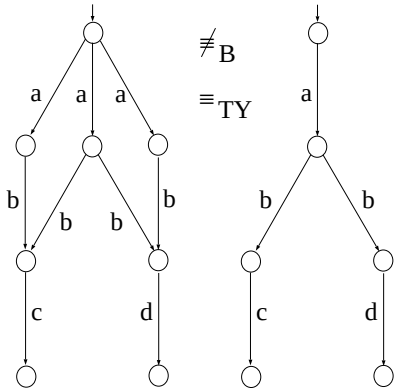

FIGURE 3.11. Comparison of trajectory equivalence and bisimulation equivalence

1. For each $x_H \in S_H$ there exists $x_G \in S_G$ such that $\Phi(x_H, x_G)$.
2. For each $x_G \in S_G$ there exists $x_H \in S_H$ such that $\Phi(x_H, x_G)$.
3. If $\Phi(x_H, x_G)$, $\sigma \in \Sigma_R$ and $x'_H \in f_H(x_H, \sigma)$, then there is a x'_G such that $x'_G \in f_G(x_G, \sigma)$ and $\Phi(x'_H, x'_G)$.
4. If $\Phi(x_H, x_G)$, $\sigma \in \Sigma_R$ and $x'_G \in f_G(x_G, \sigma)$, then there is a x'_H such that $x'_H \in f_H(x_H, \sigma)$ and $\Phi(x'_H, x'_G)$.
5. (Bisimulation with marking) $\Phi(x_H, x_G)$ implies $x_H \in X_{H,m}$ iff $x_G \in X_{G,m}$.

Two DES H and G are considered to be equivalent in the bisimulation semantics, denoted by $H \equiv_B G$, if there exists a bisimulation relation between H and G with $\Sigma_R = \Sigma_H \cup \Sigma_G$, $S_H = X_H$, and $S_G = X_G$.

There has not been work treating the supervisory control of nondeterministic systems in the bisimulation semantics yet. On the other hand, a precise connection between the notions of bisimulation and controllability of a language with respect to another language has been established in [7]. This connection has allowed supervisory control theory to take advantage of efficient algorithms for building bisimulation relations, namely those in [89, 44], by using them to synthesize an automaton that generates the supremal controllable sublanguage, a key step in the solution of supervisory control problems (cf. section 3.3).

3.7.5 The isomorphism semantics

The strongest notion of equivalence between automata is that their graph representations be isomorphic. Isomorphism is clearly stronger than bisimulation equivalence. In [43], Fabian considers supervisory control problems

Researcher(s)	Plant	Spec.	Supervisor	Nonblocking	Observations
Ramadge & Wonham	T	T	Det.	Yes	Partial
Inan	T	T	Non-Det.	Yes	Partial
Overkamp	F	F	Non-Det.	No	Partial
Heymann & Lin	TY	TY	Det.	Yes	Partial
Shayman & Kumar	TY	T	Non-Det.	Yes	Partial
Fabian	T	I	Non-Det.	Yes	Full

TABLE 3.2. Summary of review of nondeterministic supervisory control

where the plant is deterministic, the specification is nondeterministic, and the specification equivalence is isomorphism. The motivation for this semantics is that the nondeterministic specifications model the concurrent operations of several manufacturing subsystems, hence the states carry "physical information" about the system, in fact more information than captured by bisimulation equivalence. This explains the choice of isomorphism by the author. The control problems addressed in [43] are for fully-observed systems, but the issue of blocking is considered.

3.7.6 Discussion

We summarize in Table 3.2 the discussion in this section on some supervisory control problems for nondeterministic systems that have been considered so far in the literature.

We expect that more research will be performed on nondeterministic control, either in the context of the semantics mentioned above or using other semantics. The choice of a semantics is inherently related to the problem at hand. Normally, one would choose the coarsest (in the sense of least restrictive notion of equivalence) semantics necessary to capture the behavioral information that is deemed necessary for a given problem. On the other hand, it should be noted that the use of a more detailed semantics may possibly be advantageous from a computational viewpoint, as demonstrated in [7] where algorithms for bisimulation relations are used for a control problem set in the trace semantics. From a control perspective, the choice of a semantics is related to the amount of information that is passed between the supervisor and the plant, e.g., from enabled events (trace semantics) to refused events (failure and trajectory semantics) to some form of state information (bisimulation and isomorphism).

We conclude this section by mentioning other recent work on control of DES where nondeterminism also arises. In [39, 40, 41], DiBenedetto *et al.* have considered the control of *input-output finite-state machines*, where the objective is model matching. The plant, controller, and desired behavior (i.e., model to match) are modeled as input-output finite-state machines.

Essentially, these models are a variation of the automata discussed in this paper where event labels are of the form: *input event / output event*. The interpretation is that upon the submission of the *input event*, the machine (i.e., automaton) will make the indicated transition and output the *output event*. In the work on model matching, the plant is deterministic but the desired behavior can be nondeterministic; in this case the objective is to match a deterministic "sub-behavior" of it. The notion of equivalence upon which "sub-behavior" is characterized cannot be directly compared to the notions of equivalence discussed in this section, partly because it involves comparing a deterministic model with a potentially nondeterministic one. It resembles the trace semantics in the sense that the sequences of *input/output* events of the two machines must be the same; however, it also resembles trajectory semantics where objects must be consistent following each event in a sequence.

The feedback loop of model matching is depicted in Figure 3.12 and is reminiscent of the feedback loop used in the control of continuous-variable systems, where the "external" input V is received by the controller M_2, which uses this input and the system's output to determine the actual input U applied to the system M_1. The right-hand side of the figure represents the model M whose behavior must be matched. Inputs labeled W represent *disturbances* that cannot be altered by the controller; one can think of them as the *uncontrollable events*. A precise connection between this recent work on model matching and the existing supervisory control theory (as discussed in section 3.3) can be found in [7, 6]. Essentially, if *both* M_1 and M are deterministic, then, in the case of full observation, the two frameworks are "equivalent" in the sense that a problem instance in one framework can be mapped to a problem instance in the other framework whose solution in that framework is the same as the solution obtained in the original framework.

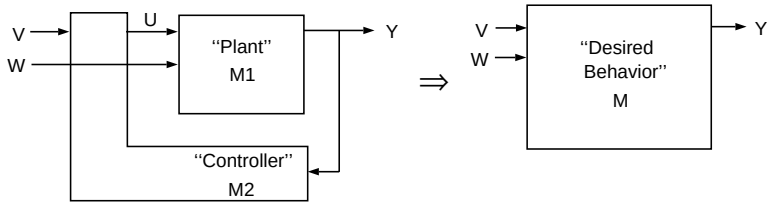

FIGURE 3.12. The feedback loop of input-output finite-state machine control

3.8 Hybrid Systems and Optimal Control

Hybrid systems have recently emerged as traditional *time-driven* systems are becoming integrated with *event-driven* components. The most common instance of such a system arises when a conventional plant is allowed to interact with an event-driven "supervisor". The role of the supervisor is to initiate actions in the plant through commands which are viewed as "events" (e.g., start a machining process in a manufacturing environment), and to process information from the plant that is issued also in the form of "events" (e.g., machine has failed). In the latter case, the responsibility of the supervisor may be to take corrective actions (e.g., initiate shutdown or replace failed tool). Several modeling frameworks have been proposed for such systems (e.g., [49, 42, 1]). Common views adopted for hybrid systems include modeling multiple time scales, a slow time scale for the discrete dynamics and a fast time scale for the continuous dynamics; and continuous-time systems where discrete events are injected as jump processes. However, in a broader sense, any system combining time-driven and event-driven dynamics is hybrid, and the control involved need not be limited to events; instead, the control may well be modeled through a continuous variable, such as the ramp-up speed of a machine.

Hybrid systems naturally arise in a large class of manufacturing processes and open up a seemingly vast application area for control methodologies aimed at integrating the time-driven and event-driven components of these processes. In this section, we will limit ourselves to a description of a major problem area encountered in manufacturing today which calls for a modeling framework based on integrating event-driven with time-driven components of a process and for a mathematical formulation well-suited to optimal control techniques.

3.8.1 Statement of the Problem

Let us consider a manufacturing system of the type commonly encountered in the process industries (e.g., steelmaking) or the more sophisticated environment of semiconductor fabrication facilities. In such systems, "raw material" enters the manufacturing plant and is processed at several machines (or work centers) to become a finished product. Machines have finite capacity, so that only a certain number of parts can be processed by a particular machine at any one time. Material handling equipment transfers parts from one machine to another and buffering in between machines is usually necessary. The operation of such a system is usually described by a DES, typically a queueing network where *servers* (i.e., machines) are viewed as stochastic time delays representing the time required to complete service. However, a server has to be viewed not as simply a system component where a part incurs some time delay, but instead as a device which performs actions on the parts being processed in such a way as to

change their physical state (e.g., size, temperature, chemical composition) according to some continuous dynamics. The events affecting any such part correspond to starting and stopping the server that processes the part.

In order to best describe the hybrid dynamic system in this manufacturing setting, we associate two types of states to any piece that is being manufactured: (a) The *physical state* of the piece, characterized by variables such as temperature, width, length, etc., and (b) The *temporal state* of the piece, identifying the amount of time it has spent in the system since an order for it was placed or since it began processing as raw material. The former is subject to time-driven dynamics described by differential (or difference) equations, while the latter satisfies event-driven dynamics commonly described through the max-plus equations we saw in section 3.3. In terms of system performance, the physical state measures product quality. The temporal state measures on-time delivery and demand responsiveness. The basic tradeoff is due to the fact that product quality can be made near-perfect if processing times and quality control are not constrained; this, however, would result in an infeasible system with unstable inventories and product response times. Conversely, if no adequate time is devoted to certain parts of the process, much of the product will be scrapped due to poor quality. By monitoring both states, there is a tremendous opportunity for dynamically controlling manufacturing processes through both the time-driven component and the event-driven component. For example, by adjusting the settings of a particular machine, a piece of marginal quality may be improved; it might otherwise be processed in uncontrolled fashion and ultimately be scrapped resulting both in wasted material and wasted time (which was used for this piece instead of one of adequate quality).

To set up an appropriate hybrid system model, consider a simple single-stage manufacturing process where parts are indexed by $k = 1, 2, \cdots$ An arriving part at time a_k is characterized by the temporal state a_k and the physical state $x_k(a_k)$. The part normally waits in a queue preceding the server responsible for the process until its turn comes to be processed. The waiting time is denoted by $w_k \geq 0$. Thus, at the point the part is ready to enter the server, its full state is given by $(a_k + w_k, x_k(a_k + w_k))$. Note that a waiting process may alter the physical state of the part according to some time-driven dynamics (e.g., cooling results in a lower temperature state), so that it need not be true that $x_k(a_k + w_k) = x_k(a_k)$. This adds to the complexity of the process and related control problem.

Time-Driven system component. The time-driven component of this system arises when the part begins some, possibly highly complex, process at the server. The dynamics are generally characterized by

$$\dot{x}_k = f(x_k, u_k, t) \tag{3.31}$$

with initial state $x_k(a_k + w_k)$ and some, possibly multivariate, control u_k.

In a closed-loop setting, the control may depend not only on the observed physical state $x_k(a_k + w_k)$, but also on the temporal state $a_k + w_k$, thus revealing the hybrid nature of this system.

Event-Driven system component. The event-driven component corresponds to the dynamics of the temporal state. For the simple queueing system we are considering here, the standard max-plus recursive equation that characterizes the departure time d_k of the kth part is

$$d_k = max(d_{k-1}, a_k) + s_k(u_k) \tag{3.32}$$

What is important to observe here is that d_k depends on the process time $s_k(u_k)$ which depends on the control u_k. It is often the case, in fact, that $s_k = u_k$, i.e., control in this hybrid system is applied by selecting the processing time itself. This is the case in many heat treatment operations of metal or semiconductor wafers, for instance.

The hybrid nature of the simple dynamic system defined by (3.31)-(3.32) is further seen when one considers cost metrics of interest that reflect product quality traded off against response time as discussed earlier. As an example of a typical problem, let D_k be a desired target delivery date for the kth part, and let q_k denote a target quality level for the physical state $x_k(d_k)$ of the part (assumed scalar for simplicity). Then, over a time horizon defined by a production volume of N parts, we may introduce a quadratic cost metric of the form

$$J_N = \sum_{k=1}^{N} \left[\alpha(d_k - D_k)^2 + \beta(x_k(d_k) - q_k)^2\right] \tag{3.33}$$

which we seek to minimize by selecting control variables u_k, $k = 1, \cdots, N$. In this setting, the control is selected only at the start of the kth processing cycle. More generally, the control may be $u_k(t)$ with $\in [a_k + w_k, d_k]$. It should be clear that the full state of the hybrid system at any time instant involves the temporal and physical state of all parts present in the system (queue and server) at the time.

3.8.2 Using Optimal Control in systems with Event-Driven Dynamics

A natural question that arises is whether one can use classical optimal control techniques for a problem such as the minimization of J_N in (3.33) subject to (3.31)-(3.32). In this respect, a simple but crucial observation is the following. The event-driven dynamics in (3.32) are described by a recursive equation similar in form to a standard discrete-time difference equation. However, the index k does not correspond to time steps, but rather to a counter for the departure events in the DES characterized by

(3.32). Moreover, time no longer acts as a synchronizing agent, that is, the kth state update in (3.32) does not correspond to the kth "clock tick", but rather the asynchronous occurrence of the kth event. Nonetheless, even though the interpretation of k is different, the mathematical representation of the event-driven dynamics is the same as that of a time-driven discrete-time system with a particular form of nonlinearity brought about by the max operation.

We point out that the hybrid system model described by (3.31)-(3.32) is deterministic in nature, whereas event times such as part arrivals and departures are usually stochastic. Note, however, that measuring event times is generally not as susceptible to noise as in measuring continuous-time signals involving variables such as temperature, pressure, etc. Thus, one motivation for seeking closed-loop optimal control solutions is that they may provide dynamic control policies dependent on temporal state variables which are generally easy to observe.

With this brief introduction in mind, one can envision a rich theory for the optimal control of hybrid systems that can parallel the successes of classical methodologies. However, this effort is still in its infancy. Some concrete problems that have been considered very recently in [46, 91] have shown that the discontinuities in the state dynamics caused by the max operation in (3.31) are not insurmountable. In [91], for example, it has been shown that an explicit optimal solution to the following problem can be obtained:

$$\text{Minimize} \quad \alpha d_N^2 + \sum_{k=1}^{N} \left[\alpha d_k^2 + \beta u_k^2\right] \tag{3.34}$$

subject to

$$d_{k+1} = max(d_k, a_{k+1}) + q/u_k^2 \tag{3.35}$$

where the controllable variable u_k may be thought of as the ramp-up speed of a process seeking to bring the physical state of every piece from 0 to a desired state q (e.g., a temperature level).

Clearly, the real challenge will be the development of a theory capable to handle a multistage manufacturing process. It is interesting to note the analogy between: a manufacturing process whose goal is to optimally transfer a part from its "raw material" initial state x_0 to a desired final state x_f determined by specific physical attributes; and the classical optimal control problem of specifying an optimal trajectory to transfer an object in space (e.g., an airplane) from initial state x_0 to a desired final position x_f. In this view, many manufacturing processes can be described through trajectories where control can be applied, not continuously, but at specific points in time and space described by the processing steps and their characteristics.

3.9 Acknowledgments

The work of the first author is supported by the National Science Foundation under grant EEC-95-27422, by AFOSR under contract F49620-95-1-0131, and by the Air Force Rome Laboratory under contracts F30602-95-C-0242 and F30602-97-C-0125. The work of the second author is supported by the National Science Foundation under grants ECS-9312134 and ECS-9509975, and the ARO under grant DAAH04-96-1-0377. It is also a pleasure for the authors to acknowledge the contributions and assistance of the following colleagues and students: George Barrett, Vibhor Julka, Christos Panayiotou, David Pepyne, Meera Sampath, Demosthenis Teneketzis, and Felisa Vázquez-Abad.

[1] T.A. Henzinger A. Rajeev. A. and E.D. Sontag (Eds.). *Hybrid Systems III - Vol. 1066 of Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[2] A. Arnold. *Finite Transition Systems*. Prentice Hall, NJ, 1994.

[3] F. Baccelli, G. Cohen, G.J. Olsder, and J.P. Quadrat. *Synchronization and Linearity*. Wiley, 1992.

[4] J. C. M. Baeten and W. P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Great Britain, 1990.

[5] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin. Supervisory control of a rapid thermal multiprocessor. *IEEE Trans. Automatic Control*, 38(7):1040–1059, July 1993.

[6] G. Barrett and S. Lafortune. Supervisory control, bisimulation and input/output finite state machine model matching. Technical Report CGR97-05, Department of Electrical Engineering and Computer Science, University of Michigan, March 1997.

[7] G. Barrett and S. Lafortune. Using bisimulation to solve discrete event control problems. In *Proc. 1997 American Control Conf.*, pages 2337–2341, Albuquerque, NM, June 1997.

[8] S. Bavishi and E. Chong. Automated fault diagnosis using a discrete event systems framework. In *Proc. 9th IEEE International Symposium on Intelligent Control*, pages 213–218, 1994.

[9] N. Ben Hadj-Alouane, S. Lafortune, and F. Lin. Variable lookahead supervisory control with state information. *IEEE Trans. Automatic Control*, 39(12):2398–2410, December 1994.

[10] N. Ben Hadj-Alouane, S. Lafortune, and F. Lin. Centralized and distributed algorithms for on-line synthesis of maximal control policies under partial observations. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 6(4):379–427, October 1996.

[11] A. Benveniste and P. Le Guernic. Hybrid dynamical systems theory and the SIGNAL language. *IEEE Trans. Automatic Control*, 35(5):535–546, May 1990.

[12] R. Boubour, C. Jard, A. Aghasaryan, E. Fabre, and A. Benveniste. Petri net approach to fault detection and diagnosis in distributed nets. In *Proc. 36th IEEE Conf. on Decision and Control*, San Diego, CA, December 1997.

[13] A. Bouloutas, G. W. Hart, and M. Schwartz. Simple finite-state fault detectors for communication networks. *IEEE Transactions on Communications*, 40(3):477–479, March 1992.

[14] A. T. Bouloutas, G. W. Hart, and M. Schwartz. Fault identification using a finite state machine model with unreliable partially observed data sequences. *IEEE Transactions on Communications*, 41(7):1074–1083, July 1993.

[15] J.G. Braker and G.J. Olsder. The power algorithm in max-algebra. *Linear Algebra and its Applications*, 182:67–89, 1993.

[16] B. A. Brandin and W. M. Wonham. Supervisory control of timed discrete-event systems. *IEEE Trans. Automatic Control*, 39(2):329–342, February 1994.

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

[18] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[19] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[20] P. E. Caines and S. Wang. COCOLOG: A conditional observer and controller logic for finite machines. *SIAM Journal on Control and Optimization*, 33(6):1687–1715, November 1995.

[21] P. E. Caines and Y. J. Wei. The hierarchical lattices of a finite machines. *Systems & Control Letters*, 25:257–263, 1995.

[22] X. Cao. *Realization probabilities - the dynamics of queueing systems.* Springer-Verlag, 1994.

[23] C. Cassandras, S. Lafortune, and G. Olsder. Introduction to the modelling, control and optimization of discrete event systems. In A. Isidori, editor, *Trends in Control. A European Perspective*, pages 217–291. Springer–Verlag, September 1995.

[24] C. G. Cassandras. *Discrete Event Systems: Modeling and Performance Analysis.* Aksen Associates/Irwin, 1993.

[25] C.G. Cassandras and V. Julka. Scheduling policies using marked/phantom slot algorithms. *Queueing Systems: Theory and Applications*, 20:207–254, 1995.

[26] C.G. Cassandras and C.G. Panayiotou. Concurrent sample path analysis of discrete event systems. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 1997. Submitted.

[27] C.G. Cassandras and S.G. Strickland. Observable augmented systems for sensitivity analysis of markov and semi-markov processes. *IEEE Trans. Automatic Control*, 34(10):1026–1037, 1989.

[28] C.G. Cassandras and S.G. Strickland. On-line sensitivity analysis of markov chains. *IEEE Trans. Automatic Control*, 34(1):76–86, 1989.

[29] E. Chen and S. Lafortune. Dealing with blocking in supervisory control of discrete event systems. *IEEE Trans. Automatic Control*, 36(6):724–735, June 1991.

[30] Y.-L. Chen and G. Provan. Modeling and diagnosis of timed discrete event systems - a factory automation example. In *Proc. 1997 American Control Conf.*, pages 31–36, Albuquerque, NM, June 1997.

[31] E. K. P. Chong and P. J. Ramadge. Convergence of recursive optimization algorithms using ipa derivative estimates. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 1:339–372, 1992.

[32] T. Y. L. Chun. Diagnostic supervisory control: A DES approach. Master's thesis, Dept. of Elec. Eng., Univ. of Toronto, 1996.

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

[34] S. L. Chung, S. Lafortune, and F. Lin. Supervisory control using variable lookahead policies. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 4(3):237–268, July 1994.

[35] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya. Supervisory control of discrete-event processes with partial observations. *IEEE Trans. Automatic Control*, 33(3):249–260, March 1988.

[36] G. Cohen, D. Dubois, J.P. Quadrat, and M. Viot. A linear system-theoretic view of discrete event processes and its use for performance evaluation in manufacturing. *IEEE Trans. Automatic Control*, 30:210–220, 1985.

[37] R.A. Cuninghame-Green. Minimax algebra. In *Number 166 in Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, 1979.

[38] A. Darwiche and G. Provan. Exploiting system structure in model-based diagnosis of discrete event systems. In *Proceedings of the Seventh International Workshop on the Principles of Diagnosis, DX-96*, Val Morin, Canada, October 1996.

[39] M. D. DiBenedetto, A. Saldanha, and A. Sangiovanni-Vincentelli. Model matching for finite state machines. In *Proc. of 33rd Conf. Decision and Control*, pages 3117–3124, Lake Buena Vista, FL, USA, December 1994.

[40] M. D. DiBenedetto, A. Saldanha, and A. Sangiovanni-Vincentelli. Strong model matching for finite state machines. In *Proc. of 3rd European Control Conference*, pages 2027–2034, Rome, Italy, September 1995.

[41] M. D. DiBenedetto, A. Saldanha, and A. Sangiovanni-Vincentelli. Strong model matching for finite state machines with non-deterministic reference model. In *Proc. of 34rd Conf. Decision and Control*, pages 422–426, New Orleans, LA, USA, December 1995.

[42] P. Antsaklis et al. (Eds.). *Hybrid Systems II - Vol. 999 of Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[43] M. Fabian. *On Object Oriented Nondeterministic Supervisory Control*. PhD thesis, Chalmers University of Technology, Sweden, 1995.

[44] J. Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Sci. Comput. Programming*, 13:219–236, 1990.

[45] M.C. Fu. Convergence of the stochastic approximation algorithm for the gi/g/1 queue using infinitesimal perturbation analysis. *Journal of Optimization Theory and Applications*, 65:149–160, 1990.

[46] M. Gazarik and Y. Wardi. Optimal release times in a single server: an optimal control perspective. In *Proc. 35th IEEE Conf. on Decision and Control*, pages 3831–3836, December 1996.

[47] P. Glasserman. *Gradient Estimation via Perturbation Analysis.* Kluwer, 1991.

[48] P. Glasserman and D.D. Yao. *Monotone Structure in Discrete-Event Systems.* Wiley, 1994.

[49] R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel (Eds.). *Hybrid Systems - Vol. 736 of Lecture Notes in Computer Science.* Springer-Verlag, 1993.

[50] J. Gunnarsson and J. Plantin. Synthesis of a discrete system using algebraic methods. In *Proc. International Workshop on Discrete Event Systems WODES'96*, pages 220–225. IEE, August 1996.

[51] M. Heymann. Concurrency and discrete event control. *IEEE Control Systems Magazine*, 10:103–112, 1990.

[52] M. Heymann and F. Lin. On-line control of partially observed discrete event systems. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 4(3):221–236, July 1994.

[53] M. Heymann and F. Lin. Discrete event control of nondeterministic systems. Technical Report # CIS 9601, Department of Computer Science Technion, Israel Institute of Technology, January 1996.

[54] M. Heymann and F. Lin. Nonblocking supervisory control of nondeterministic systems. Technical Report # CIS 9620, Department of Computer Science Technion, Israel Institute of Technology, October 1996.

[55] M. Heymann and G. Meyer. An algebra of discrete event processes. Technical Report NASA Memorandum 102848, NASA, Ames Research Center, Moffett Field, CA, USA, June 1991.

[56] Y.C. Ho. On the numerical solutions of stochastic optimization problems. *IEEE Trans. Automatic Control*, 42:727–729, 1997.

[57] Y.C. Ho and X. Cao. *Perturbation Analysis of Discrete Event Dynamic Systems.* Kluwer, 1991.

[58] Y.C. Ho and C.G. Cassandras. A new approach to the analysis of discrete event dynamic systems. *Automatica*, 19:149–167, 1983.

[59] Y.C. Ho, A. Eyler, and D.T. Chien. A gradient technique for general buffer storage design in a serial production line. *International journal of production research*, 17:557–580, 1979.

[60] Y.C. Ho, X.Cao, and C.G. Cassandras. Infinitesimal and finite perturbation analysis for queueing networks. *Automatica*, 19:439–445, 1983.

[61] L. Holloway and S. Chand. Time templates for discrete event fault monitoring in manufacturing systems. In *Proc. 1994 American Control Conference*, pages 701–706, 1994.

[62] L. E. Holloway. On-line fault monitoring of a class of hybrid systems using templates with dynamic time scaling. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III - Verification and Control*, pages 259–269. Springer, 1996.

[63] L. E. Holloway and S. Chand. Distributed fault monitoring in manufacturing systems using concurrent discrete event observations. *Integrated Computer-Aided Engineering*, 3(4):244–254, 1996.

[64] L. E. Holloway, B. H. Krogh, and A. Giua. A survey of Petri net methods for controlled discrete event systems. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 7(2):151–190, April 1997.

[65] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

[66] K. Inan. Nondeterministic supervision under partial observation. In G. Cohen and J. Quadrat, editors, *11th International Conference on Analysis and Optimization of Systems: Discrete Event Systems*, pages 39–48. Springer–Verlag, June 1994.

[67] K. Inan. Supervisory control: Theory and application to the gateway synthesis problem. Preprint, Middle East Technical University, Ankara, Turkey, 1994.

[68] K. M. Inan and P. P. Varaiya. Algebras of discrete event models. *Proc. of the IEEE*, 77(1):24–38, January 1989.

[69] R.M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[70] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics*, 23:462–466, 1952.

[71] R. Kumar, V. Garg, and S. I. Marcus. Predicate and predicate transformers for supervisory control of discrete event dynamical systems. *IEEE Trans. Automatic Control*, 38(2):1232–247, February 1993.

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

[73] R. Kumar and M. A. Shayman. Nonblocking supervisory control of nondeterministic systems via prioritized synchronization. *IEEE Trans. Automatic Control*, 41(8):1160–1175, August 1996.

[74] H.J. Kushner. *Approximation and Weak Convergence Methods for Random Processes with Applications to Stochastic System Theory.* MIT Press, 1984.

[75] H.J. Kushner and D.S. Clark. *Stochastic Approximation for Constrained and Unconstrained Systems.* Springer-Verlag, 1978.

[76] H.J. Kushner and G. Yin. Stochastic approximation algorithms for parallel and distributed processing. *Stochastics*, 22:219–250, 1987.

[77] F. Lin. Diagnosability of discrete event systems and its applications. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 4(2):197–212, May 1994.

[78] F. Lin, J. Markee, and B. Rado. Design and test of mixed signal circuits: a discrete event approach. In *Proc. 32nd IEEE Conf. on Decision and Control*, pages 246–251, December 1993.

[79] F. Lin and W. M. Wonham. Decentralized supervisory control of discrete-event systems. *Information Sciences*, 44:199–224, 1988.

[80] H. Marchand and M. Le Borgne. Partial order control and optimal control of discrete event systems modeled as polynomial dynamical systems over Galois fields. Tech. Rep. 3273, INRIA, Rennes, France, October 1997.

[81] R. E. Miller and S. Paul. Structural analysis of protocol specifications and generation of maximal fault coverage conformance test sequences. *IEEE/ACM Transactions on Networking*, 2(5):457–470, October 1994.

[82] S. Morioka and T. Yamada. Performance evaluation of marked graphs by linear programming. *International Journal of Systems Science*, 22:1541–1552, 1991.

[83] T. Murata. Petri nets: Properties, analysis, and applications. *Proc. of the IEEE*, 77(4):541–580, April 1989.

[84] B. T. Murray and J. P. Hayes. Testing ICs; getting to the core of the problem. *IEEE Computer*, 29(11):32–38, November 1996.

[85] J. S. Ostroff. *Temporal Logic for Real-Time Systems.* Research Studies Press and John Wiley & Sons, 1989.

[86] A. Overkamp. *Discrete Event Control Motivated by Layered Network Architectures.* PhD thesis, Rijksuniversiteit Groningen, The Netherlands, 1996.

[87] A. Overkamp. Supervisory control using failure semantics and partial specifications. *IEEE Trans. Automat. Contr.*, 42(4):498–510, April 1997.

[88] C. M. Özveren, A. S. Willsky, and P. J. Antsaklis. Stability and stabilizability of discrete event dynamic systems. *Journal of the ACM*, 38(3):730–752, July 1991.

[89] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, June 1987.

[90] Y. Park and E. Chong. Faul detection and identification in communication networks: A discrete event systems approach. In *Proc. 33rd Allerton Conf. on Communication, Control, and Computing*, September 1995.

[91] D.L. Pepyne and C.G. Cassandras. Modeling, analysis, and optimal control of a class of hybrid systems. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 1998. To appear.

[92] I. Phillips. Refusal testing. *Theoretical Computer Science*, 50:241–284, 1987.

[93] A. D. Pouliezos and G. S. Stavrakakis. *Real time fault monitoring of industrial processes.* Kluwer Academic Publishers, 1994.

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

[95] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proc. of the IEEE*, 77(1):81–98, January 1989.

[96] P.J. Ramadge and W.M. Wonham. Supervisory control of discrete event processes. In D. Hinrichsen and A. Isidori, editors, *Feedback Control of Linear and Nonlinear Systems, Lecture Notes on Control and Information Sciences No. 39*, pages 202–214. Springer Verlag, 1982.

[97] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[98] R. Rubinstein. *Monte Carlo Optimization, Simulation and Sensitivity of Queueing Networks.* Wiley, 1986.

[99] K. Rudie and W. M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Trans. Automatic Control*, 37(11):1692–1708, November 1992.

[100] M. Sampath. *A Discrete Event Systems Approach to Failure Diagnosis*. PhD thesis, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, 1995.

[101] M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete event systems. *IEEE Trans. Automatic Control*, 43(7), July 1998. Abridged version in *Proceedings of the 36th IEEE Conference on Decision and Control*, San Diego, CA, December 1997.

[102] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete event systems. In *Proc. 11th International Conference on Analysis and Optimization of Systems*, pages 73–79. Lecture Notes in Control and Information Sciences, vol. 199, Springer-Verlag, June 1994.

[103] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Failure diagnosis using discrete event models. In *Proc. 33rd IEEE Conf. on Decision and Control*, pages 3110–3116, Orlando, FL, December 1994.

[104] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete event systems. *IEEE Trans. Automatic Control*, 40(9):1555–1575, September 1995.

[105] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Failure diagnosis using discrete event models. *IEEE Trans. Control Systems Technology*, 4(2):105–124, March 1996.

[106] R. Sengupta and S. Lafortune. An optimal control theory for discrete event systems. *SIAM Journal on Control and Optimization*, 36(2), March 1998.

[107] M. Shayman and R. Kumar. Supervisory control of nondeterministic systems with driven events via prioritized synchronization and trajectory models. *SIAM J. Control Optim.*, 33(2):469–497, March 1995.

[108] R. S. Sreenivas. On the existence of supervisory control policies that enforce liveness in discrete-event dynamic systems modeled by controlled Petri nets. *IEEE Trans. Automatic Control*, 42(7):928–945, July 1997.

[109] J. G. Thistle. Supervisory control of discrete event systems. *Mathematical and Computer Modelling*, 23(11/12):25–53, 1996.

[110] P. Vakili. A standard clock technique for efficient simulation. *Operations Research Letters*, 10:445–452, 1991.

[111] F.J Vázquez-Abad, C.G. Cassandras, and V. Julka. Centralized and decentralized asynchronous optimization of stochastic discrete event systems. *IEEE Trans. Automatic Control*, 1995. To appear.

[112] F.J Vázquez-Abad and K. Davis. Strong points of weak convergence: A study using rpa gradient estimation for automatic learning. *Automatica*, 1996. submitted.

[113] N. Viswanadham and T. L. Johnson. Fault detection and diagnosis of automated manufacturing systems. In *Proc. 27th IEEE Conf. on Decision and Control*, pages 2301–2306, Austin, Texas, 1988.

[114] Y. J. Wei and P. E. Caines. Lattice structure and hierarchical COCOLOG for finite machines. In *Proc. 33rd IEEE Conf. on Decision and Control*, Lake Buena Vista, FL, December 1994.

[115] K. C. Wong and J. H. van Schuppen. Decentralized supervisory control of discrete event systems with communication. In *Proc. International Workshop on Discrete Event Systems WODES'96*, pages 284–289, London, August 1996. IEE.

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