Control of Discrete Event Systems

Jana Košecká
University of Pennsylvania

Control of Discrete Event Systems

Abstract

Discrete Event Systems (DES) are a special type of dynamic systems. The "state" of these systems changes only at discrete instants of time and the term "event" is used to represent the occurrence of discontinuous changes (at possibly unknown intervals). Different Discrete Event Systems models are currently used for specification, verification, synthesis as well as for analysis and evaluation of different qualitative and quantitative properties of existing physical systems.

The main focus of this paper is the presentation of the automata and formal language model for DES introduced by Raniadge and Wonham in 1985. This model is suitable for the examination of some important control theoretic issues, such as controllability and observability from the qualitative point of view, and provides a good basis for modular synthesis of controllers. We will also discuss an Extended State Machine and Real-Time Temporal Logic model introduced by Ostroff and Wonham in [OW87]. It incorporates an explicit notion of time and means for specification and verification of discrete event systems using a temporal logic approach. An attempt is made to compare this model of DES with other ones.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-92-35.

Control of Discrete Event Systems

MS-CIS-92-35
GRASP LAB 313

Jana Košecká

University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

April 1992

Control of Discrete Event Systems

Jana Košecká

Department of Computer and Information Science

University of Pennsylvania

April 26, 1992

Special Area Exam

Advisor: Ruzena Bajcsy

Contents

List of Figures

List of Tables

Abstract

Discrete Event Systems (DES) are a special type of dynamic systems. The "state" of these systems changes only at discrete instants of time and the term "event" is used to represent the occurrence of discontinuous changes (at possibly unknown intervals). Different Discrete Event Systems models are currently used for specification, verification, synthesis as well as for analysis and evaluation of different qualitative and quantitative properties of existing physical systems.

The main focus of this paper is the presentation of the automata and formal language model for DES introduced by Ramadge and Wonham in 1985. This model is suitable for the examination of some important control theoretic issues, such as controllability and observability from the qualitative point of view, and provides a good basis for modular synthesis of controllers. We will also discuss an Extended State Machine and Real-Time Temporal Logic model introduced by Ostroff and Wonham in [OW87]. It incorporates an explicit notion of time and means for specification and verification of discrete event systems using a temporal logic approach. An attempt is made to compare this model of DES with other ones.

1 Introduction

Before starting the discussion on the different modeling approaches for DES, we will give an intuitive definition of a dynamic system; a more formal description can be found in Appendix A.

Systems in general can be seen as compositions of elements, whose relations and interactions are governed by known laws (e.g. a mechanical system governed by Newton's laws). A system can be described as *dynamic* when some external forces act on it. The effect of an external force on the system causes a change of the system state, which can be expressed in terms of internal parameters of the system, not necessarily observable (measurable), but causing possibly at a later time the change of output parameters. Based on the characteristics of the system's components and their interactions, systems can be classified as discrete or continuous, time-invariant or time-varying, linear or nonlinear, deterministic or nondeterministic etc.

The familiar class of dynamic systems are so called Continuous Variable Dynamic Systems (CVDS), where the physical world is described by differential equations, and the state of the system changes continuously. In case of DES the state changes at discrete instants of time. Due to the different nature of DES, so far there is no general method for modeling such systems, which would eventually serve as a analog of differential equations for CVDS. Different characteristics of DES which give rise to various modeling methods are described in the following two sections.

1.1 DES Characteristics

Discrete Event Systems (which are the focus of this paper) are mostly man-made systems arising in the domains of manufacturing, robotics, organization and delivery services, vehicular traffic, and computer and communication networks. Events in these systems may correspond, for example, to the transmission of a packet in communication systems, completion of a task or machine failure in manufacturing, etc. The behavior of these systems is truly *nonlinear* and they have *time-varying parameters*. Due to the non-terminating interaction with the environment these systems are often affected by unexpected interventions, which means that they are *discontinuous*. Moreover, they often demonstrate *uncertain behavior*, caused by the fact that available measurements and inputs are often disturbed by noise. Therefore, the future evolution of the system may be unpredictable (nondeterministic). The interactions between particular components are complex and no longer governed by known physical laws describable by differential equations. Due to the complex structure and interactions of these systems, there is a need to model and analyze them at different *hierarchical* levels. Because of the special nature of these systems, in the past different formal methods were proposed for their modeling, emphasizing different aspects of the system design and analysis. The main objective of these efforts was to assure the appropriate behavior of the system and its full functionality in the given environment, by means of appropriate control.

1.2 Modelling of DES

Several attempts have been made to model DES analytically, but there is still no unified theory that supports all the features that one would desire of a full theory of DES. The main distinction can be made between *logical DES models* and *timed or performance DES models.* In the former ones the concept of time is implicit and the time events occur is ignored and only the order in which they occur is considered. In the latter time is incorporated as a part of the model.

Logical models have been successfully used to study different qualitative properties of DES. The behavior of the system using logical models is expressed in terms of system trajectories, i.e. listings of events that occur along the sample path. Typically the set of possible trajectories is specified first. This can be done using some form of transition structure (automata, Petri nets) or by means of algebraic equations (CSP, finitely recursive processes), or by logical calculus (temporal logic). Further, some constraints can be imposed on the system specifying it's desired behavior (expressed in terms of all admissible trajectories). The trajectories are then investigated whether they satisfy the desired properties expressed by the constraints. Properties of interest may include stability, convergence, correct use of resources, correct event ordering, deadlock, liveness etc. The need for examining these different qualitative properties has resulted in the development of a large variety of tools employed in such areas as semantics of concurrent programs, communicating sequential processes, synchronization issues in operating systems, communication protocols in networking and logical analysis of digital circuits. Logical models may be used as a basis of verification and synthesis of DES.

Timed models on the other hand are more suitable for answering performance related questions. These models can be further classified according to whether the timing of events is known a priori as *nonstochastic* (e.g. timed Petri nets, min-max-algebra) or whether the event timing is modeled by making some statistical assumptions as *stochastic* (e.g. Markov chains, queueing networks) models. Since the time is inherently continuous variable, performance measures are often formulated in terms of continuous variables and these models often rely on smoothing properties of the "expectations" or on the "average" operator. In stochastic models it is usually not that hard to specify the set of admissible trajectories, but very often the analytic solutions are very complex, if not infeasible. This

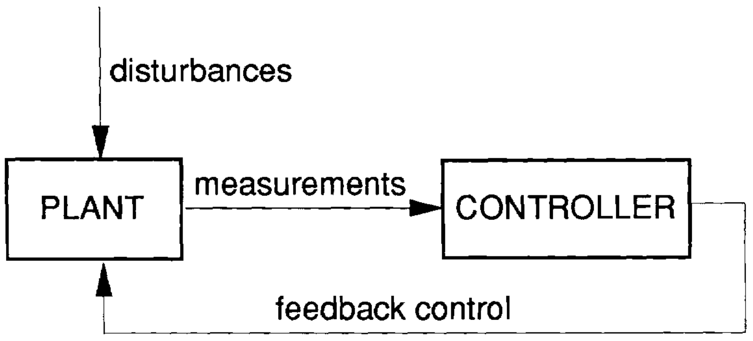

Figure 1: Control diagram

if not infeasible. This has led to the use of simulation as a tool for quantitative analysis. Namely perturbation analysis methods [Ho87] for estimating gradients of performance measures can provide some answers to the questions dealing with quantitative properties of DES. Typical performance criteria of interest are: average throughput, flow or wait time, work in progress etc.

In order to describe the systems under investigation in a uniform way, we adopt the terminology of control theory. The system together with the environment in which it is located can be referred to as a *plant* (i.e., subject of control). Its correct behavior is achieved by designing the *controller* that will interact with the plant. The control diagram representing this situation is shown in Figure 1. Since the open loop behavior of the plant is often unsatisfactory, the role of the feedback after determining the current state of the plant is to take corrective action by issuing appropriate control commands. It is of fundamental importance to take into account the environment in which our system operates in order to deal with unknown disturbances caused by the environment. These disturbances can have disastrous consequences when they are not taken into account.

Paper Organization The focus of the second section will be the introduction of the automata and formal language model for discrete event systems proposed by Ramadge and Wonham in [RW87b] and the presentation of its suitability for dealing with control-theoretic issues. Notions of supervisory control, controllability and observability are defined and discussed.

The third section deals with an Extended State Machine model with an explicit notion of time and problems of specification and verification of discrete event systems using a temporal logic approach.

We conclude by presenting the foremost characteristics that models provide and mention some ideas for the future work in this area.

2 Automata Model for DES

2.1 Definitions

In this section an automata and formal languages model [RW87b] is introduced and its applicability is demonstrated on some simple examples from operating systems and manufacturing. Different components of the model are recognized and their interaction is investigated. The notion of supervisory control is also introduced.

One can think of modeling the DES as a nondeterministic finite (finiteness is not required) automaton, where transitions between states are labeled by events. In the case of the logical model which we are about to investigate, the overall behavior of the system is specified by possible event trajectories. An event trajectory is specified by listing the events that occur during a particular path. Adopting the notation from formal language theory [HU79], event trajectories can be thought of as strings over the fixed alphabet, where particular elements of the alphabet denote events. Let Σ denote the set of events that the system can generate. Then Σ^* denotes the set of all strings of elements Σ, including the empty string ϵ. The empty string represents no event. The subset of all possible event trajectories is $L \subseteq \Sigma^*$. The subset L represents all event trajectories which are possible for the system and fully characterize its behavior. Futher the *characteristic language* L is required to be prefix closed. A language L is *prefix closed* if

$$L = \{u : uv \in \ L \, for \ some \ v \in \Sigma^*\}.$$

The characteristic language L can be described in terms of a machine which generates all strings from L. We define a generator $\mathcal{G}$ to be a 5-tuple

$$\mathcal{G} = (Q, \Sigma, \delta, q_0, Q_m)$$

Q - is the set of all possible states,

Σ - is the set of all possible events

δ - is the transition function $\delta : \Sigma \text{ x } Q \rightarrow Q$,

q_0 - is the initial state,

Q_m - is the subset of states called marker state, $Q_m \subset Q$.

Marking states are used to distinguish a subset $L_m(\mathcal{G})$ of strings from $L(\mathcal{G})$ that may be "marked" or recorded, perhaps representing completed tasks carried out by the DES.

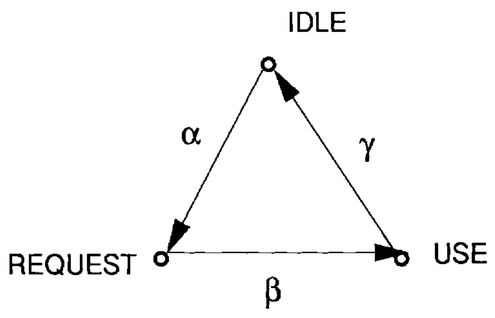

Figure 2: A simple generator. $\Sigma = \{\alpha,\beta,\gamma\}$, $Q = \{$IDLE, REQUEST, WORKING $\}$, $Q_m = \{$IDLE$\}$, $q_0 =$ IDLE

The language *generated* by $\mathcal{G}$ is

$$L(\mathcal{G}) = \{w : w \in \Sigma^* \textit{ and } \delta(w, q_0) \text{ is defined } \}.$$

The language *marked* by $\mathcal{G}$ is

$$L_m(\mathcal{G}) = \{w : w \in L \textit{ and } \delta(w, q_0) \in Q_m\}.$$

State transitions are considered to occur spontaneously, asynchronously, and instantaneously, and their occurrence is signaled (to an observer) by their label σ. A generator $\mathcal{G}$ may have more than one event available for selection at a given state, however, distinct events at a given state always carry distinct labels. A simple example of a generator $\mathcal{G}$ is shown in Figure 2.

The *closed behavior* of L in Figure 2. can be expressed using regular expressions notation, as

$$L(\mathcal{G}) = (\alpha\beta\gamma)^*(\epsilon + \alpha + \alpha\beta)$$

The *marked behavior* for the example in Figure 2. is

$$L_m(\mathcal{G}) = (\alpha\beta\gamma)^*.$$

2.2 Controlled DES

In order to adjoin a means of control, two classes of events have to be distinguished. The events in the firts class are called **controllable events**, i.e. events which can be prevented from occurring. These events can be *enabled* (allowed to occur) or *disabled* (prevented from occurring) at particular instances of time during the system's evolution. The events in the second class denote the **uncontrollable events**. Over these the controller does not have any influence, i.e., they are always enabled. Thus the set of events can be partitioned as $\Sigma = \Sigma_c \cup \Sigma_u$, where events in Σ_c are controllable and events in Σ_u are uncontrollable. Examples of uncontrollable events are machine breakdown in manufacturing applications, loss of packets in communication systems, malfunction of some part of a robot, etc.

The enabling and disabling of a certain event in a particular state is determined by a control pattern for that state. Let

$$\Gamma : \{0,1\}^{\Sigma_c}$$

be the set of all binary assignments to the elements of Σ_c. Each assignment $\gamma \in \Gamma$, i.e. each function

$$\gamma : \Sigma_c \rightarrow \{0,1\}$$

is a **control pattern** (i.e., event σ is enabled when $\gamma(\sigma) = 1$). By introducing the notion of controlled events we can modify the transition function δ of our original generator $\mathcal{G}$ to δ_c

$$\delta_c : \Gamma \times \Sigma \times Q \rightarrow Q$$

where

$$\delta_c(\gamma, \sigma, q) = \begin{cases} \delta(\sigma, q) & \text{if } \delta(\sigma, q) \text{ is defined and } \gamma(\sigma) = 1 \\ undefined & \text{otherwise.} \end{cases}$$

For each fixed control pattern γ a generator $\mathcal{G}(\gamma)$ can be obtained from $\mathcal{G}$ by deleting those events σ which have $\gamma(\sigma) = 0$, i.e., those that the control pattern disables. The control action consists of suitably switching patterns γ, γ', γ", ... in Γ.

Formally the generator $\mathcal{G}_c = (Q,\ \Gamma \times \Sigma,\ \delta_c,\ q_0,\ Q_m)$ is called **Controlled Discrete Event System** (CDES).

So far we have illustrated the expressiveness of the formalism and associated the notions of control with particular events. By doing so we recognized relevant states of the system and characterized its behavior by the language $L(\mathcal{G})$. The language $L(\mathcal{G})$ characterizes an *open-loop behavior* of the system.

2.3 Supervisory control

Since the open-loop behavior of the system is often unsatisfactory, our next main objective is to design a controller for a plant in such a way that the plant behaves in accordance to the specified constraints. Such a controller is called **supervisor**. In another words, the proper supervisor[1]

[1]The proper supervisor has some additional properties, namely it is nonblocking and nonrejecting.

has to generate a sequence of control patterns γ, γ', γ", ..., in response to previously observed events in such a way that the correct behavior of the plant will be ensured. A supervisor can be thought of as the map

$$f : L \to \Gamma$$

specifying for each possible string from our language L the control input $f(w)$ in Γ to be applied at that point. This mapping can be also realized by a pair

$$\mathcal{S} = (S, \phi)$$

where S is an automaton

$$S = (X, \Sigma, \xi, X_0, X_m)$$

where

$$\phi : X \to 2^{\Sigma} \text{ s.t. for each } w \in L(\mathcal{G}),\ f(w) = \phi(\xi(w, x_0)).$$

Thus a clear distinction can be made between the subject of control, "plant", and the agent doing the controlling, "supervisor". The function ϕ determines what events will be enabled and what events will be disabled in the next state, and plays the role of the feedback control.

Transitions between states of the supervisor do not occur freely but are driven by strings $s \in \Sigma^*$ generated by the plant. By coupling $\mathcal{G}$ and $\mathcal{S}$ into a feedback loop we obtain the closed loop system $\mathcal{S}/\mathcal{G}$ called the **Supervised Discrete Event System** (SDES) (see Figure 3). The controllable discrete events of $\mathcal{G}$, are now constrained by the control determined by the states of $\mathcal{S}$. More formally :

$$\mathcal{S}/\mathcal{G}_c = (X \times Q, \Sigma, \xi \times \delta_c, (x_0, q_0), Q_m)$$

where transition map is:

$$\xi \times \delta_c : \Sigma \times X \times Q \to X \times Q.$$

where

$$(\xi \times \delta_c)(\sigma, x, q) = \begin{cases} (\xi(\sigma, x)\delta(\sigma, q)) & \text{if } \phi(\sigma, x) = 1 \\ undefined & \text{if } \phi(\sigma, x) = 0. \end{cases}$$

We will present an example of SDES at the end of this section to demonstrate how the plant and the supervisor operate together.

Let us now examine the class of languages generated by SDES. The **language controlled** by $\mathcal{S}$ in $\mathcal{G}$ of a closed-loop system SDES $L_c(\mathcal{S}/\mathcal{G})$ is defined as follows:

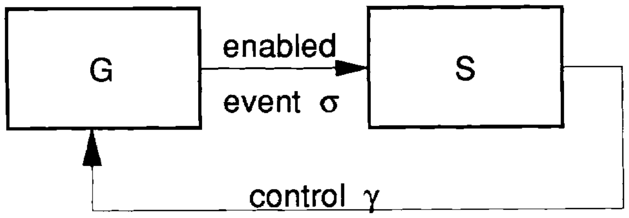

Figure 3: Supervised DES diagram

$$L_c(\mathcal{S}/\mathcal{G}) = L(\mathcal{S}/\mathcal{G}) \cap L_m(\mathcal{G})$$

where $L_m(\mathcal{G})$ is the marked language representing all event trajectories which correspond to the marked (completed) tasks and $L(\mathcal{S}/\mathcal{G})$ is the restriction on this language imposed by supervision. In other words $L_c(\mathcal{S}/\mathcal{G})$ represents all event trajectories corresponding to those strings of the uncontrolled process which are marked and "survive" under supervision. By introducing the control feedback certain events are disabled in particular states and therefore certain strings from $L_m(\mathcal{G})$ are excluded from $L_c(\mathcal{S}/\mathcal{G})$.

2.4 Example

The following example illustrates how the plant (Figure 4) and the supervisor (Figure 5) operate together. The languages $L_m, L(\mathcal{S}/\mathcal{G}), L_c(\mathcal{S}/\mathcal{G})$ defined in the previous section are described in the context of this example.

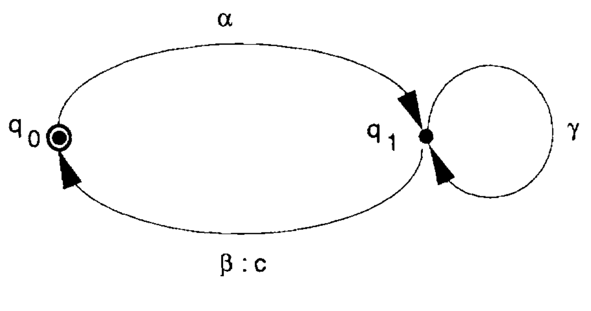

Figure 4: The generator over the alphabet $\Sigma = (\alpha, \beta, \gamma)$, $Q_m = \{q_0\}$, $\Sigma_c = \{\beta\}$

The corresponding marked language is

$$L_m(\mathcal{G}) = (\alpha\gamma^*\beta)^*.$$

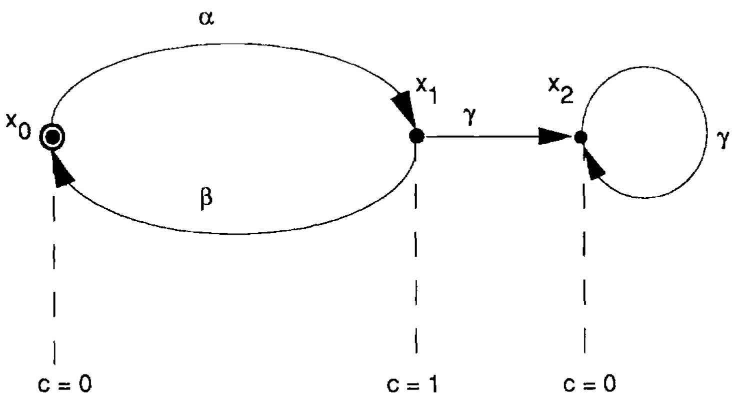

Figure 5: The supervisor $X_m = \{x_0\}$. The supervisor disables event β ($c = 0$ in x_2) immediately after the occurrence of the event γ , and therefore certain strings from $L_m(\mathcal{G})$ are excluded.

It can be seen that

$$L(\mathcal{S}/\mathcal{G}) = (\alpha\beta)^*(1 + \alpha\gamma^*)$$

thus, the language controlled by $\mathcal{S}$ in $\mathcal{G}$ is

$$L_c(\mathcal{S}/\mathcal{G}) = L(\mathcal{S}/\mathcal{G}) \cap L_m(G) = (\alpha\beta)^* = L_m(\mathcal{S}/\mathcal{G}).$$

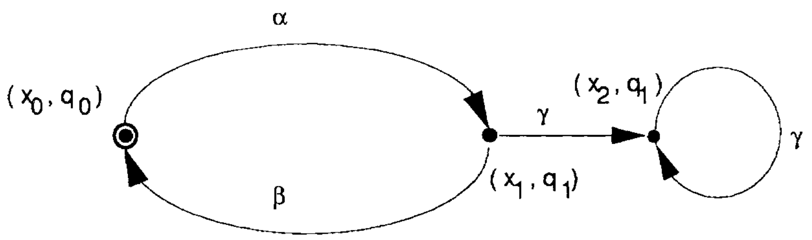

Figure 6: Supervised DES

2.5 Controllability

In the previous section the concept of a supervised discrete event system was introduced. The main objective of the supervisor is to modify the open-loop behavior of the plant so that it satisfies some specified constraints. Having described the plant by a generator $\mathcal{G}$ and specifying a supervisor by an automaton $\mathcal{S}$, the behavior of SDES $L_c(\mathcal{S}/\mathcal{G})$ determined by coupling $\mathcal{S}$ and $\mathcal{G}$ together is defined as the subset of possible event trajectories which survive under the supervision and correspond to the marked tasks. The language $L_c(\mathcal{S}/\mathcal{G})$ represents *the desired behavior* of our system.

In this section we will address the problem of the existence of a supervisor. Namely, given a desired behavior of the plant represented by language K, we will investigate under which circumstances we can find an appropriate feedback control to obtain this behavior.

The problem of obtaining the generator $\mathcal{G}$ which represents the desired behavior K of the plant is not addressed here. One possible way of approaching this problem is by specifying a set of strings which can be excluded from the language L representing all possible trajectories, which can be found in [Ram89].

The existence of a supervisor for a given plant, i.e. the existence of appropriate feedback control, is very closely related to the concept of **controllability**. This concept first introduced by Kalman, plays an important role in theoretical and practical aspects of modern control theory.

A system is said to be **controllable** when based on the information about the current state of the system and by means of appropriate control we can reach any desired state of the system. Within the automata and formal language framework of discrete event systems the concept of controllability is defined in various ways. The authors of [RW87b] adopt the concept of controllability in terms of events. This concept can be formulated formally as follows.

Suppose that $K \subset \Sigma^*$ is the language representing the desired behavior of the plant and $\overline{K}$ is it prefix closure as defined on page 6. Let language $L \subset \Sigma^*$ be representing all possible trajectories. We say that the **language** K **is controllable** if

$$\overline{K}\Sigma_u \cap L \subset \overline{K}.$$

In other words, this condition requires that for any prefix of a string in K, i.e., any $w \in \overline{K}$, if w is followed by an uncontrolled event $\sigma \in \Sigma_u$ in L, then it must be also a prefix of K. In a more intuitive way, since the uncontrollable events cannot be prevented from occurring, it is clear that if such an event occurs, then the path along which that event occurred must remain in K in order for K to be controllable (feasible closed loop behavior). If the plant's desirable behavior can be described by language K, and K is controllable and prefix closed, then the existence of a supervisor is guaranteed by Theorem 6.1 in [RW87b]. This claim also holds in the opposite direction, namely if there exists a proper supervisor for language K, then K is controllable. Moreover the proof is constructive (see Proposition 5.1. in [RW87b]) so the realization of supervisor (S, ϕ) can be obtained from the generator of K. Supervisor construction is illustrated by the example at the end of this section.

The whole class $C(K)$ of controllable sublanguages of K can be obtained. $C(K)$ characterizes all possible behaviors which can be achieved by means of control. The authors in [RW87b] proved also that this class is a partially ordered set under subset inclusion, and is closed under union (Proposition 7.1 in [RW87b]). In the case when the language K specifying the desired behavior of the plant is not controllable, given a set $C(K)$ of all controllable sublanguages of K, we can find a natural approximation of K, $K' \subset K$, which is controllable. This approximation will correspond to the supremum of the partially ordered set of all controllable sublanguages $C(K)$. This problem is stated and elaborated in great detail in [WR87]. The above mentioned closure property of the class $C(K)$ also suggests some approaches on how to deal with the modular design of the supervisory control problem [RW87a].

2.6 Efficient supervisor

The supervisor for a given plant might not be unique. The relation between different supervisors was established, in terms of the projection π. A total function $\pi : X \to \hat{X}$ is a *projection* from S to $\hat{S}$ provided that:

1. π is surjective,
2. $\pi(x_0) = \hat{x}_0$ and $X_m = \pi^{-1}(\hat{X}_m)$,
3. $\hat{\xi} \circ (id_\Sigma \times \pi)(\sigma, x) = \pi \circ \xi(\sigma, x)$ for all (σ, x) where $\xi(\sigma, x)$ is defined,
4. $\hat{\phi} \circ \pi = \phi$.

The projection is displayed in the following diagram:

$$\begin{tikzcd}
\Sigma \times X \arrow[r, "\xi"] \arrow[d, "\mathrm{id} \quad \pi"'] & X \arrow[d, "\pi"] \\
\Sigma \times \hat{X} \arrow[r, "\hat{\xi}"] & \hat{X}
\end{tikzcd}
\qquad
\begin{tikzcd}
X \arrow[rd, "\phi"] \arrow[d, "\pi"'] & \\
\hat{X} \arrow[r, "\hat{\phi}"'] & \{0,1\}^{\Sigma}
\end{tikzcd}$$

where $\hat{S}$ is being referred to as a **quotient of** S under π.

After constructing the proper supervisor $\mathcal{S}$ such that

$$\overline{K} = L(\mathcal{S}/\mathcal{G})$$

we can define an equivalence relation on Σ^*. Two strings are **control-equivalent** $s \sim s'$ if for all $\sigma \in \Sigma_c$, $s\sigma \in \overline{K}$ iff $s'\sigma \in \overline{K}$. Namely, two strings are control-equivalent if the control action immediately following either one is the same for every $\sigma \in \Sigma$.

Recall from automaton theory that the equivalence relation $\mathbf{e}$ on Σ^* is a *right congruence* if, whenever $s, s' \in \Sigma^*$ and $s \equiv s' (\text{mod } \mathbf{e})$, then for all $t \in \Sigma^*$, $st \equiv s't (\text{mod } \mathbf{e})$.

In general the control-equivalence relation is not necessarily a congruence relation, because after an occurrence of event σ even though the strings $s\sigma$ and $s'\sigma$ will end up having the same relation with respect to the membership in $\overline{K}$, they may not be anymore control-equivalent.

We can think of the control-equivalence relation $\sim$, as a partition of the state space X of the supervisor automaton S, where two states are control-equivalent when they generate the same control pattern. The initial realization of the supervisor might not be efficient. By finer partitioning the state space of S (i.e. getting finer and finer equivalence relations on states) we will be looking for the coarsest congruence relation on the state set of S. This congruence relation states that for any two strings s, s' which are control-equivalent, if they are followed by any event $\sigma \in \Sigma$, $s\sigma$ and $s'\sigma$ will again be control-equivalent. Let us denote the coarsest right congruence which is finer then $\sim$, by the symbol $\approx$ and for $s \in \Sigma^*$ let $[s]$ be the equivalence class of s mod $\approx$. Then

$$\overline{S} = (\overline{X}, \Sigma, \overline{\xi}, \overline{x}_0, \overline{X})$$

where

$$\overline{X} = \{[s] : s \in \overline{K}\},\ \overline{x}_0 = [1]$$

is the automaton of the **efficient supervisor**, where the state space $\overline{X}$ is a quotient structure of the original state space X under $\approx$. Here efficiency characterizes the minimal number of states.

Furthermore, the authors in [RW87b, Section 10] showed, that the efficient supervisor can be obtained by projection from a supervisor directly obtained from the recognizer of the legal language L_g. This implies the main result of the **Quotient structure theorem** [RW87b], which states intuitively that every efficiently constructed supervisor is a quotient (high-level or lumped model) of the desired *closed-loop behavior.* This result is similar in spirit to the Internal Model Principle of Regulator Theory. [2]

2.7 Example

The following example illustrates how to construct a supervisor $\mathcal{S}$ given the desired behavior of the system. An alternative supervisor for the system is described to demonstrate that the supervisor might not be unique. Finally a quotient supervisor is constructed for the given problem, representing an efficient supervisor.

In this example we consider two users of a single resource modeled by the generators $\mathcal{G}_1, \mathcal{G}_2$ (Figure 7.), to the left and to the right respectively.

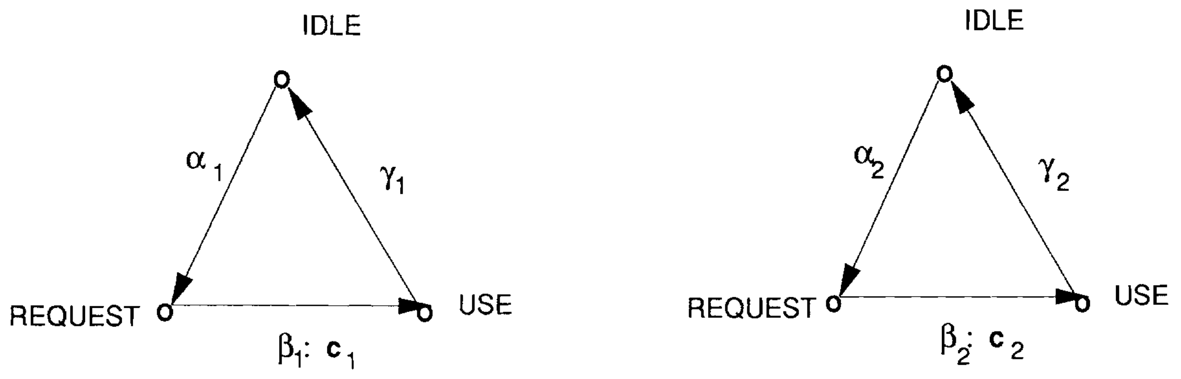

Figure 7: Two users of a resource

The objective of a supervisory control is to manipulate controls c_1 and c_2 in order to satisfy the following synchronization requirements:

[2]The Internal Model Principle of Regulator Theory is expresses the approach by which the appropriate feedback control can be obtained only from the description of the system. Thus having a knowledge of the nature of relations between particular elements of the system is crucial for designing an appropriate feedback control.

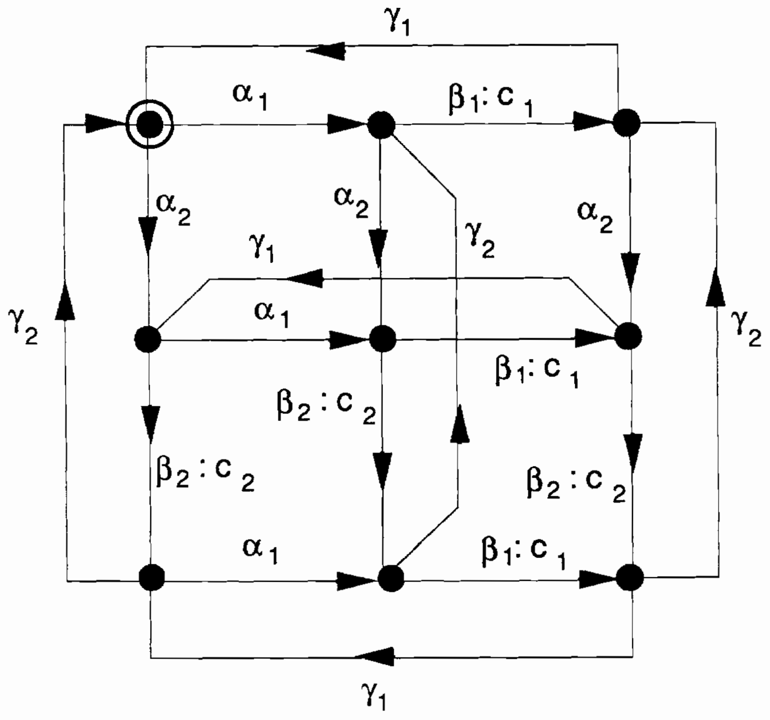

Figure 8: Shuffle product of $\mathcal{G}_1$ and $\mathcal{G}_2$.

1. *Mutual exclusion*: $\mathcal{G}_1, \mathcal{G}_2$ never simultaneously occupy their USE states.

2. *Fair usage*: The USE states of $\mathcal{G}_1, \mathcal{G}_2$ are occupied on a first-come-first-served basis.

We model the joint operation of $\mathcal{G}_1$ and $\mathcal{G}_2$ by the *shuffle product* $\mathcal{G} = \mathcal{G}_1 \| \mathcal{G}_2$. This DES is determined by the concurrent actions of $\mathcal{G}_1$ and $\mathcal{G}_2$ under the assumption that these actions are asynchronous and independent. The shuffle product rules out the simultaneous occurrence of an event in $\mathcal{G}_1$ with an event in $\mathcal{G}_2$ but otherwise places no constraint on their joint behavior.

The state transition diagram is shown in Figure 8. $L_m(\mathcal{G})$ consist of all words over the alphabet $\Sigma = \{\alpha_1, \beta_1, \gamma_1, \alpha_2, \beta_2, \gamma_2\}$.

The problem of formalizing the conditions of our system is not addressed here and it is assumed that the legal behavior of the system $L_g \subset L_m(\mathcal{G})$ is explicitly determined by the automaton on Figure 9. It can be shown that the language L_g is controllable and L_m−closed. Therefore, by Theorem 6.1. in [RW87b] the proper supervisor $\mathcal{S} = (S, \phi)$ exists. The state diagram for L_g can serve as a state diagram for $\mathcal{S}$ and the feedback map ϕ is defined as

$$\phi(x)(c_1) = \begin{cases} 1 & \text{if an edge labeled } \beta_1 \text{ issues from x} \\ 0 & \text{otherwise} \end{cases}$$

Table 1 provides a complete description of the state feedback ϕ. The supervisor might not be unique, so in the second row of the Table 1 there is a feedback ϕ_o for an alternative supervisor $\mathcal{S}_o$

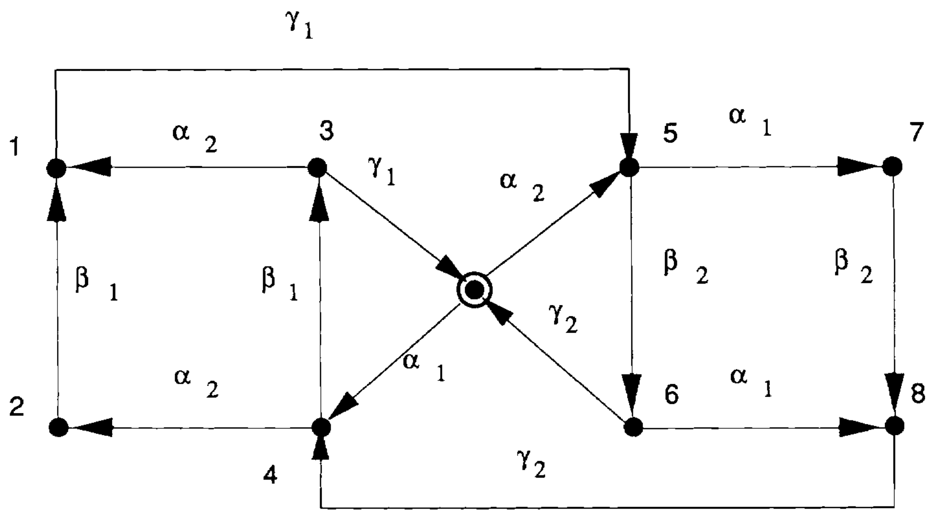

Figure 9: Recognizer for L_g legal behavior

State	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
ϕ	00	00	10	00	10	01	00	01	00
ϕ_0	00	10	10	10	10	01	01	01	01
π	x_0'	x_1'	x_2'	x_3'	x_4'	x_5'	x_6'	x_7'	x_8'

Table 1: Feedbacks for $\mathcal{S}$, $\mathcal{S}_o$ and projection π for example on Figure 7.

determining the same language as $\mathcal{S}$, i.e.

$$L(\mathcal{S}_o/\mathcal{G}) = L(\mathcal{S}/\mathcal{G}) = \overline{L_g}.$$

From $\mathcal{S}_o$ one can construct a new supervisor $\mathcal{S}'$ by the projection $\pi : \mathcal{S}_o \rightarrow \mathcal{S}'$ described in the last row of Table 1. The new supervisor in Figure 10 has only 5 states and is actually representing a queue that stores events α in the order of occurrence and which are popped by the corresponding events γ.

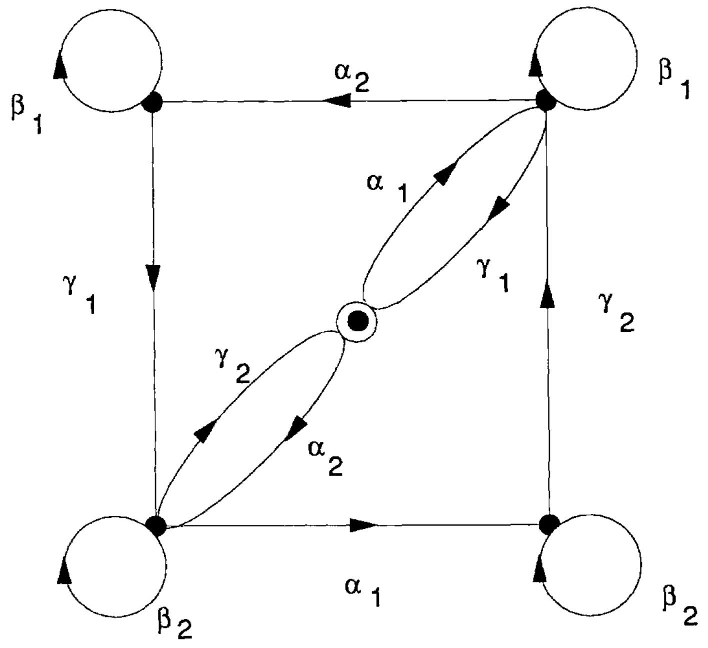

Figure 10: Quotient supervisor

2.8 Observability

The crucial part of the design of an appropriate feedback control for the plant is the amount of information which we can observe monitoring the plant's behaviour. Then based on our observations we can then determine the state of the system and subsequently issue a correct control pattern.

Not in all cases all of the events generated by the controlled discrete event system can be observed by a supervisor. The fact that some of the events might not be observable to the supervising agent has to be taken into account while designing the supervisor. In order to incorporate this case in our model the additional alphabet Σ_o is recognized, such that $\Sigma_o \subset \Sigma$ and all events in Σ_o are **observable**. With the alphabet Σ_o a *projection* P is associated such that

$$P : \Sigma^* \rightarrow {\Sigma_o}^*$$

where P is given by

$$\begin{aligned} P(\epsilon) &= \epsilon \\ P(\sigma) &= \epsilon, \;\; \sigma \in \Sigma - \Sigma_o \\ P(\sigma) &= \sigma, \;\; \sigma \in \Sigma_o \\ P(s\sigma) &= P(s)P(\sigma) \end{aligned}$$

The mapping P defines an equivalence relation $kerP$ on Σ^* such that $(s, s^{'}) \in kerP$ iff $P(s) = P(s^{'})$.

Thus for a given language $L \subset \Sigma^*$, the projection P defines an observable language $P(L) \subset \Sigma_o^*$. In the following section the problem of the existence of a supervisor for partially observable system is addressed.

The question to be answered in the following paragraph is: *Given the language $K \subset L$ representing the desired behavior, does there exist a supervisor $\mathcal{S}$ such that $L_c(\mathcal{S}/\mathcal{G}) = K$, taking into account that not all events can be observed by the supervisor?* The existence of such a supervisor is closely related to the concept of **observability**, widely studied in classical control theory. There the question can also be stated a little bit differently, namely: *Is the information observed by the supervisor through the certain period of time sufficient to determine the state of the system (and therefore to be able to issue an appropriate feedback)?*

The system is said to be **observable** if all it states are observable. Before defining the concept of observability in terms of event trajectories as adopted in [LW88c] some definitions have to be introduced.

Let's define a binary relation act_K such that for $s, s' \in K$, $(s, s') \in act_K$ if there does not exist $\sigma \in \Sigma$ such that either

$$s\sigma \in K \text{ and } s'\sigma \in L(\mathcal{G}) - K \textbf{ or } s\sigma \in L(\mathcal{G}) - K \text{ and } s'\sigma \in K$$

namely, all one step continuations of s and s' that remain in L($\mathcal{G}$) (i.e. that are possible) will yield the same result with respect to membership of K. More formally, with each string $s \in \Sigma^*$ we can associate an active set $A_K(s)$ and an inactive set $IA_K(s)$ defined in the following way:

$$A_K(s) = \begin{cases} \{\sigma : s\sigma \in \overline{K}\} & \text{if s} \in \overline{K} \\ \emptyset & \text{otherwise} \end{cases}$$

$$IA_K(s) = \begin{cases} \{\sigma : s\sigma \in L(G) - \overline{K}\} & \text{if s} \in \overline{K} \\ \emptyset & \text{otherwise} \end{cases}$$

Then we can say that $(s, s') \in act_K$ iff

$$A_K(s) \cap IA_K(s') = \emptyset = A_K(s') \cap IA_K(s).$$

The **language K is** said to be **observable** iff

$$ker\ P \leq act_K$$

i.e., for any $s, s' \in \Sigma^*$ if $P(s) = P(s')$ then $(s, s') \in act_K$. In order for a system to be observable the projection P has to retain sufficient information for a supervisor to decide whether after the enabling or disabling of a particular event the resultant string will be in K. The notion of observability is essential for the existence of appropriate feedback control.

Having the projection mask P on alphabet Σ determining the observable events, the **P-supervisor** (supervisor under the projection P) exists iff language K is controllable and observable (see Theorem 2.1 in [LW88c]).

When the language K does not satisfy the previous condition there is again the possibility to approximate K as it was possible when the language K was uncontrollable. However, in this case a unique maximal controllable and observable sublanguage of K need not exist. In order to look for an observable approximation of K the class of so called **P-normal** languages is examined.

Language $K \subset L(G)$ is **P-normal** when

$$K = L(G) \cap P^{-1}(P(K)).$$

P^{-1} is a mapping which for each string $s \in \Sigma^*$ will return all strings which are equivalent to s under $ker\ P$. Language K is **normal** when it is uniquely determined by it projection P.

In other words, it means that K has to be composed from the union of the cosets of $ker\ P$ which are in L. This situation is represented graphically in Figure 11.

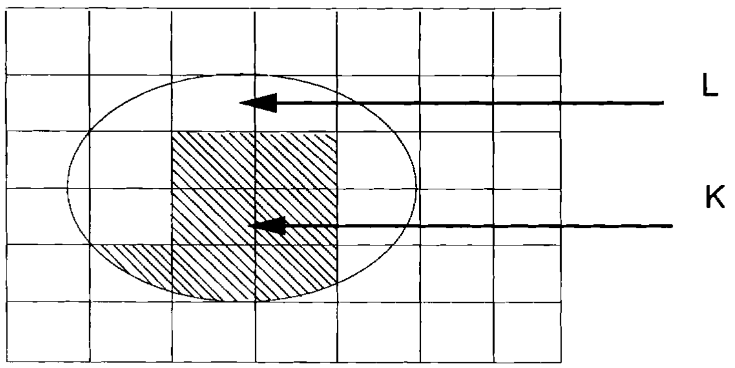

Figure 11: P-normal language (squares correspond to the cosets of $ker\ P$ in L.)

Normality is a stronger condition than observability, i.e., if language K is P-normal then K is also observable.

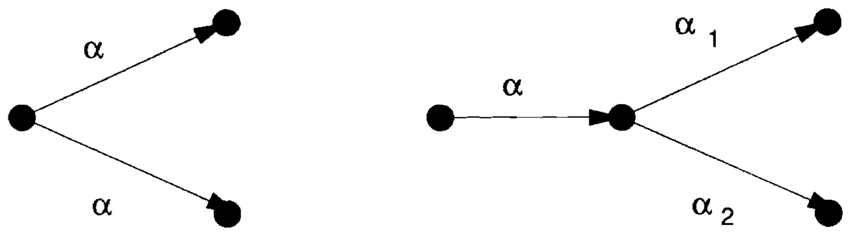

Figure 12: a) nondeterministic transition, b) deterministic transition

The notion of the **unobservable** events also suggests a way for dealing with nondeterministic "noisy" behavior of the plant. A nondeterministic transition α (e.g. Figure 12a.) can be modeled by introducing new unobservable events α_1, α_2 (see Figure 12b.)

Now from the supervisors point of view α is still nondeterministic, but the formal description is now deterministic.

The idea of partial observations determined by the mapping P can be used nicely in a decentralized supervisory control [LW88b]. Local agents can simultaneously supervise DES $\mathcal{G}$ where each of them has some local information (observable by them) and controls associated with them.

3 Real-Time Issues

The main focus of the previous sections was the understanding of basic control-theoretic issues such as controllability and observability in the framework introduced by Ramadge and Wonham in [RW87b], [LW88c]. While doing so a lot of idealizing assumptions were introduced:

1) communications between plant $\mathcal{G}$ and supervisor $\mathcal{S}$ took place with zero time delay and

2) in case the plant is defined as the shuffle product of component generators $\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_k$, events in G_i occur in interleaving fashion so that the true concurrency of events is ruled out,

3) it was assumed that he correct behavior of the system is given by language L_g and the problem of obtaining this correct behavior was not addressed.

The fact that the framework has no means to deal with timeout or delays which in real-life situations can not be ignored may have serious consequences. For example, consider a moving robot which has to stop or turn within a certain amount of time before colliding with another object in the environment.

In this section we will present another representative from the class of logical DES models, namely the Extended State Machine (ESM) and Real-Time Temporal Logic (RTTL) model introduced by Ostroff and Wonham in [OW87]. This model has not only suitable means for investigating whether system trajectories satisfy desired properties, but can be also used for verification and synthesis of DES and incorporates implicit notion of time. The motivation for desiring of above characteristics is outlined below.

Due to the high complexity of discrete event dynamic systems obtaining the model representing the correct behavior of the system is very difficult. The correctness is playing an important role here because mistakes made in the design may have undesirable consequences during operation. The notion of a system's correctness is closely related that of verification. In other words having a model of the system under investigation, we would like to be able to verify (prove) it correctness with respect to the expected legal behavior of the system (plant and controller) fulfilling certain requirements. Since most of the real-life systems are subject to real-time constraints, the absolute timing information is essential in certain applications where certain events have to occur within some time interval after entering a particular state. The main focus of the methods that are addressing these problems, is to provide rigorous computational models and semantically precise specification languages for expressing the requirements imposed on the system behavior and representing plants and controllers. In addition to that some satisfaction relation (or the proof system) is needed in order to reason whether the system satisfies the specified requirements as well as some decision procedures for verification. The proof system then should be proved to be sound with respect to the semantics so that no incorrect program can be proven correct. The completeness of the proof system plays an important role as well, assuring that every correct program can be proven to be correct.

3.1 ESM and RTTL Framework

The approach proposed by Ostroff and Wonham in [OW87] uses the Extended State Machines (ESM) for describing the system and Real-Time Temporal Logic (RTTL) for specifying the required plant behavior and for verifying whether the system satisfies the specifications. The expressive power of EMS is illustrated on the shared track example below.

3.2 Example

Suppose that the plant under investigation consists of two trains which share a common section of the track. On the shared section is a diesel pump for refueling the trains. The train fuel tank can hold up to 1000 gallons of diesel. In order to prevent the two trains from entering the shared track simultaneously, two traffic lights have been installed. The ESM model for the train system can be described as a parallel composition of the plant and the controller.

$$trainSystem = plant \,||\, controller$$

$$plant = train_1 \,||\, train_2 \,||\, pump$$

The extended state machine (EMS) for i-th train is on Figure 13. The EMS consists of the following components:

- ACTIVITIES. A set of *activity variables* where $X = \{$*travel, wait, sharedTrack, pumpConnect, pumpDisconnect*$\}$
- VARIABLES. A *data variable* y_i. Where y_i represents the level of diesel in the tank of the i-th train,
- CHANNELS. A set of *communication channels* $C = \{c_i, m_i, n_i\}$ where c_i is an input channel and m_i, n_i are output channels for the i-th train (e.g. channel c_i is used to receive commands from the controller and m_i and n_i are channels for sending messages to the controller).

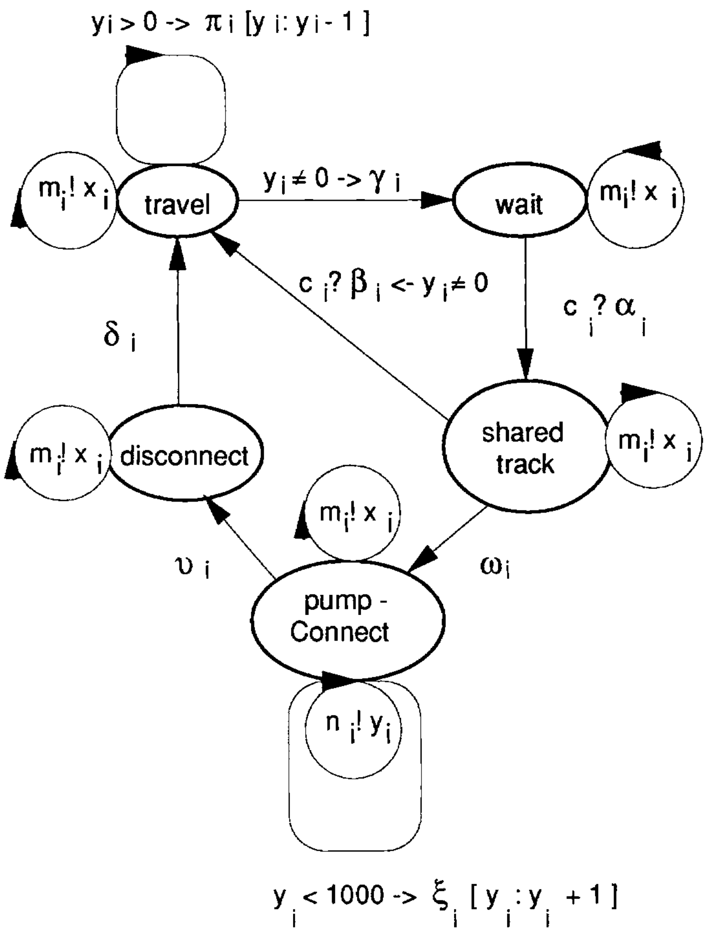

Figure 13: Extended state machine for the i-th train

- TRANSITIONS. The activities represents the states of the system and the *transitions* between states represents events. The events are of the form (*guard* $\rightarrow$ *operation*). The *guards* are boolean valued expressions in the data variables y_i. If the value of the guard is true then the operation is enabled. At a certain state more than one operation can be enabled which allows for nondeterminism. In addition each transition has a lower time bound l and an upper time bound u associated with it. The event is in fact a 3-tuple ($L1$, *guard* $\rightarrow$ *operation*, $L2$), where $L1$ is the exit activity and $L2$ is the source activity. By setting the upper time bound to infinity, we can represent so-called **spontaneous events**, which are never forced to occur. These events may represent the unpredictable interactions with the plant. On the other hand, by setting the upper time bound to a certain finite value, the event is forced to occur within some time interval. These types of **forced events** are suitable for preventing undesirable situations by changing the state of the system.

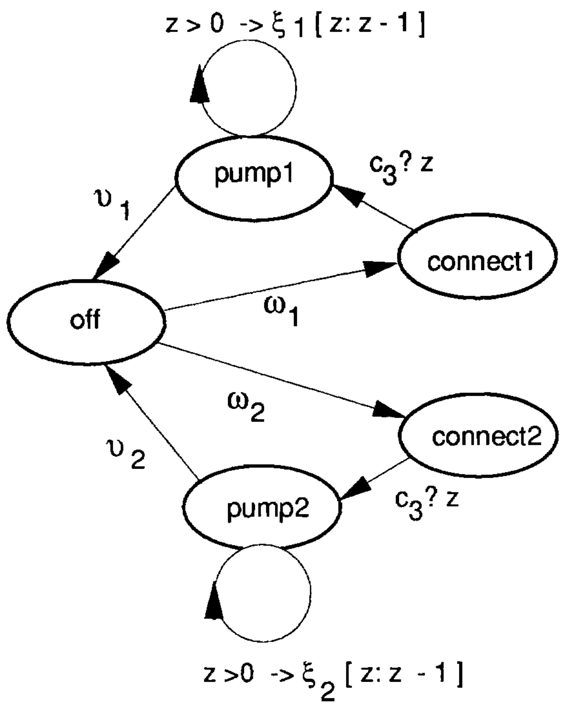

Figure 14: Extended state machine for the pump

In addition to the above mentioned variables of the system, two special variables have to be distinguished. The *next transition* variable **n** whose type is the set of all transitions and the *clock variable* **t** representing the time and used for asserting lower and upper timebounds on transitions.

By looking at the ESM model of the *trainSystem* different types of transitions can be recognized.

- The transition $y_i > 0 \rightarrow \pi_i[y_i : yi - 1]$ is an example of a **local event** changing the variable y_i of the i-th train only.
- The transition $y_i < 100 \rightarrow \xi_i[y_i : y_i + 1]$ is an example of a **shared transition** between the pump and the train. In general shared transition together with time bounds can also serve as a means of control of one ESM's over another. Suppose for example shared transition τ between ESM's M1 and M2. M1 could impose a control over a transition τ in M2, by setting particular finite upper bound therefore forcing the transition τ to occur, whenever both transitions are enabled. Similarly by setting up the upper time bound to infinity for a shared transition in M1, we can observe the transition of M2 by ESM M1.
- The $c_i?m_i$ is a **communicating transition**, in which the plant receives a message from channel c_i. In addition to messages also commands can be sent, where receiving a command

is followed by it's execution. (e.g. $c_i?\alpha_i$ means receive a command from the controller to enter the shared track or $m_i!x_i$ sends the current activity variable to the controller). Send and receive are events which define communicating transition. The upper time bound for this transition is taking into account time for guard evaluation, transition and also the time to do the handshaking. The communicating transition α_i constitutes an example of an forced event, i.e., event which may not be enabled for infinitely many clock ticks.

ESM's provide a designer with a visually simple state transition diagram of the system together with the representational advantages of programming languages (e.g., data variables, assignments, sends, receives, guarded commands).

3.3 Legal Trajectories

One of the main objectives of using formal methods for modeling DES is to assure the desired behavior of the system by means of appropriate control. The behavior of the system can be expressed in terms of the possible state trajectories. The state trajectory is an infinite sequence of states, together with transitions between them. Due to the constraints on the behavior of the system determined by a particular ESM not every path in the state space is possible. The subset of all trajectories determined by the ESM structure which characterizes the actual behavior of the system is called a **set of legal trajectories**. Legal trajectories are formed from initialized trajectories (i.e., trajectories which satisfy an initial condition of the system) and their suffixes. Perceiving ESM's of a system as the syntactic structure, legal trajectories provide both the meaning (formal operational semantics) fully describing its behavior and a basis for deducing system properties. A detailed definition of legal trajectories can be found in [OW87].

3.4 RTTL Specifications

The legal trajectories provide, also the semantics for Real-Time Temporal Logic (RTTL), which is the assertion language for specifying constraints imposed on the systems behavior.

The basis for the RTTL specification and verification language is linear time temporal logic, introduced by Manna and Pnueli in [MP83], with additional proof rules for real-time properties. RTTL formulas are first-order predicate formulas (state formulas) on systems variables (activity or data variables) together with some temporal operators (e.g. $\bigcirc$ - next, $\mathcal{U}$- until, $\diamond$ - eventually, etc.) RTTL formulas are evaluated in a sequence of states as opposed to state formulas which are being evaluated single state. So the fact that a state formula ϕ is satisfied in state s_0 is denoted by

$s_0(\phi) = \mathit{true}$. The satisfaction relation for a RTTL formula is defined inductively in terms of legal trajectories as follows.

Satisfaction. The satisfaction relation is defined for an arbitrary trajectory $\sigma = s_0 s_1 s_2 ...$, where σ_k is given by $\sigma_k = s_k s_{k+1} s_{k+2} ...$.

- If w is a state formula then $\overset{\sigma}{\models} w$ iff $s_0(w) = \mathit{true}$.
- $\overset{\sigma}{\models} \bigcirc w$ iff $\overset{\sigma_1}{\models} w$. $\bigcirc w$ may be paraphrased: w will be true in the next state.
- $\overset{\sigma}{\models} w_1 \mathcal{U} w_2$ iff $\exists k \geq 0$ such that $, \overset{\sigma_k}{\models} w_2$ and $\forall i, 0 \leq i < k, \overset{\sigma_i}{\models} w_1$.

  $w_1 \mathcal{U} w_2$ can be paraphrased as: eventually w_2 will hold and *until* then w_1 holds continuously.

The other temporal operators may be defined in terms of $\bigcirc$ and $\mathcal{U}$ as follows:

- $\Diamond w$ stands for $(\mathit{true}\, \mathcal{U}\ w)$, i.e. *eventually* w will hold true in some state.
- $\Box w$ stands for $\neg(\Diamond(\neg w))$, i.e. *henceforth*, w holds true in all states.
- $w_1 \mathcal{P} w_2$ stands for $(\neg((\neg w_1)\, \mathcal{U}\ w_2))$, i.e if w_2 eventually occurs, then w_1 must *precede* w_2.

So if **S** is a temporal formula specifying the required behavior of the system to be ensured by the controller, and Σ_m is the set of legal trajectories of the system, specification **S** is Σ_m**-valid** if it satisfies all legal trajectories.

An example of temporal logic specifications for the shared track example described previously is given below:

(S1) *Safety.* $\Box(x_1 \in \{\mathit{travel}, \mathit{wait}\} \vee x_2 \in \{\mathit{travel}, \mathit{wait}\})$, specifying the *safety property* saying that both trains may not simultaneously use the shared track, i.e either the first train is traveling or waiting or the second train must be traveling or waiting.

(S2) *Priority.* If one of the trains has been allowed to use the shared track, the currently waiting train must have first priority to use the shared track once the track is vacated:

1. $(\mathbf{n} = \alpha_1 \wedge x_2 = \mathit{wait}) \rightarrow \bigcirc(\mathbf{n} = \alpha_2 \mathcal{P} \mathbf{n} = \alpha_1)$.
2. $(\mathbf{n} = \alpha_2 \wedge x_1 = \mathit{wait}) \rightarrow \bigcirc(\mathbf{n} = \alpha_1 \mathcal{P} \mathbf{n} = \alpha_2)$.

(S3) *Real-time response.* For each train i,

1. $(\mathbf{n} = \alpha_i \wedge t = T) \rightarrow (\bigcirc\Diamond(x_i \neq st \wedge t \leq T+2r)) \wedge (\mathbf{n} \neq \beta_i \mathcal{U} t \geq T+r)$, i.e., no trains should be allowed on a shared track longer than 2r ticks of the clock. However, the train should not be ejected before r ticks of the clock. T is a global variable and r is a constant corresponding to the required response time.

2. $(\mathbf{n} = \omega_i \wedge y_i = Y \wedge t = T) \rightarrow \Diamond(x_3 = pump_i \wedge z = 1000 - Y \wedge t \leq T + 2r)$. Within 2r ticks of the occurrence of the shared event ω_i the pump must be instructed to fill the fuel tank of the train. The activity variable of the pump is x_3 and has one data variable z corresponding to the fuel remaining to be pumped to the train. Y is a global variable.

The specifications (S1), (S2), (S3) refer only to activities and variables of the plant, nothing about the controller is mentioned. This allows to analyze the open loop behavior of the system with respect to the given specifications prior to the synthesis of the controller. This analysis may provide some information about different control policies can and possibly be used for the controller synthesis [Ost89].

3.5 Controllers

Controllers can be implemented in real-time distributed programming languages with constructs for timeout and delays. Since the program is just a sequence of statements (assignments and communications) which are updating the values of data and activity variables, each state of the system can be thought of as a certain relation between variables. These relations are determining the truth values of the guards, i.e., determining which operations will be enabled or disabled. The evolution of the system is determined by the program flow of the controller. The authors in [OW90] chose for implementing a controller distributed real-time programming language CONIC. They also showed that a CONIC task can be represented as an ESM where the transformation between these two representations is straightforward and can be automatized. This provides further evidence for expressive power of ESM's as well as a way in which plant and controller can be treated in a uniform manner. A detailed description of the controller can be found in [OW90] together with it state machine. A schematic description is depicted in Figure 15.

Given an ESM's characterizing the *trainSystem* and given a RTTL specification of the system behavior, there is a way to check whether the closed-loop system is valid, namely whether all specifications are satisfied by all legal trajectories. The decision procedures for checking the system validity of a certain property have been developed and automatized for a small class of real-time

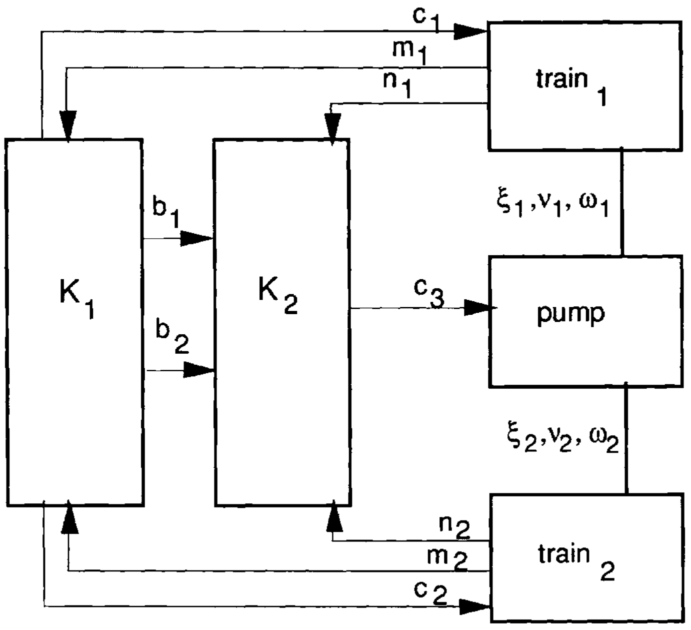

Figure 15: The controller composed from two ESM's (CONIC tasks) K1, K2.

properties, e.g. safety and liveness. In general safety property is defined as making sure that bad things don't happen and liveness has been defined as insuring that the good things eventually occur. These two properties can be successfully verified for the finite state machines, using automated decision procedures. For infinite state systems heuristics have been developed to guide the designer in searching for correctness proofs.

4 Conclusions

The authors in both approaches provided a framework for modeling DES which is suitable for investigating qualitative properties of these system. In both cases the system under investigation was decomposed to a plant and a controller, where the main goal of the controller was to assure the desired behavior of the plant. This decomposition allows to investigate open-loop and closed-loop behaviors of the system separately.

The framework introduced by Ramadge and Wonham successfully addresses the control-theoretic problems and concepts of discrete event dynamic systems and introduces new techniques from automata and formal language theory. The closure properties of regular languages suggest ways of dealing with the modular synthesis of supervisors by defining an algebra of supervisors [RW87a] and investigating concepts of controllability, observability and supervision. The concept of a controllable language allows investigation of the entire class of controllable language, which can be achieved by supervision. The problem of the distributed and hierarchical control can be nicely addressed using this framework. The suitability of this framework for investigating control-theoretic issues lies in the fact that open-loop behavior of the system is clearly separated from the feedback control. Even though some attempts were made to take into account communication delays [LW88a] they did not provide any means to express real-time constraints explicitly.

The notion of controllability and observability are in their nature similar to the safety and liveness properties which are defined in concurrency theory. The safety criterion expresses the need to make sure that bad things will not happen in the system. This can be achieved by making sure that by *observing* the events in the system we can determine it's state and issue an appropriate feedback control to prevent bad things from happening. In case of the liveness property, we want to make sure that good things will happen in our system, or in control-theoretic terms we have to make sure that by means of an appropriate control we can reach the desired state of the system.

In the ESM/RTTL framework authors Ostroff and Wonham demonstrated the significant expressive power of ESM framework as well as its suitability to handle specification and verification problems of DES. Moreover, they showed that even though temporal logic approaches so far have not been adequate for the qualitative analysis of real-time systems due to the interleaved model of concurrency computation, timing requirements can be formulated and verified satisfactorily by introducing lower and upper bounds on transitions. More recent work in the area concentrates mostly on investigating methods for automatic design of controllers as well as better and more automated decision procedures for infinite state systems. The introduction of the time as a special variable raises the question of the appropriateness of the global clock approach, which may cause some problems in the real-life systems. The authors also did not touch upon the scheduling issues of DES and probably assumed some kind of scheduling algorithm which was incorporated implicitly in the model.

The applicability of the proposed frameworks to real-life systems presents some limitations. One of the important characteristics of DES is that they often exhibit uncertain properties due to the fact that inputs and measurements are often disturbed by noise. None of the models provide the means to account for the uncertainty. Even though the models allow a possibility for nondeterministic behavior, they do not have a means to represent and propagate the probabilistic information of events and take this information into an account while investigating systems properties. Furthermore, the models should be able to capture the dynamic and transients aspects of systems behavior as known from continuous variable dynamic systems, as opposed to concentrating only on the output analysis.

The most difficult part of system design, which remains largely unaddressed is the mapping from real-world environments (e.g. robots, sensors, pumps, trains) to formal mathematical models (e.g. events, states, time bounds).

A Appendix

A dynamic system [Son90] can be described as a quadruple

$$\Sigma = (\mathcal{T}, X, U, \phi)$$

where

$\mathcal{T}$ - is a time set, where $\mathcal{T}$ is a subgroup of $(\Re, +)$. In most cases $\mathcal{T}$ is either $\Re$ or $\mathcal{Z}$ (real or integer numbers),

$\mathcal{X}$ - is a set called state space of Σ (output functions),

$\mathcal{U}$ - is a nonempty set called control-value or input-value space of Σ,

ϕ - is a map, $D_\phi \rightarrow X$, called transition map which is defined on subset D_ϕ of set

$$\{(\tau, \sigma, x, \omega) \mid \sigma, \tau \in \mathcal{T},\ \sigma \leq \tau,\ x \in X,\ \omega \in U^{[\sigma,\tau)}\}.$$

D_ϕ - is the domain of transition function ϕ and can be thought of as the set of all possible evolutions of the systems. Choosing a particular one will bring the system to the next state.

ω - is a map from the time interval $[\sigma, \tau)$ into $\mathcal{U}$ and can be thought of as a restriction of control input $u(.)$

The formal description expresses the intuitive notion of a system that evolves in time according to the transition rules specified by ϕ. At each instant of time τ, the state x summarizes all information needed in order to describe the future evolution of the system.

In the most general case the following notation is usually adopted:

$$x(\tau) = \phi(\tau, \sigma, x, \omega)$$

This can be read as the state at time τ resulting from the starting time σ and state x applying the function ω. When the parameters σ and τ are clear from context a simpler notation is often adopted, namely:

$$x(\tau) = \phi(x, \omega).$$

References

[Gil76] W. J. Gilbert. *Modern Algebra with Applications.* John Wiley and Sons, 1976.

[Ho87] Yu-Chi Ho. Performance evaluation and perturbation analysis of discrete event systems. *Transactions on Automatic Control*, AC-32(7):563–572, 1987.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[LW88a] Y. Li and W. M. Wonham. On supervisory control of real-time discrete-event systems. *Information sciences*, 46:159–183, 1988.

[LW88b] F. Lin and W. M. Wonham. Decentralized supervisory control of discrete-event systems. *Information Sciences*, 44:199–224, 1988.

[LW88c] F. Lin and W. M. Wonham. On observability of discrete event systems. *Information Sciences*, 44:173–199, 1988.

[MP83] Z. Manna and A. Pnueli. How to cook a temporal proof system for your pet language. *Proc. Symp. Principles Programming Languages*, 141–154, January 1983.

[Ost89] J. S. Ostroff. Synthesis of controllers for real-time discrete event systems. *Proceedings of 27th IEEE Conf. Decision and Control*, 138–144, December 1989.

[OW87] J.S. Ostroff and M.W. Wonham. State machines, temporal logic and control: a framework for discrete event systems. *Proceedings on 26th Conference on Decision and Control, Los Angeles, CA*, 681–686, December 1987.

[OW90] J. S. Ostroff and W. M. Wonham. A framework for real-time discrete event control. *IEEE Transactions on Automatic Control*, 35(4):386–397, april 1990.

[Ram89] P. J. Ramadge. Some tractable supervisory control problems for discrete-event systems modeled by Buchi automata. *IEEE Transactions on Automatic Control*, 34(1):10–19, January 1989.

[RW87a] P. J. Ramadge and W. M. Wonham. Modular feedback logic for discrete event systems. *SIAM J. Contr. Optimization*, 25(5):1202–1218, January 1987.

[RW87b] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Contr. Optimization*, 25(1):206–230, 1987.

[Son90] E.D. Sonntag. *Mathematical Control Theory. Deterministic Finite Dimensional Systems. Texts in Applied Mathematics 6*, Springer-Verlag, 1990.

[WR87] W.M. Wonham and P.J. Ramadge. On supremable controllable sublanguage of a given language. *SIAM J. Control and Optimization*, 25(3):637–639, 1987.