TM05 1:30

Proceedings of the 1996 IEEE International
Symposium on Intelligent Control
Dearborn, MI • September 15-18, 1996

# A Hybrid Systems Approach to Output Scheduling in ATM Based Real-Time Systems

Michael Lemmon and P.J. Antsaklis
Dept. of Electrical Engineering
University of Notre Dame
Notre Dame IN 46556

## 1. Introduction

There has been recent interest in the use of broadband communication networks to implement distributed control systems (DCS). Such systems use a high speed local area network (LAN) to connect smooth dynamical systems (plants) with their computer controllers. DCS have appeared with increasing frequency in process control and power plant supervision[1]. There has been recent interest in packet switched networks using asynchronous trnsfer mode (ATM) [2] [3] as the communication backbone in DCS [4].

ATM-based control systems can be modeled by both discrete event and continuous-state (smooth) dynamical systems. These systems can therefore be viewed as *Hybrid Control Systems* (HCS) [5] [6] [7] [8]. In this case, the networked plants and controllers form the continuous-state part of the HCS. The arrival and service processes within the network's ATM switches can be modeled as discrete event processes. An important issue in the design of such networks involves the synthesis of a *scheduling protocol* [9] to minimize the transport lag (delay) between network nodes. The scheduling protocol can be treated as a discrete event supervisor controlling the order in which packets (cells) are transmitted by the ATM switch.

This paper proposes using on-line observations to synthesize optimal scheduling protocols. In section 2, an ATM-based real-time system is recast in the hybrid system modeling framework proposed in [6]. In section 3, we summarize our prior work [10] [11] [12] in the on-line identification of optimal DES controllers.

## 2. ATM-based Real-Time Control

This paper considers a real-time system using a single ATM switch to interconnect sensors, actuators, and processors. The following description of an ATM-switch is patterned after [2] and [3]. The inputs to the switch are from plant sensors or are feedback signals generated by the control processors. The outputs of the ATM switch are connected to plant actuators and the input buffer of the control processors. The switch is seen to consist of four parts; the controllers on the input and output ports, the control unit, and the switching fabric. When a cell arrives at an input controller, it is assigned to an output port. The switching fabric routes the cell from the input port to the appropriate output buffer. An *output scheduling protocol* is used to decide the order in which buffered outputs are transmitted. The scheduling protocol is implemented by a control unit. This control unit is also responsible for other high level functions including connection setup, network maintenance and monitoring.

In the design of real-time control systems, an important specification that the communication network needs to satisfy are "hard" time delay constraints [14] [15] [16]. The delay between a sensor reading and the controller's response needs to be constrained so that the overall system preserves stability. Delays in the system can arise from a variety of sources. Delays due to node processing and propagation delays are usually constant and are not affected by the ATM switch. Within the ATM switch, delays occur in the input/output controllers as well as propagation delays within the switching fabric. These delays will also be constant provided the switching fabric is non-blocking. Another delay occurs in the output buffer. This output delay is caused by the transmission order of the queued cells. This type of delay is directly dependent on the choice of output scheduling protocol used by the ATM switch.

Examples of well-known scheduling protocols include round-robin (RR) and first come first served (FCFS) protocols. These protocols are essentially static mechanisms with relatively low computational overhead. In certain cases, however, it may be desirable to use a dynamic scheduling protocol [9]. Dynamic

0-7803-2978-3/96/$5.00 © 1996 IEEE

scheduling decides on the order of served cells based on the current state of the switch's output queues. Such dynamic scheduling protocols can take into account dependencies between various nodes in such a way that can reduce the bandwidth required over existing static protocols. The implementation of such dynamic schemes, however, requires that the dynamics of the arrival processes be known. Such knowledge, however, is rarely available on an a priori basis. This paper proposes using on-line observations of the arrival processes to identify "optimal" scheduling protocols.

To achieve this objective, we first recast the system as a *hybrid dynamical system* [6] [7] [8] [5]. Hybrid dynamical systems consist of discrete event systems interfaced to smooth dynamical systems. One recently studied class of hybrid systems uses a logical DES controller to supervise the behaviour of continuous-state (CSS) plants [6]. The above ATM-network possesses this hybrid nature. The plant and processes connected to the ATM network are modeled as continuous-state systems. The arrival and transmission of cells in the ATM switch mark a discrete event process which arises when controller/plant measurements are packetized at discrete time instants. From the ATM switch's viewpoint, the plants and controllers therefore appear to be a logical DES which we refer to collectively as the *DES plant.* The output scheduling protocol controls the traffic flow through the switch and can be interpreted as the DES controller found in the HCS framework of [6]. This HCS framework is an extension of the supervisory Ramadge-Wonham supervisory control formalism [17].

In order to describe the language generated by the DES plant, we first need to examine the switch's processing of cells in more detail. There are a variety of traffic types handled in ATM networks. The highest priority (class A) traffic is suitable for real-time control. It is a connection oriented traffic mode which guarantees the quality of service (QoS) (i.e., delay time) for a message. The ATM switch's scheduling protocol services the output buffers at regular timeslots. The transmission of a cell from the switch therefore marks a controlled event which occurs in a synchronized manner. The arrival of cells, however, can be generated from a variety of different sources. A cell arrival therefore marks an event which occurs in an asynchronous manner. The ATM switch has no control over the arrival process (except during connection setup) so arrivals can be treated as uncontrollable events. We therefore see that the cell arrivals to and transmission from the switch represent uncontrollable and controllable logical events, respectively.

In particular, let's denote the arrival of a cell from the $i$th source (plant or controller) destined for node $j$ in the network as $\alpha_{ij}$. Arrival events are generated whenever the source packetizes its output and sends it to the ATM switch. Let $\Delta$ be a fixed output quantization interval. Let $y$ be the source's output and define a quantization operator so that $Q_\Delta(y) = \lfloor y/\Delta \rfloor$. It will be assumed that a packet is generated whenever $Q_\Delta(y)$ changes. Denote the servicing of a cell arriving from the $i$th source in the $j$th output buffer as $\sigma_{ij}$. These symbols form the event alphabet $\Sigma$. The controllable and uncontrollable events in this alphabet are denoted as $\Sigma_c$ and $\Sigma_u$, respectively. Since the servicing events occur at regular intervals, they can be used to mark time between batches of cell arrivals. We will assume that only a finite number, $k$, of uncontrollable events can occur between any two consecutive service events. The DES plant language, $L(G)$, is therefore contained within the following regular expression $(\Sigma_u^k \Sigma_c)^*$.

In our case, the objective of output scheduling is to minimize the transmission delay of cells. These delay specifications are "hard" since violation may result in system instability. In order to use logical DES control theory to determine an "optimal" scheduling protocol, this delay requirement must be formulated as a regular specification language, $\bar{K}$. We are interested in determining "optimal" scheduling protocols. The protocol will serve as the DES controller of the DES plant. Assume that the protocol is a regular language $L(S)$ where $S$ is the minimal deterministic finite automaton (DFA). We introduce the usual notion of a controlled finite automaton $G|S$ generating a regular langauge $L(G|S)$. The objective is to determine $S$ so that $L(G|S)$ is the largest controllable sublanguage of the specification $\overline{K}$. We have therefore framed the HCS controller synthesis as a logical DES controller synthesis problem.

## 3. On-line Controller Synthesis

This section summarizes prior work in the use of inductive learning of DES controllers. In the following discussion it is assumed that the plant language, $L(G)$, and the control specification, $\overline{K}$, are regular prefix-closed languages over an event alphabet, $\Sigma$. It is assumed that $\Sigma$ is partitioned into controllable, $\Sigma_c$, and uncontrollable, $\Sigma_u$, events. The plant is assumed to be completely observable. It is assumed that the specification language and the uncontrolled events are initially known. The plant language, $L(G)$, however, is assumed to be unknown. We also assume that we may not know a minimal deterministic finite automaton (DFA) accepting the specification language.

Under the preceding assumptions, we are interested in identifying the DFA for the supremal controllable sublanguage, $K^{\uparrow}$ of the specification.

In the ATM-based real-time system discussed above, it will be extremely difficult to characterize the arrival process in an a priori manner. This means that the DES plant language, $L(G)$, will be initially unknown. Recall, however, that controllability can only be assessed once we know $L(G)$. Therefore if we are to compute $K^{\uparrow}$ we will need to use on-line observations of $L(G)$ in determining $K^{\uparrow}$. In this section, we discuss how Angluin's $L^*$-procedure [13] may be used to accomplish this.

It is well known that computation of the supremal controllable sublanguage can be performed in an iterative manner using the following formula [18]

$$K_0 = \overline{K} \quad (1)$$

$$K_{i+1} = K_i - [(L(G) - K_i)/\Sigma_u]\Sigma^* \quad (2)$$

This iteration produces a sequence of languages by removing uncontrollable plant behaviours in $L(G) - K_i$ from the original specification. Note that each, $K_i$, can be computed from the preceding one once we have observed an uncontrollable plant behaviour which is "illegal" with respect to $K_{i-1}$. The preceding equations therefore provide a means of including observed plant behaviours into the original specification. This iteration therefore serves as the basis for our on-line identification methods.

If we already have a minimal DFA consistent with $\overline{K}$, then the construction of $K^{\uparrow}$ can be done in a straightforward manner. Let $M_0$ be the minimal automaton accepting $\overline{K}$. Assume that this automaton has state space $Q_0$. If we use the specification language automaton $M_0$ to control the plant, then we may generate an illegal behaviour if the specification is uncontrollable. Let $s$ be such an uncontrollable (illegal) string. We therefore know that $s \in L(G)$ but $s \notin \overline{K}$. The only way this illegal behaviour could have occured was through an uncontrollable transition out of a legal state in $M_0$. If we then remove uncontrollable suffixes from $s$, until we obtain a legal behaviour $t$, then the state $q(t)$ associated with this behaviour can be removed from $M_0$. The resulting machine, $M_1$, will accept a language, $K_1$, which is smaller than $\overline{K}$ but which contains $K^{\uparrow}$. By repeating this operation using $K_1$ as $\overline{K}$, we obtain a sequence of machines $M_i$ whose number of states form a monotone decreasing sequence. The resulting iteration is shown below

$$M_0 = M(Q_0, \Sigma, \delta_0, q_0) \quad (3)$$

$$M_{i+1} = M(Q_i - q(t), \Sigma, \delta_{i+1}, q_0) \quad (4)$$

Since $M_0$ had a finite number of states, this means that the iteration terminates after a finite number of updates yielding a DFA accepting $K^{\uparrow}$.

The above method can be used once the automaton for $\overline{K}$ is known. Many times, however, the specification may be given as a regular expression or an informal specification. In these situations the $L^*$-algorithm can be used to help compute the DES controller.

### $L^*$-Algorithm

The $L^*$-procedure constructs approximations to boolean functionals through the use of a *membership oracle* and (sometimes) an *equivalence oracle*. A membership oracle is a function declaring whether or not a given string lies in the target language. An equivalence oracle is a function that declares whether or not an hypothesized DFA accepts the target language. In the event that the conjectured DFA is not accepting, then the oracle returns a *counterexample* illustrating the difference between the two sets. The $L^*$-algorithm is important because it has been shown to converge after a finite number of updates which is polynomial in the size of the minimal DFA and the length of the observed counterexamples [13].

The $L^*$-procedure constructs an *observation table* representing the regular language to be learned. The table is represented by the ordered triple $(S, E, T)$, where $S$ and $E$ are prefix closed and suffix closed languages, respectively. $T : (S \cup S\Sigma)E \rightarrow \{0, 1\}$ is a partial function agreeing with the declarations of a *membership oracle* over strings in $(S \cup S\Sigma)E$. The observation table can be represented in tabular form as shown below. The rows are labeled with strings in $S$ and $S\Sigma - S$. The columns are labeled with strings in $E$. An entry in the table is indexed by a string $s \in S \cup S\Sigma$ and $e \in E$. The value of 1 indicates that the string $se$ is accepted by the membership oracle and 0 indicates otherwise.

| | | $\varepsilon$ | $A$ |
|---|---|---|---|
| $S$ | $\varepsilon$ | 1 | 1 |
| | $aA$ | 1 | 0 |
| | $a$ | 1 | 1 |
| $S\Sigma - S$ | $aB$ | 1 | 1 |
| | $aa$ | 1 | 1 |
| | $aAB$ | 1 | 0 |
| | $aAb$ | 1 | 1 |
| | $B$ | 1 | 1 |
| | $A$ | 1 | 0 |

The preceding table is useful in identifying the Nerode equivalence classes of the minimal DFA consistent

with the table entries. Define a function row : $(S \cup S\Sigma) \rightarrow \{0,1\}^{|E|}$. This function returns a single row of the above table. The rows of the table mark right invariant equivalence classes of a regular language consistent with the table provided the table is *complete* [13]. Any observation table can be completed through an algorithmic procedure using calls to a membership oracle [13]. The table shown above is complete. Define an equivalence relation $R_L$ such that $sR_Lt$ if and only if row$(s)$ = row$(t)$. Because the table is complete, $R_L$,is right invariant and can be used by the MyHill-Nerode characterization to construct a DFA accepting a regular language consistent with the table entries. In particular, it has been shown that the constructed DFA will be minimal in the sense that any other automaton consistent with the table will have more states. The label for the states are the distinct rows of the observation table. The DFA for the above table is shown below.

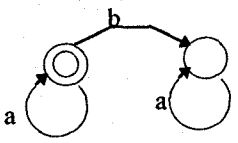


**Figure 1:** Automaton constructed by $L^*$-algorithm

A flowchart for the procedure is shown in figure 2. Given an initial set $S_0^-$ and $E_0^-$ (ususally chosen to be null), the procedure evaluates the observation table, $(S_0^-, E_0^-, T)$. It then completes the table to obtain $(S_0, E_0, T)$ and constructs an acceptor $M_0 = M(S_0, E_0, T)$. This acceptor is then given to the equivalence oracle as an hypothesized acceptor for the target language $\overline{K}$. If the DFA is not equivalent, then the oracle returns a counterexample illustrating where the DFA and the target language disagree. This counterexample is used to construct another completed observation table $(S_1, E_1, T)$. From this table another acceptor is constructed using the Nerode equivalence classes as states and the process repeated until the equivalence query returns no more counter-examples.

## DES Controller Synthesis through the $L^*$ Procedure

In [10] and [12], the $L^*$-procedure was modified to determine optimal logical DES controllers. The modification involved using a time-varying membership oracle. The time-varying membership oracle conditions the oracle's declarations on a list of observed uncontrollable behaviours. If we let $C$ be a collection

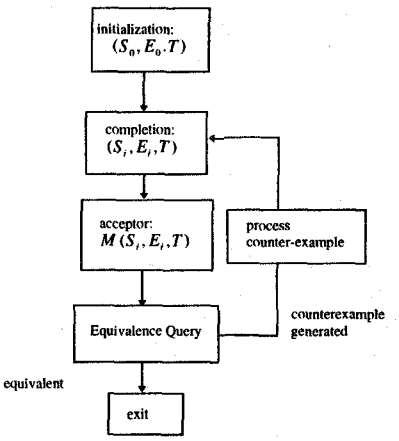


**Figure 2:** High Level Flowchart of $L^*$-procedure

of observed uncontrollably illegal plant behaviours, then the language formed by discarding uncontrollable suffixes of strings in $C$ can be denoted as

$$D_u(C) = \{\, s \in L(G) \text{ such that } st \in C \text{ and } t \in \Sigma_u^* \} \tag{5}$$

The set $C$ is a set of example plant behaviours that can be used to update a partially specified membership oracle. This suggests that we can construct $C$ in an iterative manner. In particular, let $C_{i+1} = C_i \cup \{s_{i+1}\}$ where $s_{i+1}$ is an observed uncontrollable (illegal) behaviour. We therefore have a growing set of observed behaviours that can be used to modify the membership oracle. In particular, we introduce a membership oracle represented by the following partial function

$$T_i(s) = \begin{cases} 0 & \text{if } T_{i-1}(s) = 0 \text{ or } s \in D_u(C_{i+1})\Sigma^* \\ 1 & \text{otherwise} \end{cases} \tag{6}$$

The function $T_i(s)$ can then be used to evaluate and complete a given observation table in exactly the same way as done in the traditional $L^*$-procedure. An immediate consequence of the preceding discussion is that the language consistent with $T_i$ will be $\overline{K} - D_u(C_i)\Sigma^*$.

Provided the set $C$ is finite, then this iteration terminates after a finite number of iterations. A simple example of this procedure is shown below. In this example we consider an event alphabet $\Sigma = \{\varepsilon, a, b\}$ where $b$ is uncontrollable. The unknown plant language is $a^*ba^*$ and the specification is $a^kba^*$ where $k \leq 2$. The initial membership oracle is the prefix-closed specificatin $\overline{K}$. An initial observation table is constructed by taking $S = \varepsilon$ and $E = \varepsilon$. The resulting complete observation table is shown below. The acceptor extracted from this table is shown in figure 3.

| | $\varepsilon$ |
|---|---|
| $\varepsilon$ | 1 |
| $a$ | 1 |
| $b$ | 1 |

The acceptor for the observation table is then used as a controller and generates controllable and uncontrollable plant behaviours. The procedure stops searching when an illegal (w.r.t $\bar{K}$) plant behaviour is identified by the membership oracle. The simulation program written to implement this procedure produced the illegal accepted plant behaviour *aaab*. This behaviour is used to update the observation table as shown below. The acceptor extracted from this table is shown in figure 3.

| | $\varepsilon$ | $a$ | $aa$ |
|---|---|---|---|
| $\varepsilon$ | 1 | 1 | 1 |
| $a$ | 1 | 1 | 0 |
| $aa$ | 1 | 0 | 0 |
| $b$ | 1 | 1 | 1 |
| $ab$ | 1 | 1 | 1 |
| $aab$ | 1 | 1 | 1 |

As before, the acceptor is used to control the plant. In this case no illegal behaviours were uncovered. We then begin using the original specification to generate controllable plant behaviours and see if there are any accepted legal behaviours. In this case an unaccepted legal plant behaviour *baaa* is discovered and added to the observation table. The resulting completed table is shown below. The acceptor extracted from this table is shown in figure 3. This is the supremal controllable sublanguage, $K^{\uparrow}$, for the specification language.

| | $\varepsilon$ | $a$ | $aa$ | $aaa$ |
|---|---|---|---|---|
| $\varepsilon$ | 1 | 1 | 1 | 0 |
| $a$ | 1 | 1 | 0 | 0 |
| $b$ | 1 | 1 | 1 | 1 |
| $aa$ | 1 | 0 | 0 | 0 |
| $ba$ | 1 | 1 | 1 | 1 |
| $baa$ | 1 | 1 | 1 | 1 |
| $baaa$ | 1 | 1 | 1 | 1 |
| $ab$ | 1 | 1 | 1 | 1 |
| $aab$ | 1 | 1 | 1 | 1 |
| $baaaa$ | 1 | 1 | 1 | 1 |

## 4. Summary

In the synthesis framework discussed in [6], an extension of the Ramadge-Wonham formalism can be used to synthesize controllers for HCS. This synthesis,

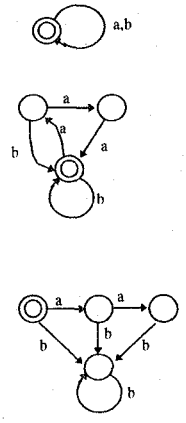


**Figure 3:** sequence of acceptors generated by modified $L^*$-procedure

however, requires a prior extraction of the DES plant language. The analytical determination of such a DES plant language may not be possible. The ATM-based real-time control system examined by this paper is an example of such a system. In this case, a characterization of packet arrival processes will generally be impossible to determine beforehand. In order, therefore, to synthesize the controller, we will need to do an on-line synthesis. This paper has suggested that a modified verion of Angluin's $L^*$-learning procedure may provide the framework for such an on-line synthesis method.

### References


[1] R. Rodd and F. Deravi, *Communication Systems for Industrial Automation*, Prentice-Hall International, 1989.

[2] H. Bruneel and B.G. Kim, *Discrete-Time Models for Communication Systems Including ATM*, Kluwer Acacemic Publisher Inc., 1993.

[3] A. Raha, M. Malcolm, and W. Zhao, "performance evaluation of admission policies in ATM based embedded real-time systems", *Proceedings 19th IEEE Conference on Local Computer Networks*, pp. 129-138, October 2-5, 1994, Minneapolis, Minnesota

[4] C.M. Aras, J.F. Kurose, D.S. Reeves, and H. Schulzrinne, "real-time communication in packet-switched networks", *Proceedings of the IEEE*, Vol 82, No. 1, pp. 122-139, 1994.

[5] A. Nerode and W. Kohn, "models for hybrid control systems : automata, topologies, controllability, observability" in [7]

[6] J.A. Stiver, P.J. Antsaklis, and M.D. Lemmon, "a logical DES approach to the design of hybrid control systems", technical report of the ISIS group ISIS–

94–011, University of Notre Dame, Notre Dame, IN Oct. 1994 (revised: May 1995). to appear in *mathematical and computer modeling* special issue on discrete event systems.

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

[8] A. Nerode, P.J. Antsaklis, S. Sastry (eds) *Hybrid Systems 2*, to appear in Lecture Notes in Computer Science, Springer Veralg, 1996.

[9] K. Ramamritham, R. A. Stankovic, "scheduling algorithms and operating systems support for real-time systems", *Proceedings of the IEEE*, Vol 92, No. 1, pp. 55-67, 1994.

[10] M. Lemmon, P. Antsaklis, X. Yang, C. Lucisano (1995), " "Control System Synthesis through Inductive Learning of Boolean Concepts", *IEEE Control Systems Magazine*, June 1995.

[11] Xiaojun Yang, M.D. Lemmon, P.J. Antsaklis "inductive inference of logical DES controllers using the $L^*$-algorithm", *Proceedings of the American Control Conference*, Seattle, Washington, June 1995.

[12] Xiaojun Yang, M. D. Lemmon, P.J. Antsaklis, "inductive inference of optimal controllers for uncertain logical discrete event systems", *Proceedings International Symposium on Intelligent Control*, Monterey CA, August 1995.

[13] D. Angluin (1987), "Learning regular sets form queries and counter-examples", *Int. J. Information and Computation*, Vol 75, No. 1, pp. 87-106, 1987.

[14] A. Gosiewski and A. Olbrot, "the effect of feedback delays on the performance of multivariable linear control systems", *IEEE Trans. Automatic Control*, Vol AC-25, pp. 729-734, Aug. 1980.

[15] K.G. Shin and C.M. Krishna, and Y-H Lee, "a unified method for evaluating real-time computer controller and its application", *IEEE Trans. of Automatic Control*, Vol AC-30, no. 4, pp. 357-366., Apr. 1985.

[16] Z.V. Rekasius, "stability of digital control with computer interruption", *IEEE Trans. of Automatic Control*, Vol AC-31, no. 4. pp. 356-359, Apr. 1986.

[17] P. Ramadge and W.M. Wonham (1987), "Supervisory control of a class of discrete event processes", *SIAM Journal of Control and Optimization*, Vol 25, No. 1., pp. 206-230, Jan. 1987.

[18] W.M. Wonham and P.J. Ramadge (1987), "on the supremal controllable sublanguage of a given language", *SIAM Journal of Control and Optimization*, vol 25, No. 3, pp. 637-659, 1987.