

SUPERVISION OF DISCRETE EVENT SYSTEMS WITH COMMUNICATION DELAYS

S. Balemi and U.A. Brunner

Automatic Control Laboratory, Swiss Federal Institute of Technology (ETH)
8092 Zürich, Switzerland

Abstract

In this paper, we present an input-output interpretation of supervision of discrete-event systems by exploiting and reinterpreting known results from the literature. We show that synthesis issues can be recast into the framework of Ramadge and Wonham. However, if the interconnection of supervisor and plant is affected by delays, we must introduce the concept of delay insensitive language and of delay insensitive supervisors.

1. Introduction

Discrete Event Processes (DEP) model a large variety of practical systems, *e.g.* flexible manufacturing systems and computer networks. The behavior of these systems is naturally described by a record or *trace* of certain qualitative changes (or *events*) in the system, by ignoring micro-changes occurring between two events.

Ramadge and Wonham introduced a linguistic approach on control of DEP by considering the possible traces that can be executed by the system to be strings on an alphabet of symbols representing the events. The set of all such strings is then called a *language*, and represents the possible behavior of the system. We denote the alphabet of symbols by Σ . The quantity Σ^* stands for the set of all finite sequences (or strings) over Σ . For a language $L \subset \Sigma^*$, \bar{L} denotes the set of prefixes of strings in L . We say a language L is *prefix-closed* if $L = \bar{L}$. $\text{prj}[\Sigma'](s)$ is the string obtained by removing from s all symbols not in the alphabet Σ' . $\text{prj}[\Sigma'](L)$ is the obvious extension to languages. Its converse, for an implied Σ is $\text{prj}^{-1}[\Sigma'](L') = \sup\{L \subseteq \Sigma^* \mid \text{prj}[\Sigma'](L) = L'\}$.

1.1. Process Model and Process Composition

A process is a triple $P = (\Sigma, L_P, M_P)$ composed of two languages L_P and M_P of finite traces over the alphabet Σ . L_P is prefix-closed and represents all partial traces of P , while $M_P \subseteq L_P$, is a set of distinguished traces, the *marked language*. Often M_P marks the set of successfully *completed* traces.

An important operator on processes is the *prioritized synchronous composition* [1]. First, we need some additional notation. The *generalized projection* of a string s on a *closed language* L , denoted by $\text{gprj}[L](s)$, is a string

defined recursively by $\text{gprj}[L](\epsilon) = \epsilon$ and with $\sigma \in \Sigma$:

$$\text{gprj}[L](s.\sigma) = \begin{cases} \text{gprj}[L](s).\sigma & \text{if } \text{gprj}[L](s).\sigma \in L \\ \text{gprj}[L](s) & \text{otherwise.} \end{cases}$$

$\text{gprj}[L](K)$ is the obvious extension to a language K . Note that for an arbitrary alphabet Σ' , $\text{prj}[\Sigma'](K) = \text{gprj}[(\Sigma')^*](K)$. We are ready to define our operator.

Definition 1: *The prioritized synchronous composition $P_1 \parallel_B P_2$ of two processes $P_1 = (\Sigma_1, L_{P_1}, M_{P_1})$ and $P_2 = (\Sigma_2, L_{P_2}, M_{P_2})$ with respect to the sets $A \subseteq \Sigma_1$ and $B \subseteq \Sigma_2$, is defined by*

$$P_1 \parallel_B P_2 = (\Sigma_1 \cup \Sigma_2, L', M') \quad (1)$$

where L' is defined by $\epsilon \in L'$ and by the recursive condition that for $s \in L'$, $s.\sigma \in L'$ if and only if

$$\begin{array}{ll} \text{gprj}[L_{P_1}](s).\sigma \in L_{P_1} \wedge \text{gprj}[L_{P_2}](s).\sigma \in L_{P_2} & \forall \sigma \in A \cap B \\ \text{gprj}[L_{P_1}](s).\sigma \in L_{P_1} & \forall \sigma \in A - B \\ \text{gprj}[L_{P_2}](s).\sigma \in L_{P_2} & \forall \sigma \in B - A \\ \text{gprj}[L_{P_1}](s).\sigma \in L_{P_1} \vee \text{gprj}[L_{P_2}](s).\sigma \in L_{P_2} & \forall \sigma \notin B \cup A. \end{array}$$

M' is described by the set of strings $s \in L'$ such that

$$\text{gprj}[L_{P_1}](s) \in M_{P_1} \wedge \text{gprj}[L_{P_2}](s) \in M_{P_2}.$$

The sets A and B are called the priority sets for process P_1 and P_2 respectively. They indicate the events that can occur in the composition only if the respective process agrees on their execution. The marked language M' of the composition is the set of strings in L' that are marked by both P_1 and P_2 . The special case of $A = \Sigma_1$ and $B = \Sigma_2$ is called the *full synchronous composition*, and is denoted by $P_1 \parallel P_2$. In this case, the processes must always execute simultaneously a common event (in $\Sigma_1 \cap \Sigma_2$), while the other events can happen independently in each respective process. The full synchronous composition is commutative and associative and therefore the full synchronous composition of n processes, denoted by $\parallel_{i=1}^n P_i$, can be defined.

1.2. Supervisory Control Theory

The basic problem of supervisory control introduced by Ramadge and Wonham consists in modifying the open-loop behavior of a plant by eliminating traces from the plant behavior. For a plant process $P = (\Sigma, L_P, M_P)$,

this is achieved by full synchronization of the plant process with another process $S = (\Sigma, L_S, M_S)$ called supervisor. In the synthesis procedure, we are free to choose the supervisor while the plant process is given and cannot be altered. The composition of the plant P with a supervisor S is then $P||S = (\Sigma, L_P^c, M_P^c)$. L_P^c is called the closed-loop language, M_P^c the marked closed-loop language and $P||S$ the *supervised* plant process.

Ramadge and Wonham postulate that the event set Σ of the plant is partitioned into *controllable* events Σ_c and *uncontrollable* events Σ_u . The controllable events can be prevented by synchronization from occurring in the plant, while the uncontrollable events are the remaining events over which we have no control authority. Thus, we are only interested in the class of supervisors that would never inhibit by synchronization uncontrollable events. This is expressed by the condition

$$P||S = P_{\Sigma}||_{\Sigma_c}S \quad (2)$$

that a supervisor has to satisfy for a particular plant P . In addition, the class of supervisors is further restricted to those also guaranteeing termination of a marked string. For this, Ramadge and Wonham [2] introduced the concept of non-blockingness. A process P is non-blocking if

$$L_P = \overline{M_P}. \quad (3)$$

We are now ready to present the standard supervisory control problem [3].

Supervisory Control Problem

Given a plant P , and a specification language L_{spec} for the closed-loop behavior, find a supervisor S such that

1. $\emptyset \subset M_P^c \subseteq L_{\text{spec}}$,
2. $P||S$ is non-blocking,
3. $P||S = P_{\Sigma}||_{\Sigma_c}S$,

In particular we note that if $P||S$ is non-blocking, then any string in L_P^c can be extended to a string marked both in the plant and in the supervisor.

Controllability: Ramadge and Wonham [3] introduced the notion of *controllability* to characterize the languages of supervisors solving the supervisory control problem [3]. A language $K \subset \Sigma^*$ is called *controllable* if the inclusion $\overline{K}\Sigma_u \cap L_P \subseteq \overline{K}$ holds. The class of controllable sublanguages of a language L is denoted by

$$\mathcal{C}(L) = \{K : K \subseteq L \wedge K \text{ is controllable}\}.$$

As the class of controllable languages is closed under language union, there exists a *supremal* element of $\mathcal{C}(L)$, denoted by $\text{sup } \mathcal{C}(L)$. It is shown in [3] that the supervisory control problem has a solution if and only if the language $\text{sup } \mathcal{C}(M_P \cap L_{\text{spec}})$ is non-empty. The particular supervisor

$$S_{\text{sup}} = (\Sigma, \overline{\text{sup } \mathcal{C}(M_P \cap L_{\text{spec}})}, \text{sup } \mathcal{C}(M_P \cap L_{\text{spec}})) \quad (4)$$

is always a solution if and only if $\text{sup } \mathcal{C}(M_P \cap L_{\text{spec}})$ is non-empty. It is also the least restrictive supervisor in the sense that it does not prevent any trace that is al-

lowed by any other supervisor meeting the specification.

Modular Plant: The plant is usually composed of modular subsystems. Each plant subsystem $1 \leq i \leq n$ can be modeled by a process $P_i = (\Sigma_i, L_i, M_i)$, and the global plant is $P = \parallel_{i=1}^n P_i$. If the languages of the plant's components are represented by automata, the number of states in the global plant's automaton increases exponentially with n , the number of modular components.

Modular Specification: A typical specification is given as a collection of languages. Clearly, these can be intersected to yield a global specification language; however, this procedure is subject to state-space explosion just as the composition of modular plants.

Fortunately, in the case of modular specifications, the principles of modular synthesis [4] can be applied. For this, Ramadge and Wonham introduced the notion of *non-conflicting* languages. Two languages L_1 and L_2 are *non-conflicting* if $\overline{L_1} \cap L_2 = \overline{L_1} \cap L_2$. Consider then two specification languages L_{spec_1} and L_{spec_2} , and two supervisors each solving the supervisory control problem for one of the two specification languages. The full synchronous composition of the two modular supervisors is solution of the supervisory control problem for the specification language $L_{\text{spec}} = L_{\text{spec}_1} \cap L_{\text{spec}_2}$ if the marked languages of the two modular supervisors are non-conflicting.

2. An Input-Output Semantics

Plant Model

The model interpretation proposed in the original Ramadge and Wonham framework consists of a plant which is an event "generator", "wildly" producing events; the only way to affect the behavior of the plant is by enabling or disabling the controllable events. In their semantics, the plant alone *schedules* the occurrence of both controllable and uncontrollable events.

In our opinion the "generator" model is not appropriate for most real systems. In real systems, events often do not occur spontaneously, but only as *responses to commands*. Commands typically control system actuators, while responses report readings from sensors.

The partition of Σ into Σ_c and Σ_u is interpreted differently. As proposed by Balemi in [5], the set Σ_c models the inputs of the plant whereas the set Σ_u stands for the plant outputs. From now on we refer to the inputs as *commands* and to the outputs as *responses*.

Supervisor Model

In the original model, the supervisor acts as a device that restricts the behavior of the plant by dynamically disabling/enabling the controllable events (see figure 1). In the input-output model, the supervisor does not simply prevent controllable events from occurring (by means of the synchronization $S||P$) but actually triggers

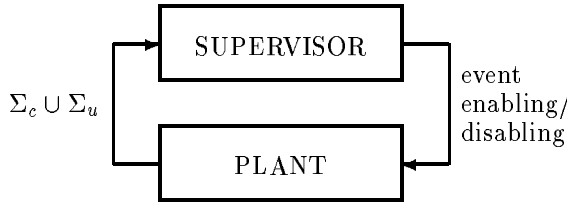


Figure 1: Generator plant with supervisor disabling controllable events

or *forces*¹ commands to the input of the plant. The generation of events is therefore initiated not only by the plant, but by both plant and supervisor. Commands are produced by the supervisor, and responses by the plant (see figure 2).

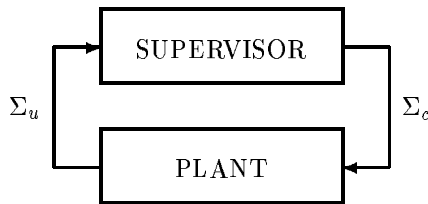


Figure 2: Symmetric feedback loop of input-output plant and supervisor

Equation (2) can be understood as the condition under which the connection of plant and supervisor yields the same closed-loop behavior for the events in Σ_u either subject or not subject to synchronization with the supervisor. Then, the input-output plant can arbitrarily force the unbuffered communication of responses (the events in Σ_u) from the output of the plant toward the input of the supervisor, and the supervisor will always be able to accept the responses sent.

The input-output view of the connection of plant and supervisor leads to the similar assumption that the plant cannot prevent the occurrence of commands produced by the supervisor and that any command from the supervisor must be accepted by the plant. A supervisor satisfies this additional assumption if the condition

$$P||S = P_{\Sigma_u}||_{\Sigma}S \quad (5)$$

holds. Equation (5) can be seen as a *dual* to equation (2). It is easily seen that conditions (2) and (5) can be combined into

$$P||S = P_{\Sigma_u}||_{\Sigma_c}S. \quad (6)$$

Σ_u and Σ_c are the disjoint *priority sets* and at the same time the *outputs* of the plant process P and of the supervisor process S respectively. We say, the composition of a supervisor and a plant satisfying equation (6) is *well-posed* [5]. This is in analogy to the concept of well-posedness of the interconnection of linear systems

¹This notion of forcing events is different from the one presented in [6].

in control theory, where well-posedness means that there exists a state space description of the interconnection, and therefore that the interconnection “makes sense”.

We are now ready to present the input-output supervisory control problem which incorporates the additional condition (5).

Input-Output Supervisory Control Problem — Given a plant P and a specification language $L'_{\text{spec}} \subseteq \Sigma_u^*$ for the closed-loop behavior, find a supervisor S such that

1. $\emptyset \subset \text{prj}[\Sigma_u](M_P^c) \subseteq L'_{\text{spec}}$,
2. $S||P$ is non-blocking,
3. the composition of S and P is well-posed.

Remark that here we restrict our attention to local specifications given on the responses Σ_u only. This may seem unnatural in a standard view, however it is in our opinion not a relevant restriction in an input-output semantics. In fact, commands represent the request of some desired changes, and therefore no constraint should be imposed on simple requests. In order to model *e.g.* the limited availability of some resource, additional responses can be embedded in the language of the plant at modeling time. Moreover, the restriction to this class of specifications will be of importance later in dealing with communication delays.

Theorem 1: *The input-output supervisory control problem has a solution if and only if the language $\text{prj}[\Sigma_u](\text{sup } \mathcal{C}(M_P \cap \text{prj}^{-1}[\Sigma_u](L'_{\text{spec}})))$ is non-empty [5].*

Then, if a solution exists, the supervisor of (4) with $L_{\text{spec}} = \text{prj}^{-1}[\Sigma_u](L'_{\text{spec}})$ is also a solution of the input-output supervisory control problem.

In particular, if the condition of existence of a solution is satisfied, any supervisor S solving the (standard) supervisory control problem for P and satisfying $L_S \subseteq \overline{M_P}$ (and therefore the additional condition (5)) is also a solution of the input-output supervisory control problem. If S solves the (standard) supervisory control problem but $L_S \not\subseteq \overline{M_P}$, the modified supervisor $S' = S||P$ is a solution of the input-output supervisory control problem.

3. Supervisor-Plant Connection with Delays

In general, the connection between the plant and the supervisor is affected by delays. Transmission delays affect the communication between supervisor and plant in both directions, and the time needed for the commands to be executed can also be regarded as delay for the responses.

The supervisor and the plant can be thought of as connected by two communications channels that buffer the messages sent by one process to the other (see figure 3). In the sequel, we make the assumption (possibly enforced by a communication protocol acting on the supervisor output) that, while many responses can be on

the way to the supervisor, at most one command can be in the channel from the supervisor to the plant ².

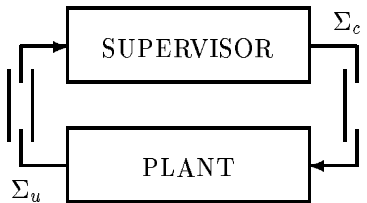


Figure 3: Supervisor-plant connection with delays

Consider a plant process represented by an automaton. Because of the delays, the plant automaton could be, for a given string executed by the supervisor, in another state than the one expected under the assumption of delay-free connection. Let us suppose for instance that the supervisor has sent a command σ_c to the plant, while a response σ_u already produced by the plant has not yet reached the supervisor. Then, the supervisor should have sent the command only if the plant can accept the command and the response both in the order $\sigma_u.\sigma_c$ and $\sigma_c.\sigma_u$, while the supervisor is able to accept them in the order $\sigma_c.\sigma_u$. In more general terms, we must extend for the sequel the notion of well-posedness in order to characterize the supervisor-plant connections with delays. A connection with communication delays is then well-posed if any event sent by one process reaching the input of the other process can always be accepted.

The strings processed by the supervisor and the plant at a given time are not always the same. We are therefore interested in supervisors whose behavior does not depend on possible permutation of commands and responses in the plant relative to the supervisor. Moreover, a marking in the supervisor should always (without new events processed by the supervisor) eventually correspond to a marking in the plant.

Supervisory Control Problem with Delays

Given a plant P and a specification language $L'_{\text{spec}} \subseteq \Sigma_u^*$, find a supervisor S such that in a closed loop with communications delays

1. $\emptyset \subset \text{prj}[\Sigma_u](M_P^c) \subseteq L'_{\text{spec}}$,
2. S is non-blocking in the closed-loop and a marking in S eventually corresponds to a marking in P .
3. the composition of S and P is well-posed.
4. the behavior of S is unaffected by permutation of order of commands and responses in P .

Condition 2. is stricter than the similar condition for the supervisory control problems presented before. In fact, we require that no additional response may be received by the supervisor in order to guarantee that for the marked string in the supervisor, a marked string in the plant will be eventually reached. Condition 4. is new, as without delays, the language of the supervisor

²In this case the logical behavior of the plant and of the supervisor individually is the same as in the case of delay-free communication of the commands (while responses are affected by delays).

and of the plant are identical at all times and no permutations can occur. We say, a supervisor satisfying condition 2., 3. and 4. of the supervisory control problem with delays is *delay insensitive*. We introduce now the definition of a delay insensitive language.

Definition 2: A language $L \subseteq \Sigma^*$ is delay insensitive if for $s' \in L$ the condition $s'\Sigma_u \cap \bar{L} = \emptyset$ holds and for $s \in \bar{L}$, $\sigma_c \in \Sigma_c$, $\sigma_u, \sigma'_u \in \Sigma_u$

$$s.\sigma_c, s.\sigma_u \in \bar{L} \Rightarrow s.\sigma_u.\sigma_c, s.\sigma_c.\sigma_u \in \bar{L}.$$

Suppose that also $s.\sigma_c.\sigma_u.t \in \bar{L}$ for some $t \in \Sigma^*$, then

$$\begin{aligned} s.\sigma_c.\sigma_u.t \in \bar{L} &\Rightarrow s.\sigma_u.\sigma_c.t \in \bar{L}; \\ s.\sigma_c.\sigma_u.t \in L &\Rightarrow s.\sigma_u.\sigma_c.t \in L; \\ s.\sigma_u.\sigma_c.t \in \bar{L} &\Rightarrow s.\sigma_c.\sigma_u.t \in \bar{L} \end{aligned}$$

The following theorem states that delay insensitive languages are just the languages of delay insensitive supervisors.

Theorem 2: Let $K \subseteq \Sigma^*$. There exists a delay insensitive supervisor S for P such that $M_P^c = K$ if and only if K is a controllable and delay insensitive language and K is M_P -closed, i.e. $K = M_P \cap K$.

By theorem 2, the supervisory control problem with delays is solvable only if there exists a controllable, delay insensitive and M_P -closed language K such that $\emptyset \subset \text{prj}[\Sigma_u](K) \subseteq L'_{\text{spec}}$. Given a language L , we denote the set of delay insensitive sublanguages of L by

$$\mathcal{D}(L) = \{K : K \subseteq L \wedge K \text{ is delay insensitive}\}$$

and the set of delay insensitive and controllable sublanguages of L by $\mathcal{CD}(L) = \mathcal{C}(L) \cap \mathcal{D}(L)$.

We note that while the class of delay insensitive languages is closed neither under union nor intersection, the class of delay insensitive and controllable sublanguages is closed under union. We state that in the following proposition.

Proposition 1: The class $\mathcal{CD}(L)$ of delay insensitive and controllable sublanguages of a given language L is closed under union; moreover there exists a supremal element denoted by $\text{sup } \mathcal{CD}(L)$.

For the proof of this proposition, we refer the reader to [7]. Then we can state the conditions for solvability of the supervisory control problem with delays.

Theorem 3: The supervisory control problem with delays is solvable if and only if

$$\emptyset \subset \text{prj}[\Sigma_u](\text{sup } \mathcal{CD}(M_P \cap \text{prj}^{-1}[\Sigma_u](L'_{\text{spec}})))$$

Therefore, if the supervisory control problem with delays is solvable, the supervisor $S = (\Sigma, \bar{K}_{\mathcal{D}}, K_{\mathcal{D}})$ with

$$K_{\mathcal{D}} := \text{sup } \mathcal{CD}(M_P \cap \text{prj}^{-1}[\Sigma_u](L'_{\text{spec}})) \quad (7)$$

is the least restrictive solution of the supervisory control problem with delays. The concept of a delay insensitive language is important also for the plants as shown in the following theorem.

Theorem 4: Given a delay insensitive language M_P and specification language $L'_{\text{spec}} \subset \Sigma_u^*$ the language

$$K := \sup \mathcal{C}(M_P \cap \text{prj}^{-1}[\Sigma_u](L'_{\text{spec}}))$$

is also delay insensitive.

We refer again the reader to [7] for the proof. This means that a supervisor $S = (\Sigma, \overline{K}, K)$ is a solution of the supervisory control problem with delays, and therefore that the synthesis of a supervisor can be performed exactly as in the case of delay-free communications.

3.1. Modular plant

In the sequel, we denote by $\|_i M_{P_i}$, with a small abuse of notation, the marked language of the composition $\|_i P_i$. For the case of modular plants, the following theorem provides a useful result.

Proposition 2: The composition $\|_i P_i$ for $i \leq i \leq n$ of n processes P_i with delay insensitive languages M_{P_i} has a delay insensitive marked language $\|_i M_{P_i}$.

This, together with theorem 4, allows to state directly the next theorem.

Theorem 5: Given n modular subplants with delay insensitive languages M_{P_i} , $1 \leq i \leq n$, and a specification language $L'_{\text{spec}} \subseteq \Sigma_u^*$, the language

$$K := \sup \mathcal{C}(\left(\|_i M_{P_i}\right) \cap \text{prj}^{-1}[\Sigma_u](L'_{\text{spec}}))$$

is delay insensitive.

The meaning of this theorem is that for several subsystems with delay insensitive languages we can compute the global supervisor disregarding possible delays and we automatically obtain a delay insensitive supervisor.

3.2. Modular specification

Modular supervision with delays in communication is made possible by the next results.

Corollary 1: The class $\mathcal{CD}(L)$ of delay insensitive and controllable sublanguages of a given language L is closed under intersection.

Then, the results of corollary 1 and theorem 5 together lead to the last result we present in this paper.

Theorem 6: Given m specification languages $L_{\text{spec}_j} \subset \Sigma_u^*$ for $1 \leq j \leq m$ and n plants with delay insensitive languages M_{P_i} for $1 \leq i \leq n$, compute the languages

$$K_j := \sup \mathcal{C}(\left(\|_i M_{P_i}\right) \cap \text{prj}^{-1}[\Sigma_u](L_{\text{spec}_j})).$$

Consider the supervisory control problem with delays for the specification $\bigcap_j L_{\text{spec}_j}$. If a solution exists and the m languages K_j are mutually non-conflicting, $\|_j S_j$ with $S_j = (\Sigma, \overline{K_j}, K_j)$ is the least restrictive solution.

Therefore, the modular computation of delay insensitive supervisors can be done modularly for modular specifications disregarding delays if the languages of the plants or of its subplants are delay insensitive and conditions of non-conflictingness and existence of a solution are met.

4. Conclusion

In this paper, an input-output perspective on supervisory control has been presented. The plant produces responses in reaction to commands, and symmetrically, the supervisor accepts the responses of the plant as inputs and produces commands for the plant. Extension of the supervisor control problem from the framework of Ramadge and Wonham to include this new perspective was straightforward.

The supervision of systems affected by delays in the communication between the supervisor and the plant required the concept of a delay insensitive supervisor and of a delay insensitive language. The class of delay insensitive and controllable languages of a given language is closed under union. Also, the full synchronous composition of processes with delay insensitive languages yields a process with delay insensitive language.

This allowed to state that some supervision problems can be addressed assuming delay-free communications if the plant process has a delay insensitive language. Then, the supervisor synthesized assuming no delays solves the equivalent problem with delays.

Acknowledgment

The authors gratefully acknowledge the valuable comments of Y. Brave and G.J. Hoffmann.

References

- [1] M. Heymann, "Concurrency and discrete event control", *IEEE Control System Magazine*, vol. 10, pp. 103-112, June 1990.
- [2] W.M. Wonham and P.J. Ramadge, "On the supremal controllable sublanguage of a given language", *SIAM J. Control Optim.*, vol. 25, pp. 637-659, May 1987.
- [3] P.J. Ramadge and W.M. Wonham, "Supervisory control of a class of discrete event processes", *SIAM J. Control Optim.*, vol. 25, pp. 206-230, Jan. 1987.
- [4] W.M. Wonham and P.J. Ramadge, "Modular supervisory control of discrete event systems", *Mathematics of Control Signals and Systems*, vol. 1, pp. 13-30, 1988.
- [5] S. Balemi, "A setup for real discrete event system control", Technical Report # 91.07, Automatic Control Laboratory, Swiss Federal Institute of Technology (ETH), Zürich, May 1991.
- [6] C.H. Golaszewski and P.J. Ramadge, "Control of discrete event systems with forced events", in *Proc. of 26th Conf. Decision and Control*, pp. 247-251, Los Angeles, CA, USA, Dec. 1987.
- [7] S. Balemi, "DES-control of a rapid thermal multi-processor", Technical Report # 91.12, Automatic Control Laboratory, Swiss Federal Institute of Technology (ETH), Zürich, Switzerland, Aug. 1991.