Distributive Control of Logical Discrete Event Systems Using Control Objectives

Spathopoulos, Michael P.; Smedinga, Rein; Ridder, Mark A. de

Published in:
proceedings of the European Control Conference

Publication date:
1995

DISTRIBUTIVE CONTROL OF LOGICAL DISCRETE EVENT SYSTEMS USING CONTROL OBJECTIVES* †

Michael P. Spathopoulos
division of dynamics and control
University of Strathclyde
Glasgow, Scotland
email: **mps@mecheng.strath.ac.uk**

Rein Smedinga
department of computing science
University of Groningen
Groningen, the Netherlands
email: **rein@cs.rug.nl**

Mark A. de Ridder
Public Health Research Unit
University of Glasgow
Scotland

Keywords: Logical discrete event systems, control objectives, predicates, distributive control, deadlock

Abstract

The representation of the desired behaviour of a logical discrete-event system (DES), using predicates and associated sets of blocking events, is introduced. The control of a regular DES consists in blocking events when a predicate concerning the system behaviour becomes true. If a blocking event is uncontrollable an algorithm is given which transforms the associated "uncontrollable" predicates into "controllable" ones. It is shown that using this technique it is possible to derive a non-regular desired behaviour which is least restrictive.

1 Introduction

A simple but effective way of defining a logical discrete event system is by giving its alphabet set and its behaviour set, so:

$$P = \langle \mathbf{a}P, \mathbf{b}P \rangle$$

denotes a DES with possible events collected in the finite set $\mathbf{a}P$ and with possible behaviour collected in the set $\mathbf{b}P \subseteq (\mathbf{a}P)^*$. $\mathbf{b}P$ is a set of strings. For simplicity we suppose that $\mathbf{b}P$ is prefix closed (i.e., if $st \in \mathbf{b}P$ then also $s \in \mathbf{b}P$, where st denotes concatenation of s and t). DESs defined in this way can be displayed using state graphs. The number of states in such a graph is finite if the language defined by $\mathbf{b}P$ is regular.

The above definition of a DES can be extended in different ways. For example, it is possible to add a second set of strings, called the task set and denoting all completed tasks of the system, see [7, 8]. Another possibility is to define a DES as a finite recursive process, which means that next to $\mathbf{b}P$ a marking set is introduced, which gives additional information for each string in the behaviour, see [3, 9].

Controlling a DES can be done in different ways. One way is to introduce a second system, called the supervisor, that follows the system and blocks events, if necessary, in order to get desired behaviour given in terms of a finite automaton. This is the Ramadge/Wonham approach, see [4, 5]. Alternatively, the second system can be a controller in the sense that plant and controller lead to some form of desired behaviour in synchronized cooperation, see [7].

In this paper a new way of describing the desired behaviour and, consequently, of controlling the plant is introduced. The desired behaviour is given in terms of some predicates on the system behaviour together with some associated blocking events. The pairing of the predicates and blocking events, called "control objectives" takes the form "if some condition on the system behaviour becomes true, then block some events." In the real world, most of the desired behaviours can be expressed in this way: block a certain door if an animal enters a certain room, close a valve if the pressure becomes too low, etc.

Intuitively, we can think of a DES as a black box, observed by an observer from a distance. The observer is unaware of the model of the system he is observing, he only sees events happen. The observer writes down on a piece of paper each event that occurs. Because he is unable to write down more than one event symbol at a time, events occurring in parallel will be written down on paper in some (arbitrary) order. This observation leads to a growing string of symbols on that piece of paper, representing possible behaviour of the system.

Our view of controlling simply means inspecting that written string, each time a new event happens (e.g., is written

*This work is partly supported by the Dutch Foundation for Scientific Research (NWO) and the British Council.
†In: Proceedings of the third European Control Conference (ECC) 1995, september 5-8, 1995, Roma, Italy, vol.1 of 4, pages 383–388

down) and, depending on some condition, block some events. Although the technique developed in this paper holds for arbitrary observed behaviour, it should be clear by now that the observer has to check only one specific string of events, namely the string of events that actually occurs. Thus, the controller is a collection of control objectives which are evaluated online for the current string only (pathwise). Depending on the evaluated conditions on this string only, some events should be blocked. If a next event occurs, all blocking is reset, conditions are re-evaluated and a (possibly different) set of events is blocked. Because of the resetting of the blocking, this way of controlling is said to be *local*. In general, in order to evaluate the conditions, there is no need to know the whole string, e.g., if the condition depends on the number of certain events that have occurred, only this number needs to be remembered by the observer. Nevertheless, this way is particularly effective in describing "dynamic" ("time varying") desired behaviours and gives rise to "dynamic" and "non-Markovian" (not state dependent) control laws, see [10]. If some events are uncontrollable and thus cannot be blocked the conditions describing the desired behaviour become uncontrollable and their transformation to controllable ones is required. This is obtained by backpropagating the conditions by means of an off-line algorithm.

The controlled system involves inserting the collection of control objectives (controller) into the plant. If, for example, a door needs to be blocked, if a condition becomes true, then we only have to evaluate this condition just before opening the door. So, the corresponding objective can be placed into the plant before the event "door." Effectively, the controller is distributed within the plant and this design technique is called distibutive control. This becomes particularly clear when modular models of finitely recursive processes are considered, see [2, 10]. It should be noticed that the control objectives are defined on the whole system behaviour, i.e., in distributive control the specifications are *global*.

1.1 Notation

First we introduce the set notation used in this paper. In general the notation

$$\{x : B(x) : f(x)\}$$

is the set of all elements $f(x)$, constructed using some x for which $B(x)$ returns true. x and $f(x)$ may be vectors, i.e., multi-dimensional elements. For example:

$$\{n, m : 1 \leq n < 5 \ \wedge \ 1 \leq m \leq 2 : n \times m\}$$

is the integer set $\{1, 2, 3, 4, 6, 8\}$.

Based on the same syntax,

$$(\forall x : B(x) : P(x))$$

evaluates to true if for all x that satisfy $B(x)$ the condition $P(x)$ is true. For example

$$(\forall x : x > -12 : x^2 \geq 0) = true$$

Similarly,

$$(\exists x : B(x) : P(x))$$

is true if for at least one x satisfying $B(x)$ the condition $P(x)$ becomes true.

2 Control objectives

First the control objectives are defined and a definition of the plant under these objectives is given.

Definition 1 *A control objective for a process $P = \langle \mathbf{a}P, \mathbf{b}P \rangle$ is defined by*

$$\Theta(P) = [B, A]$$

with

$$\begin{array}{ll} B\colon (\mathbf{a}P)^* \rightarrow \{true, false\} & \textit{the blocking condition} \\ A \subseteq \mathbf{a}P \cup \{\epsilon\} & \textit{the blocking event set} \end{array}$$

□

A control objective $\Theta(P)$ blocks the events in A if, for any behaviour $s \in \mathbf{b}P$, the condition $B(s)$ returns true.
In the set A the symbol ϵ is also present. ϵ is the trigger event which starts the system. Blocking ϵ means "do not start the system." This symbol is added in order to handle the backpropagation in a proper way. We will return to the backpropagation in the next section.

Notice that B is a predicate on the language $\mathbf{b}P$. It is not a language itself, nor a predicate on the states of the corresponding state graph of P. This form of B allows the controlling of a regular plant (P can be represented by a finite state graph) so that the desired controlled behaviour becomes non-regular.

Definition 2 *A system $P = \langle \mathbf{a}P, \mathbf{b}P \rangle$ under the control objective $\Theta(P) = [B, A]$ is defined to be the system $P_{\Theta(P)}$ given by:*

$$P_{\Theta(P)} = \begin{cases} \langle \mathbf{a}P, \emptyset \rangle & \textit{if } \epsilon \in A \ \wedge \ B(\epsilon) \\ \langle \mathbf{a}P, \{t : t \in \mathbf{b}P \ \wedge \ t \, \mathbf{sat} \, \Theta(P) : t\} \rangle & \textit{otherwise} \end{cases}$$

where:

$$t \, \mathbf{sat} [B, A] = (\forall s, u, b : t = sbu \ \wedge \ b \in \mathbf{a}P : B(s) \Rightarrow b \notin A)$$

A system under a set of control objectives $\{\Theta_1, \ldots, \Theta_n\}$, denoted by $P_{\{\Theta_1, \ldots, \Theta_n\}}$ is now defined by

$$(\cdots((P_{\Theta_1})_{\Theta_2} \ldots)_{\Theta_n}$$

□

From the behaviour of P, each trace at for which $sat \in \mathbf{b}P$, where s is such that $B(s)$ becomes true and $a \in A$, is removed. Also, we block the whole system, i.e., prevent the system from starting, if ϵ is in the blocking event set and the corresponding blocking condition is initially true.

Example 3 Suppose the system P is given as $\langle \{a,b,c\}, (a|b|c)^* \rangle$. Here, * is the Kleene star and $|$ denotes alternatives. It is clear that P is regular, i.e., can be represented by a finite state graph. Moreover, suppose, we want to control the system in such a way that first some a's occur, next event b occurs, and last the same many c's as a's occur, i.e., $a^n bc^n$ (for arbitrary n). Clearly, the desired behaviour is no longer regular. Nevertheless, three control objectives are sufficient in order to describe this desired behaviour:

$$\Theta_1 = [s\,\mathbf{N}\,b = 1, \{a,b\}]$$
$$\Theta_2 = [s\,\mathbf{N}\,b = 0, \{c\}]$$
$$\Theta_3 = [s\,\mathbf{N}\,a = s\,\mathbf{N}\,c, \{a,b,c\}]$$

where $s\,\mathbf{N}\,e$ denotes the number of occurrences of event e in trace s (with $\epsilon\,\mathbf{N}\,e = 0$ for each event e). It is clear that, in the controlled system, the event c is blocked as long as no b has happened and a and b are blocked as soon as b has happened. Finally, all events are blocked as soon as the desired behaviour has occurred. So we have:

$$P_{\{\Theta_1,\Theta_2,\Theta_3\}} = \langle \mathbf{a}P, \{n : n \geq 0 : a^n bc^n\} \rangle$$

□

Practical experience shows that it usually is possible to define desired behaviours in terms of control objectives. Moreover, desired behaviours are often stated using such objectives: "if this happens, block these events." Instead of translating these objectives to languages (or state graphs) our proposed method uses them directly.

3 Backpropagation of control objectives

An event can only be blocked if it is controllable. Some events occur spontaneously and cannot be blocked. Such events are called uncontrollable. In general, the blocking event set in a control objective does not contain controllable only events. In these terms, the main problem to solve is to transform control objectives with uncontrollable events into control objectives where only controllable events are present. Given a system P, we know the set of controllable events, denoted by $\mathbf{c}P$ and the set of uncontrollable events $\mathbf{e}P$, sometimes called exogenous events, see [6].

A control objective in which the blocking set contains only controllable events is called a *controllable control objective* (CCO). An *uncontrollable control objective* (UCO) is defined by $[B, \{a\}]$ where $a \in \mathbf{e}P$.

The only way to derive a CCO from an UCO is to backpropagate the blocking of the uncontrollable event in the plant, i.e., instead of blocking the desired uncontrollable event, we have to block one or more controllable events that occur earlier in the behaviour of the plant, such that the uncontrollable event cannot occur anymore.

For example, if the only possible behaviour is abc and the control objective $[B(s), \{c\}]$ results in $B(ab) = true$ then c should be blocked. If c is uncontrollable, we should block earlier, i.e., block b, if b is controllable. For this, we cannot use the same condition B, but should change it accordingly to B', such that $B'(a) = B(ab)$ returns true and b is blocked.

For simplicity, we suppose that the control objectives contain only one event in the blocking set. If not, we can easily change the objective with a set of events into a set of objectives with the same condition but with one event in the blocking set.

Definition 4 *For a control objective $\Theta(P) = [B, \{a\}]$ we define the set of one step backpropagated control objectives by*

$$\Theta^{\leftarrow}(P) = \{e : e \in \mathbf{before}(a) : [wp_e(B), e]\}$$

where

$$\mathbf{before}(a) = \{a' : a' \in \mathbf{a}P \ \wedge \ (\exists t,u : t,u \in (\mathbf{a}P)^* : ta'au \in \mathbf{b}P) : a'\} \cup \mathbf{init}(a)$$

with

$$\mathbf{init}(a) = \begin{cases} \{\epsilon\} & \text{if } a \in \mathbf{b}P \\ \emptyset & \text{otherwise} \end{cases}$$

$\mathbf{before}(a)$ *is the set of events that can occur just before the occurrence of the event a, it includes the trigger event in case the system can start from the event a, and*

$$wp_e(B)(s) = B(se)$$

is the backpropagated condition. □

Notice that, because we assume $\mathbf{b}P$ to be prefix closed, the string u in the definition of $\mathbf{before}(a)$ can be omitted. Also notice that the "event" ϵ, as introduced here, corresponds to an unlabelled arrow pointing the initial state of a state graph.[1] The set $\mathbf{before}(a)$ can be computed easily in the case P is regular and is represented by a finite state automaton. The following informally described algorithm can then be used:

1. construct the reverse graph of the automaton, i.e., reverse the directions of all arrows.
2. compute, in the reversed automaton, all events that can occur after the occurrence of the event a. This can easily be done in some recursive way and is a finite process because the automaton contains only finite number of states and finite number of transitions.
3. add the trigger event ϵ in case the original system starts from the event a.
4. the computed set of events is the **before**-set of the original graph.

Next we prove that backpropagating control objectives does not lead to unwanted behaviour:

[1] ϵ as event is the trigger event, ϵ also can denote a string: the empty string. This evident double meaning should not lead to confusion.

Property 5

$$\mathbf{b}P_{\Theta\leftarrow(P)} \subseteq \mathbf{b}P_{\Theta(P)}$$

proof: We give the proof only in the case the **before**-set does not contain the trigger event ϵ. If so, an additional condition should be taken into account which only complicates the expressions, but does not contribute to the proof.
If $\Theta(P) = [B, \{a\}]$ then:

$$\begin{array}{ll}
& \mathbf{b}P_{\Theta\leftarrow(P)} \\
= & \\
& \{t : t \in \mathbf{b}P \ \wedge \ t \,\mathbf{sat}\, \Theta^{\leftarrow}(P) : t\} \\
= & \\
& \{t : t \in \mathbf{b}P \ \wedge \ (\forall e : e \in \mathbf{before}(a) : t \,\mathbf{sat}\, [wp_e B, \{e\}]) \\
& \quad : t\} \\
= & \\
& \{t : t \in \mathbf{b}P \ \wedge \ (\forall e : e \in \mathbf{before}(a) \\
& \qquad\qquad : (\forall s,u,b : t = sbu \\
& \qquad\qquad\qquad : wp_e B(s) \Rightarrow b \neq e)) \\
& \quad : t\} \\
= & \\
& \{t : t \in \mathbf{b}P \ \wedge \ (\forall e : e \in \mathbf{before}(a) \\
& \qquad\qquad : (\forall s,u,b : t = sbu \\
& \qquad\qquad\qquad : B(se) \Rightarrow b \neq e)) \\
& \quad : t\} \\
\subseteq & \quad [\text{ see below }] \\
& \{t : t \in \mathbf{b}P \ \wedge \ (\forall s,u,b : t = sbu : B(s) \Rightarrow b \neq a) \\
& \quad : t\} \\
= & \\
& \mathbf{b}P_{\Theta(P)}
\end{array}$$

where we use that:

$$\begin{array}{ll}
& (\forall e : e \in \mathbf{before}(a) : (\forall s,u,b : t = sb : B(se) \Rightarrow b \neq e)) \\
\Rightarrow & \quad [\text{ take } s' = se, \text{ for some } e \in \mathbf{before}(a) \ ] \\
& (\forall s', u, b : t = s'bu : B(s') \Rightarrow b \neq a)
\end{array}$$

□

Therefore, the use of backpropagated control objectives results in obtaining less behaviour than when blocking the event itself, as initially required. However, this does not add behaviour that is already blocked due to the original objectives.

Next we give the following definition which characterizes the least restrictive behaviour with respect to the blocking of an uncontrollable event a.

Definition 6 *The blocked behaviour after u w.r.t. event a is defined by*

$$B_a(u) = \{t : t \in (\mathbf{a}P)^* \ \wedge \ uat \in \mathbf{b}P : uat\}$$

The least restrictive blocked behaviour after u w.r.t. the uncontrollable event a and for $b \in \mathbf{before}(a)$ *is defined by*

$$\begin{array}{l}
B^*_{a,b}(u) = \\
\qquad \{s,t : s,t \in (\mathbf{a}P)^* \ \wedge \ ub \in \mathbf{b}P \ \wedge \ ubat \in B_a(ub) \\
\qquad\quad : ubs\}
\end{array}$$

□

$B_a(u)$ characterizes the behaviour of the plant that is removed when, after behaviour u, event a is blocked. $B^*_{a,b}(u)$ is the behaviour which is removed when, after u, event b is blocked, and includes the blocked behaviour B_a i.e., $B_a(ub) \subseteq B^*_{a,b}(u)$. However, as stated in the following theorem, $B^*_{a,b}(u)$ does not remove any "unnecessary" behaviour in order to include $B_a(bu)$.

Theorem 7

$$\begin{array}{l}
(\forall u, a, b : uba \in \mathbf{b}P \\
\qquad : b \in \mathbf{before}(a) \Rightarrow B_a(ub) = B^*_{a,b}(u))
\end{array}$$

proof: follows from definition. □

Following theorem 7 we conclude that the backpropagation of uncontrollable control objectives does not block unnecessary events, but only these events that should be blocked in order to get the desired behaviour, i.e., backpropagation is least restrictive.

To recap, when an uncontrollable objective exists, we proceed as follows: we compute the backpropagated control objectives for the uncontrollable events, and repeat backpropagation until all objectives become controllable. However, proceeding in this way, may lead to infinite repetitions because first, $\mathbf{before}(a)$ may contain a again, and second, before the occurrence of an event a in the behaviour of P an uncontrollable loop (loop of uncontrollable events) may be present. We deal with both situations in the next sections.

3.1 Event precedes itself

If $a \in \mathbf{before}(a)$, we have the problem of infinite repetition of the same event a when the set $\mathbf{before}(a)$ is computed. However, this event does not contribute to the derivation of controllable control objectives. In order to avoid this problem, the following definition of **before** is used:

$$\begin{array}{l}
\mathbf{before}(a) = \\
\qquad \{a' : a' \in \mathbf{a}P \ \wedge \ (\exists t, u :: ta'au \in \mathbf{b}P) \ \wedge \ a' \neq a \\
\qquad\quad : a'\} \\
\qquad \cup \, \mathbf{init}(a)
\end{array}$$

3.2 Uncontrollable loops

In case, before a, an uncontrollable loop is present, no backpropagated blocking contribution can be expected from all the events in the loop. The only way to block the event a, is to make a block, as close as possible, before the loop.

Suppose $tbu^*av \in \mathbf{b}P$ and the event a together with all the events in u are uncontrollable, i.e., $u^*a \in (\mathbf{e}P)^*$. Blocking the event a is thus impossible. Computing $\mathbf{before}(a)$ leads to a set which includes event b and an uncontrollable event from u as well. The before-set contains an event from u each time we repeat the computation. However, the only way to prevent the event a to happen is to block b. Basically, we have to block b so that the system cannot get into the uncontrollable loop.

This informal discussion leads to the following definition:

Definition 8

$$\textbf{strictBefore}(a) = \textbf{before}(a) \setminus \textbf{beforeInLoop}(a)$$

where

$$\textbf{beforeInLoop}(a) = \{a' : a' \in \mathbf{e}P \ \wedge \ (\exists u, v, t : u, v \in (\mathbf{e}P)^* \ \wedge \ t \in (\mathbf{a}P)^* : t(ua'v)^* a \subseteq \mathbf{b}P) : a'\}$$

□

beforeInLoop(a) contains the uncontrollable events that may occur before the event a in an uncontrollable loop. Since we deal with regular systems, loops can always be detected by inspecting the corresponding state graphs. It is easy to find the set **strictBefore**(a) is such a graph. First delete all uncontrollable loops and then proceed as above to find the **before**-set.

4 Controlled behaviour

Suppose we have a system $P = \langle \mathbf{a}P, \mathbf{b}P \rangle$ and a control objective $\Theta(P) = [B, \{a\}]$ with $a \in \mathbf{e}P$. By backpropagating the uncontrollable objective we can control the system in order to satisfy the desired constraint. So we compute the controlled behaviour

$$\Theta^{\Leftarrow}(P) = \{e : e \in \textbf{strictBefore}(a) : [wp_e(B), e]\}$$

Clearly, following theorem 7, $P_{\Theta^{\Leftarrow}(P)}$ is least restrictive.

Moreover, if the system P is regular, the backpropagation of control objectives can be done in a finite number of steps. This follows directly from the fact that regular systems can be represented using finite automata and both deleting uncontrollable loops and finding **strictBefore**-events can be done using a finite number of steps. We should emphasize here, that for regular systems all possible traces are known (and represented using an automaton). If the systems are not regular, or if the system is not given in a language-form in which all traces are known, loops and **strictBefore**-events cannot always be computed using finite number of steps. In the last case, additional information about the plant, may make possible the computation in finite steps.

5 Observable events

If an event is observable, i.e., the event can be seen, we place the control objective, if needed, just before the occurrence of this event. If a door should be blocked, the corresponding objective is placed in the flow just before the event "door," such that the condition is computed before using the door and before the door is blocked. If an event is not observable, e.g., if the door is hidden behind some machinery, we have to place the control objective earlier in the flow. The solution here is to backpropagate the control objectives and to block the unobservable event from a distance. The backpropagating objective should be placed in front of the first observable event, before the unobservable one. However, we do not deal with this case in the present paper.

6 Deadlock

The existence of uncontrollable events in the plant and the effort to derive completed task behaviours can cause the controlled system to deadlock. Until now, when dealing with finite automata desired behaviours, the solution to the deadlock problem has been to recognize the deadlock states and remove them. Basically, removing the deadlock state is a static problem. However, in our setup, when nonregular desired behaviours are considered, the only solution to the problem is by considered *limited deadlock predictors*. The essence here is to predict the deadlock, since recognizing it, in an infinite state graph, is impossible. Let $P = \langle \mathbf{a}P, \mathbf{b}P, \mathbf{t}P \rangle$, where $\mathbf{t}P$ denotes completed task behaviour. To avoid deadlock, the observer, each time a new event has happened, should check the following condition, called deadlock condition:

$$D(s) = (\exists t : st \in \mathbf{b}P \ \wedge \ st \notin \mathbf{t}P : (\forall a :: sta \notin \mathbf{b}P))$$

If $D(s)$ is true, the observer should block every event leading to such a deadlock trace s. However, this condition is undecidable, so we define the following *limited deadlock condition*:

$$\overline{D}(s) = (\exists t : |t| < n \ \wedge \ st \in \mathbf{b}P \ \wedge \ st \notin \mathbf{t}P : (\forall a :: sta \notin \mathbf{b}P))$$

where, by $|t|$ we denote the length of the string t.

In case the desired behaviour is regular, the maximum number of n that will guarantee deadlock prediction, i.e., it will recognize deadlock, is given by the number of the states of the finite automaton of the controlled plant. When the desired behaviour is non-regular, the limited deadlock predictor may fail to predict a deadlock and this may lead to an "impossible" situation, i.e., the prediction of a deadlock will come too late in order to be avoided. However, in the real world, a reasonable length of n may successfully result in avoiding the possible deadlock.

This involves the following deadlock control objectives:

$$\{f : f \in \mathbf{a}P : [\overline{D}_f, f]\}$$

with

$$\overline{D}_f(s) = \overline{D}(sf)$$

which are added to the controllable control objectives in order to derive the *limited guaranteed* deadlock free controlled behaviour. The above notions become appropriate when "dynamic" (logical "time varying") plant and desired behaviours are introduced, see [2, 10].

7 Conclusion

We have developed a new technique of representing desired behaviours and, consequently, of controlling a logical DES. The specifications are given in terms of blocking conditions and associated blocking events, and these may arise from quasi-language specifications. The control design based on these specifications consists in transforming "uncontrollable" conditions to "controllable" ones. Using this technique makes it possible to derive non-regular desired behaviours using finite algorithms.

References

[1] S. Balemi, P. Kozák, and R. Smedinga, editors. *Discrete Event Systems: Modeling and Control*, volume 13 of *Progress in Systems and Control Theory*. Birkhäuser Verlag, Basel, Switzerland, 1993. (Proceedings of the Joint Workshop on Discrete Event Systems (WODES'92), August 26–28, 1992, Prague, Czechoslovakia).

[2] M.A. de Ridder and M.P. Spathopoulos. Hierarchical modelling and distributive control systems design of a flexible manufacturing system. In *Proceedings of 3rd IEEE conference in Control Applications*, pages 1757–1763, Glasgow, 1994.

[3] K. Inan and P. Varaiya. Finitely recursive process models for discrete event systems. *IEEE transactions on automata and control*, 33, 1988.

[4] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM journal on Control and optimization*, 25 (1), 1987. See also: systems control group report 8515, department of electrical engeneering, University of Toronto.

[5] P.J. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1), 1989.

[6] R. Smedinga. *Control of discrete events*. PhD thesis, University of Groningen, 1989.

[7] R. Smedinga. *An Overview of Results in Discrete Event Systems using a Trace Theory Based Setting*, pages 43–56. Volume 13 of Balemi et al. [1], 1993. (Proceedings of the Joint Workshop on Discrete Event Systems (WODES'92), August 26–28, 1992, Prague, Czechoslovakia).

[8] R. Smedinga. Effective control of logical discrete event systems in a trace theory setting using the reflection operator. In Guy Cohen and Jean-Pierre Quadrat, editors, *11th International Conference on Analysis and optimalization of Systems, Discrete event systems, Sophia-Antipolis, June 15-17, 1994*, number 199 in Lecture notes in Control and Information sciences, pages 66–72. Springer Verlag, 1994.

[9] M.P. Spathopoulos and M.A. de Ridder. Some issues on the modelling and control of discrete event systems using an algebra of processes. In *IMA conference in control, Manchester*, 1992.

[10] M.P. Spathopoulos and M.A. de Ridder. Modelling and distributive control design of a subway system. Proceedings European Control Conference, 1995.