

INPUT/OUTPUT DISCRETE EVENT PROCESSES AND SYSTEM MODELING

Silvano Balemi*

Abstract. In this paper we present an input/output interpretation of supervisory control theory. As opposed to the model proposed by Ramadge and Wonham, the plant does not spontaneously generate all events as outputs, but reads some events, called commands, as inputs and produces other events, called responses, as outputs. Based on this input/output interpretation we propose a modeling framework that starts with a physical description of the system and ends with the system model. This is performed by designing software routines interacting with the system at appropriate time instances.

1. INTRODUCTION

The supervisory control theory introduced by Ramadge and Wonham [1] provides a framework for the treatment of theoretical and computational issues related to the control of discrete event systems. However, the interpretation and implementation of the results obtained in this research area are not always straightforward. This is partly due to the model semantics of a plant generating events with a supervisor only passively preventing the generation of some of the events.

This semantics is not appropriate for modeling many real life systems because it does not naturally allow the supervisor to generate events of its own. A partial step in the direction of a supervisor able to force the occurrence of certain events in the plant was done by Golaszewski and Ramadge [2]. In this paper we use an explicit input/output semantics for the plant. The plant reads inputs, called commands, and produces outputs, called responses. Symmetrically, a controller accepts as inputs the responses produced by the plant and generates as outputs the commands for the plant.

A crucial step toward the practical control of a discrete event system is the modeling step. As opposed to the modeling of continuous-valued systems however, the task of modeling a discrete event system is not a “passive” process in which one selects the best suitable model from a class of models according to some given criteria. A model for a discrete event system must not be “identified” but designed and constructed.

This is performed by adding to the given system some additional structure, for instance in form of some software that interacts with the (physical) system via actuators and sensors. Following the I/O semantics above, the inputs correspond to the call of software routines, while the outputs are messages reporting the qualitative changes occurred in the system.

*Automatic Control Laboratory, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland; e-mail: balemi@aut.ee.ethz.ch

In this paper we first introduce the input/output control problem [3, 4, 5]. Then we present a modeling procedure for obtaining an input/output model of the considered discrete event system. We illustrate this with the example of a valve of a semiconductor manufacturing piece of equipment [4, 5].

1.1 Preliminaries and Notation

The behavior of discrete event systems is naturally described by sequences (called *trace*) of certain qualitative changes (called *events*) in the system, by ignoring micro-changes occurring between two events. Ramadge and Wonham introduced a language-theoretical approach on control of DES by considering the possible traces 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 . The empty string is denoted by ϵ . For a language $L \subset \Sigma^*$, \overline{L} denotes the set of prefixes of strings in L . We say a language L is *prefix-closed* if $L = \overline{L}$.

1.2 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* [6]. Before defining it, we need additional notation. The *generalized projection* of a string s onto a prefix-closed language L , denoted by $\text{gproj}[L](s)$, is a string defined recursively by $\text{gproj}[L](\epsilon) = \epsilon$ and with $\sigma \in \Sigma$ by

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

The generalized projection of a string onto a language yields the string that is obtained from the original string by discarding in a row those letters that could not “follow” language L . The expression $\text{gproj}[L](K)$ denotes the obvious extension to a language K . For an arbitrary alphabet Σ' , the special case $\text{gproj}[(\Sigma')^*](K)$ is called the *natural projection of K on the alphabet Σ'* . We are now ready to define the composition operator.

Definition 1 (Prioritized Synchronous Composition) *The prioritized synchronous composition $P_{1A} \parallel_B P_2$ of $P_1 = (\Sigma_1, L_{P_1}, M_{P_1})$ 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*

$$P_{1A} \parallel_B P_2 = (\Sigma_1 \cup \Sigma_2, L_{P_{1A} \parallel_B P_2}, M_{P_{1A} \parallel_B P_2}) \quad (1)$$

where $L_{P_{1A} \parallel_B P_2}$ is defined by $\epsilon \in L_{P_{1A} \parallel_B P_2}$ and by the recursive condition that for $s \in L_{P_{1A} \parallel_B P_2}$ and $\sigma \in \Sigma_1 \cup \Sigma_2$ the string $s.\sigma$ is in $L_{P_{1A} \parallel_B P_2}$ only if the following

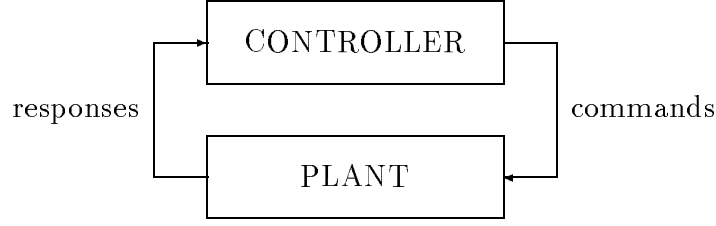


Figure 1: Closed-loop system

condition is satisfied:

$$\begin{array}{ll}
 \text{gproj}[L_{P_1}](s).\sigma \in L_{P_1} & \forall \sigma \in A - B \\
 \text{gproj}[L_{P_2}](s).\sigma \in L_{P_2} & \forall \sigma \in B - A \\
 \text{gproj}[L_{P_1}](s).\sigma \in L_{P_1} \wedge \text{gproj}[L_{P_2}](s).\sigma \in L_{P_2} & \forall \sigma \in A \cap B \\
 \text{gproj}[L_{P_1}](s).\sigma \in L_{P_1} \vee \text{gproj}[L_{P_2}](s).\sigma \in L_{P_2} & \forall \sigma \notin B \cup A.
 \end{array} \quad (2)$$

The marked language $M_{P_1 \parallel_B P_2}$ is described by

$$M_{P_1 \parallel_B P_2} = \left\{ s \in L_{P_1 \parallel_B P_2} \mid \text{gproj}[L_{P_1}](s) \in M_{P_1} \wedge \text{gproj}[L_{P_2}](s) \in M_{P_2} \right\}$$

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 events not in the priority set of a process can be executed by the other process at any time. The special case of $A = \Sigma_1$ and $B = \Sigma_2$ is called the *full synchronous composition*, and is denoted by $P_1 \parallel P_2$; the processes must execute simultaneously a common event (in $\Sigma_1 \cap \Sigma_2$), while the other events can happen independently in each respective process. The other special case of $A = B = \emptyset$ is called *parallel composition*; the processes must execute a common event if both can, otherwise they are free to execute any event independently. The parallel composition of two processes P_1 and P_2 is denoted by $P_1 \parallel P_2$.

2. AN INPUT/OUTPUT SEMANTICS

We introduce here an input/output model of discrete event systems. The alphabet of the process languages is divided into two disjoint subsets: the commands (denoted by Σ_c) and the responses (denoted by Σ_r). As proposed in [3], the set Σ_c of commands models the inputs of the plant process whereas the set Σ_r of responses stands for the plant outputs.

The basic problem we want to address is the enforcement of a behavior on the plant $P = (\Sigma, L_P, M_P)$. We do this with a process $C = (\Sigma, L_C, M_C)$ called controller. Symmetrically to the plant, the controller accepts responses as inputs and produces commands as outputs. Then, the closed-loop behavior of the plant and the controller connected together as in figure 1 can be described by the full synchronous composition of the two processes. The resulting process is then $C \parallel P = C \parallel_{\Sigma} P = (\Sigma, L_P \parallel_{\Sigma} L_C, M_P \parallel_{\Sigma} M_C)$.

For the closed-loop, we require that any response sent by the plant can be accepted by the controller. This is expressed by the condition

$$P\|C = P_{\Sigma}\|_{\Sigma_c}C. \quad (3)$$

Controllers satisfying this constraint can never inhibit by synchronization the occurrence of a response in the plant. This condition can be rephrased using the notion of controllability of a language [7]. A language $K \subseteq \Sigma^*$ is called *controllable with respect to L and Σ'* if the inclusion $\overline{K}.\Sigma' \cap \overline{L} \subseteq \overline{K}$ holds. Then it can be shown [8] that a controller satisfies equation (3) if and only if L_C is controllable w.r.t. L_P and Σ_r *i.e.* if and only if $L_C.\Sigma_r \cap L_P \subseteq L_C$ holds. Symmetrically to the previous condition, we also require in the closed-loop that any command sent by the controller can be accepted by the plant. Thus the controller does not need to check if the plant can accept the command but knows a priori that the plant can. This is expressed by the following condition

$$P\|C = P_{\Sigma_r}\|_{\Sigma}C. \quad (4)$$

As for equation (3), condition (4) holds if and only if L_P is controllable with respect to L_C and Σ_c , *i.e.* if and only if $L_P.\Sigma_c \cap L_C \subseteq L_P$ holds. The closed-loop connection of a controller and a plant satisfying the two conditions above is said to be *well-posed*. An output produced by any of the processes in the closed loop can be accepted as an input by the other process. This is in analogy to the concept of well-posedness in control theory, where it indicates that the connection of some systems “makes sense”.

In addition to the condition of well-posedness, we are interested in controllers that always guarantee the termination of a marked string. For this, Ramadge and Wonham [9] introduced the concept of non-blockingness. A process P is non-blocking if $L_P = \overline{M_P}$. We are now ready to present the input/output control problem.

Input/Output Control Problem

Given a plant P and a specification language $L_{\text{spec}}^{\min}, L_{\text{spec}}^{\max} \subseteq \Sigma^*$ for the closed-loop behavior, find a controller C such that

1. $L_{\text{spec}}^{\min} \subseteq M_P^{\circ} \subseteq L_{\text{spec}}^{\max}$,
2. $C\|P$ is non-blocking,
3. The connection of P and C is well-posed.

The solution of the control problem can be expressed using the notion of controllability. The class $\mathcal{C}(L)$ of sublanguages of a language L which are controllable with respect to L_P and Σ_r is closed under language union, and therefore there is a *supremale* element of $\mathcal{C}(L)$, denoted by $\sup \mathcal{C}(L)$. It can be shown [4, 5] that the input/output control problem has a solution if and only if $L_{\text{spec}}^{\min} \subseteq \sup \mathcal{C}(M_P \cap L_{\text{spec}}^{\max})$. If a solution exists, the controller

$$C_{\text{sup}} = (\Sigma, \overline{\sup \mathcal{C}(M_P \cap L_{\text{spec}}^{\max})}, \sup \mathcal{C}(M_P \cap L_{\text{spec}}^{\max})) \quad (5)$$

is always a solution.

3. MODELING PRINCIPLES

In this section we present a methodology for obtaining an input/output model. The construction of a model is illustrated for a gas valve component of a semiconductor manufacturing piece of equipment: the rapid thermal multiprocessor (RTM) at the Center for Integrated Systems of Stanford University [4, 5].

The task of modeling a discrete event system can be decomposed into four parts. While not all systems are easily modeled by using this methodology, the steps described below have proven to be helpful for modeling the RTM and are general enough to be of widespread practicality.

Step 1 – Model the high-level behavior of the plant. This results in the description of a fundamental process Ξ that best describes the possible functions of the (physical) plant.

Step 2 – Design a logical interface to the physical plant, starting from the fundamental process Ξ . This consists of designing low-level routines that interact with the physical system by triggering actuators or reporting sensor readings. The routines are represented by *actuating* and *sensing* processes Ψ_i .

Step 3 – Compose the description of the fundamental process with the above routines. This results in a candidate process model P' for the plant.

Step 4 – Choose a subprocess P of the process P' found in step 3 so that some desired properties of the plant are satisfied.

The fundamental process $\Xi = (\Sigma', L_\Xi, M_\Xi)$ models the qualitative changes that can occur in the system under consideration. It can be thought of as the fundamental behavior that is consistent with the physical equipment to be controlled. It represents the abstract view of the functionalities of the system.

At a more detailed level of modeling, each event in the fundamental process corresponds to a sequence of events taking place in the *actuating* and *sensing* processes. These underlying sequences of events are modeled by p separate processes or routines, $\Psi_i = (\Sigma_i, L_{\Psi_i}, M_{\Psi_i})$, $1 \leq i \leq p$, with the languages L_{Ψ_i} consisting of a command followed by a finite number of responses, *i.e.* $L_{\Psi_i} \subseteq \Sigma_c \cdot \Sigma_r^*$ (Σ_c and Σ_r are the sets of all commands resp. responses in the sets Σ_i). The initial command can be thought of as the call of the routine, while the responses as its effects. Note that the languages M_{Ψ_i} are not necessarily prefix-closed, *i.e.* $M_{\Psi_i} \subset L_{\Psi_i}$. In particular, we suppose that the only markings are the strings in L_{Ψ_i} without possible continuation, *i.e.* $M_{\Psi_i} = \{s \in L_{\Psi_i} \mid s \cdot \Sigma \cap L_{\Psi_i} = \emptyset\}$.

The interaction of fundamental process and routines can be explained in the following way. A command corresponding to an available routine is given and, after the command, the fundamental process starts producing responses from the routines according to its own possible behavior described by the language L_Ξ . The routines that have been started with a command and that have not yet ended, *i.e.* that have not yet reached a marking, are called *running* and they can be activated again only after having reached a marking.

A candidate process model for the input/output plant is the process $P' = (\Sigma, L_{P'}, M_{P'})$ given by

$$P' = \Xi \parallel (\parallel_{i=1}^p \Psi_i^*) \quad (6)$$

with the alphabet $\Sigma = \Sigma' \cup \Sigma_1 \cup \Sigma_2 \cdots \cup \Sigma_p$.

The process Ψ_i^* (see also the sequential composition of Inan and Varaiya [10]) is defined by

$$\Psi_i^* = (\Sigma_i, \overline{M_{\Psi_i}^*}, M_{\Psi_i}^*)$$

where the symbol $*$ for the languages in this equation stands for the extension of the Kleene closure to languages. Given a marked language M , this means that $s \in M^*$ if and only if there exist strings $s_1, s_2 \cdots s_m \in M$, for some m such that $s_1.s_2.\cdots.s_m = s$.

In equation (6), the processes Ψ_i^* model the repetitive execution of the routines Ψ_i . Note, however, that this repetitive execution must occur for each individual routine in a sequential fashion only. The routines ask the use of some resource, and must possibly wait until the fundamental process is ready to make the resource available. This is therefore expressed by the *synchronous composition* of the fundamental process Ξ with the process $\parallel_{i=1}^p \Psi_i^*$ resulting from the parallel composition of the repeated routines Ψ_i^* .

3.1 Nonblocking Plants with Terminating Routines

On the plant process obtained in equation (6) we want to enforce an additional constraint. We require that all the running routines can always spontaneously terminate by reaching a marking, *i.e.* the following condition must be satisfied for a process $P = (\Sigma, L_P, M_P)$

$$\forall s \in L_P \exists t \in \Sigma_r^* \mid \text{gproj}[\overline{M_{\Psi_i}^*}](s.t) \in M_{\Psi_i}^* \quad \forall i = 1, \dots, p. \quad (7)$$

The class of languages satisfying equation (7) is closed under union as stated in the next proposition.

Proposition 1 *The set of languages satisfying equation (7) is closed under union of languages. Then, given L'_P as the language of the process P' obtained from equation (6), there exists a supremal sublanguage of L'_P satisfying equation (7). Moreover, this supremal sublanguage is controllable w.r.t. L'_P and Σ_r .*

Proof: The proof is easy by noting that the class of languages satisfying equation (7) is closed under union because the property can be checked for each string in $M_{\Psi_i}^*$ or prefix individually. From this property, the existence of a supremal sublanguage of L_P satisfying equation (7) is automatically proved. The controllability w.r.t. L'_P and Σ_r of this language follows from the definition of the languages M_{Ψ_i} which have an element of Σ_c as first element of any string, and then a sequence of elements from Σ_r . ■

This proposition implies that there exists a least restrictive plant P satisfying equation (7), *i.e.* a plant allowing to start as many routines as possible under the constraint given by equation (7).

Then, we can use P as a model for the plant (combining fundamental process and routines) because by choosing the model we only limit a priori the availability of the routines that may be or may not be started in the candidate plant P' . While every routine that is not running can be usually started at any time with a command in the process P' , the enforcement of equation (7) requires that a routine be initiated only if this routine and all the routines currently running can eventually terminate by reaching a marking.

The language of the least restrictive plant can be determined as follows. Consider any command in the candidate plant P' . If after a string s in $L_{P'}$ and this command it is possible to reach a point where a routine cannot spontaneously terminate, then the continuation of the string with the command should be removed from the language.

This can be formulated as follows. Take $L_P = L_{P'}$ and consider any string $s \in L_P$ and $\sigma_c \in \Sigma_c$, then

$$\begin{aligned} \text{if } \exists i \mid \overline{\text{gproj}[M_{\Psi_i}^*](s.\sigma_c.\Sigma_r^*)} \cap M_{\Psi_i}^* &\neq \overline{\text{gproj}[M_{\Psi_i}^*](s.\sigma_c.\Sigma_r^*)} \cap M_{\Psi_i}^* \\ &\Rightarrow \text{remove } \{s.\sigma_c.\Sigma_r^*\} \text{ from } L_P \end{aligned}$$

This should be repeated for every command σ_c in any string s originally contained in $L_{P'}$. The resulting L_P is the language of a plant $P = (\Sigma, L_P, L_P \cap M_{P'})$ satisfying equation (7).

Note that this plant is not necessarily non-blocking because all running routines could have terminated while the fundamental process is not marked. Then the non-blocking plant P_{nb} can be determined using the notion of controllability. In fact the least restrictive non-blocking plant satisfying equation (7) is given by

$$P_{nb} = (\Sigma, \overline{M_{P_{nb}}}, M_{P_{nb}})$$

where $M_{P_{nb}}$ is the supremal sublanguage of $L_P \cap M_{P'}$ which is controllable w.r.t. to L_P and Σ_r . Note also that with the step from P to P_{nb} the property given by equation (7) is preserved.

3.2 Plants subject to Communication delays

In real-life systems the message exchanges between the plant and the controller are affected by communication delays. A message sent by a process needs some time to reach the other process. Besides, the time needed for the commands to be executed can be modeled as a delay in the transmission of the responses.

Because of the communication delays, there can be inconsistencies between the controller and the plant. For instance, the controller could send a command to the plant without the knowledge of a response already produced by the plant but not yet arrived at the controller input.

Then, in order to enforce a desired behavior on a plant in a closed-loop affected by communication delays, more sophisticated supervisors/controllers must be used in general. Li and Wonham [11] (in the original setup of Ramadge and Wonham) and the author [12] (in an I/O-setup) discuss the properties of such supervisors/controllers.

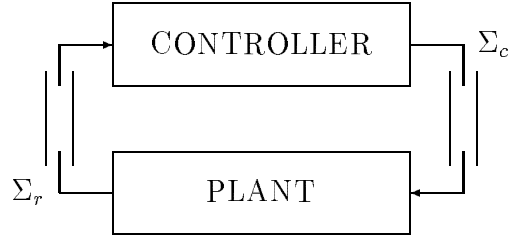


Figure 2: Closed-loop connection of plant and controller processes with communication delays.

More sophisticated controllers, however, are not necessary if the plant satisfies additional properties. For instance, if the plant is such that no command can be given while some routine is running, *i.e.* if no element of Σ_c can be accepted by the plant after a strings s such that $\text{gproj}[\overline{M_{\Psi_i}^*}](s) \notin M_{\Psi_i}^*$ for some routine index i , then a controller solving the problem on page 18, correctly enforces the specification on the plant affected by communication delays. Such plants trivially behave correctly under delays because the controller is forced to wait until *the current* routine has terminated.

3.3 Two-stage Design

Therefore we can say that a model for the plant (the control model) is *designed* from the physical description of the system. The aim of this design is to obtain a plant model ensuring desired operating constraints.

Given the plant model, the second step consists in designing the controller which enforces given specifications on the plant's closed-loop behavior. We can see these two design steps in figure 3.

As we showed for the case of a plant-controller pair affected by communication delays, the design of a controller can be performed in a less computationally intensive way and using known algorithms if the plant satisfies given properties.

Plant design versus controller design. Now we could ask ourselves if it makes sense to invest more time in the design of the model instead of determining a general model and then take care with the controller design of both operational constraints and control specifications. A partial list of reasons for doing so is given below.

1. The plant model is designed once at the beginning and then seldom changed (*e.g.* after a hardware change). The controller design is performed often. Thus the design of a more “benign” plant model can pay off in the long run.
2. In general, for the considered control problem, some restrictions on the class of plant models can be introduced without prejudicing the achievement of the control objectives.
3. Some constraints on the plant class are natural. Also, the need for new algorithms for the computation of more sophisticated controllers can be avoided.

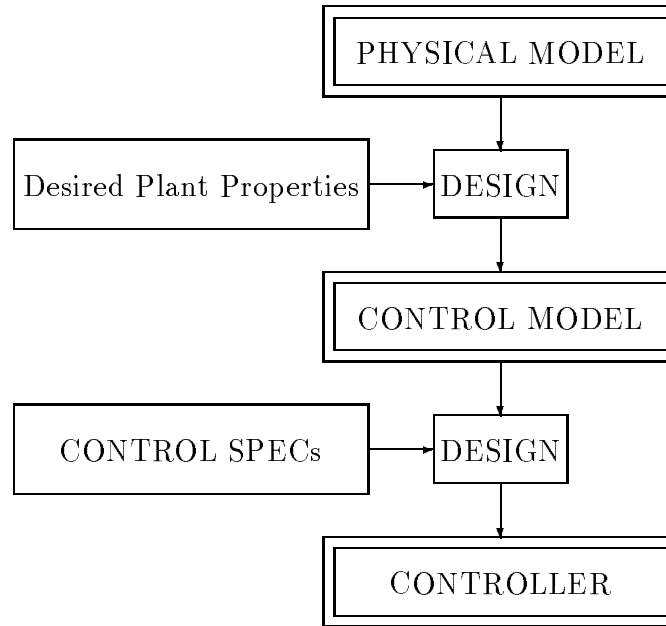


Figure 3: Two-stage design: 1) The control model for the plant is designed from the physical model of the system and the desired plant properties. 2) The controller is designed from the plant model and the control specifications

The design of a plant model can be of great importance for the success of the automatic synthesis of controllers. While the complexity of the computation of a controller for a continuous-valued system depends on the model order, the complexity of the computation of a controller for a discrete event system is not only a function of the size of the model but can heavily depend (and maybe even more crucially) on the structure of the model.

The success of the whole domain of discrete event systems will depend on the future ability of providing the plant models with the necessary *structure* suitable for the controller design.

4. AN EXAMPLE

We now present as an example the construction procedure for a gas valve of the considered semiconductor manufacturing piece of equipment. The complete alphabets of commands and responses that are modeled are

$$\begin{aligned}\Sigma_r &= \{r_valve_failed, r_valve_opened, r_valve_closed\} \\ \Sigma_c &= \{c_open_valve, c_close_valve, c_repair_valve\}.\end{aligned}$$

We adopt the convention that the transition labels starting with “c_” and “r_” denote commands and responses respectively. First, the fundamental process Ξ for the gas valve is modeled by an automaton over the alphabet

$$\Sigma_{\Xi} = \{r_valve_opened, r_valve_closed, r_valve_failed, c_repair_valve\}.$$

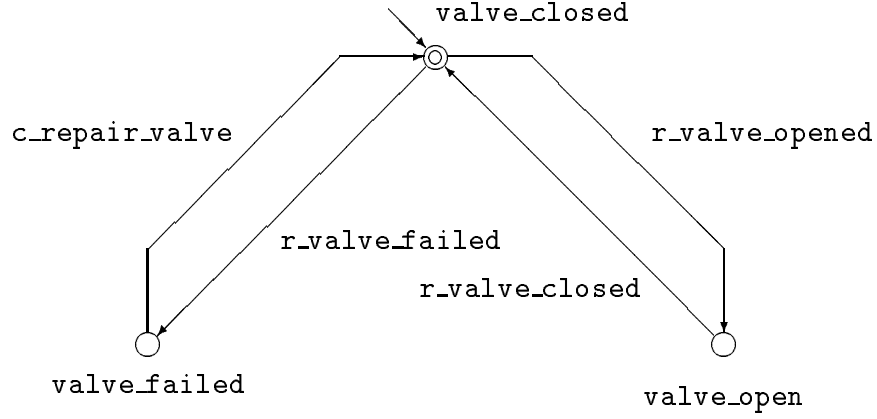


Figure 4: Automaton model of the fundamental valve process Ξ . The arrow on the top indicates the initial state. States marked by two concentric circles are marked states

The automaton is shown in figure 4, where the only marked state is the initial state. The marked language M_{Ξ} is

$$\{((r_valve_opened.r_valve_closed) + (r_valve_failed.c_repair_valve))^*\}$$

and $L_{\Xi} = \overline{M_{\Xi}}$. Secondly, the sensing and actuating processes Ψ_i are given by the languages

$$\begin{aligned} M_{\Psi_1} &= \{c_open_valve.(r_valve_opened + r_valve_failed)\} \\ M_{\Psi_2} &= \{c_close_valve.r_valve_closed\} \\ M_{\Psi_3} &= \{c_repair_valve\} \end{aligned}$$

and $L_{\Psi_i} = \overline{M_{\Psi_i}}$. The sequences in M_{Ψ_1} correspond to sensors that indicate whether the command to open the valve is successful or not. The strings in M_{Ψ_2} and M_{Ψ_3} are more detailed descriptions of sequences of events corresponding to closing and repairing the valve. We can see the process Ψ_1 and Ψ_1^* in figure 5.

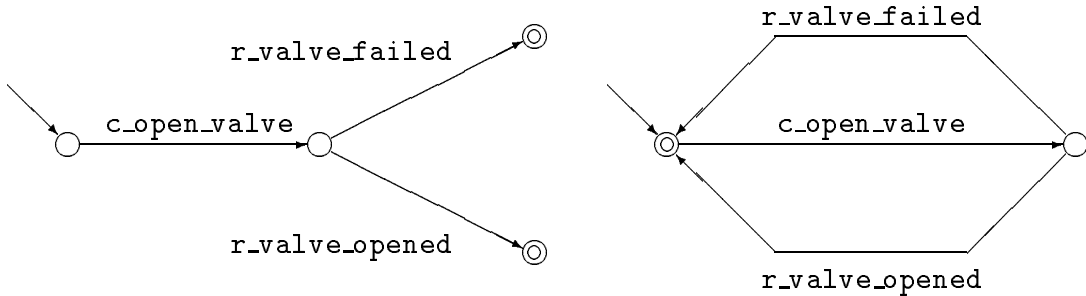


Figure 5: Actuating process Ψ_1 on the left and its infinite repetition Ψ_1^* on the right, described by automata.

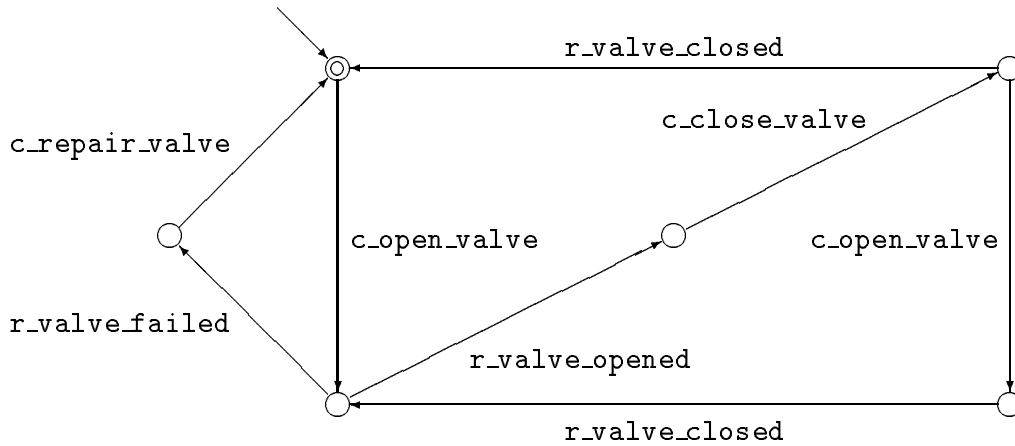


Figure 6: Least restrictive non-blocking plant also allowing the termination of all running routines.

Thirdly, composing the processes as in (6) under constraint equation (7) and of non-blockingness yields the complete input/output plant model of figure 6.

A simpler model for the gas valve process, allowing only one running routine, is given on figure 7.

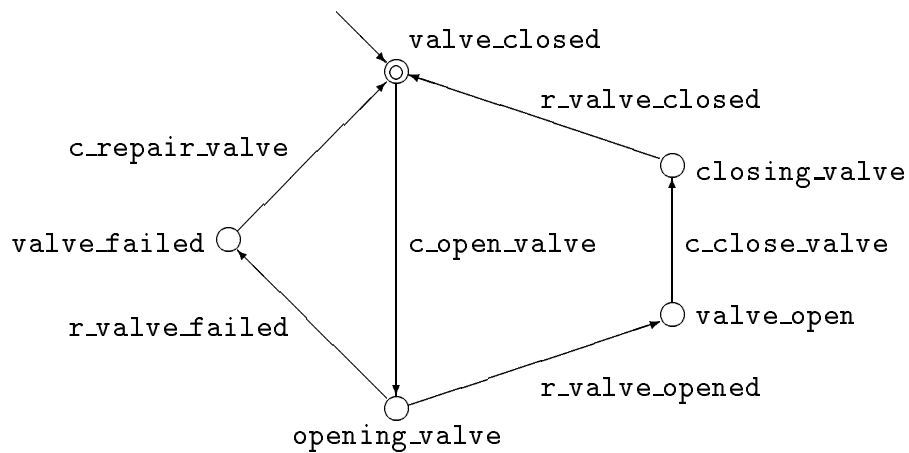


Figure 7: Simple model for the gas valve process P .

It is given by the process $P = (\Sigma, \overline{M_P}, M_P)$ where

$$M_P = \{ (c_open_valve.(r_valve_opened.c_close_valve.r_valve_closed + r_valve_failed.c_repair_valve))^* \}.$$

Only the initial state of the automaton is marked.

5. CONCLUSION

In this paper an input/output version of the supervisory control problem of Ramadge and Wonham has been introduced. This modified problem is based on an input/output model of both plant and supervisor. Then, the issue of modeling an input/output plant has been addressed, and a methodology for obtaining a plant model has been presented.

The article has tried to show that the modeling of a discrete event system is not an “identification problem” (like in the case of continuous-valued systems) but more a “design problem”. During the plant model design, the plant has to be provided with structure which then simplifies the successive controller design problem.

The author believes that the success of the synthesis of controllers for discrete event systems strongly depends on our future ability to introduce structure in our models such as to render computational complexity pheasible in practice.

REFERENCES

- [1] P.J. Ramadge and W.M. Wonham. The control of discrete event systems. *Proc. of the IEEE*, 77(1):81–98, January 1989.
- [2] C.H. Golaszewski and P.J. Ramadge. Control of discrete event systems with forced events. In *Proc. of 26th Conf. Decision and Control*, pages 247–251, Los Angeles, CA, USA, December 1987.
- [3] 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.
- [4] S. Balemi. Discrete-event systems control of a rapid thermal multiprocessor. In *Proc. of 7th IFAC/IFIP/IFORS/IMACS/ISPE Symposium on Information Control Problems in Manufacturing Technology (INCOM)*, Toronto, Canada, May 1992.
- [5] S. Balemi, G.J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G.F. Franklin. Supervisory control of a rapid thermal multiprocessor. *Joint issue Automatica and IEEE Trans. Autom. Control on Meeting the Challenge of Computer Science in the Industrial Applications of Control*, 1993. to appear.
- [6] M. Heymann. Concurrency and discrete event control. *IEEE Control System Magazine*, 10(4):103–112, June 1990.
- [7] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, January 1987.
- [8] S. Balemi. *Control of Discrete Event Systems: Theory and Application*. PhD thesis, Automatic Control Laboratory, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, May 1992.

-
- [9] W.M. Wonham and P.J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM J. Control Optim.*, 25(3):637–659, May 1987.
 - [10] K. Inan and P. Varaiya. Algebras of discrete event models. *Proc. of the IEEE*, 77(1):24–38, January 1989.
 - [11] Y. Li and W.M. Wonham. On supervisory control of real-time discrete event systems. *Information Sciences*, 46(3):159–183, 1988.
 - [12] S. Balemi. Communication delays in connections of input/output discrete event processes. In *Proc. of 31st Conf. Decision and Control*, Tucson, AZ, USA, December 1992.

