14

Modelling and Implementation of Supervisory Control Systems Using State Machines with Outputs

Moacyr Carlos Possan Junior[1,2] and André Bittencourt Leal[2]
[1]*Whirlpool Latin America*
[2]*Santa Catarina State University – UDESC*
Brazil

1. Introduction

The growth of the complexity of automated systems in industry has occurred extensively in recent years due to the production demand increase, quality improvements and flexibility to restructure the manufacturing systems in order to satisfy new procedures. Nevertheless, the evolution of control devices and their functionalities, such as processor speed, memory and network communication has advanced in parallel with the factories' requirements. Although the evolution of automation in industrial processes is a fact, there is still a scarcity of formal methods for analysis, project and implementation of control systems for Discrete Event Systems (DES) in order to reduce the development time, reduce human resources investment, and satisfy the operational requirements for certain systems in an effective way. Furthermore, the occurrence of programming bugs resulting in errors due to interruption in the process and losses due to poorly designed software is obviously unacceptable nowadays in this market that has a just-in-time mindset and is strongly focused on profits.

Usually, the projects for supervisory control systems are based on the knowledge of the system's practitioner, according to his experience in programming. The usage of formal methods is sparse, such that the reuse of documentation and source code as well as the dissemination of the knowledge generated are both impaired. Moreover, the automation of manufacturing systems has brought an increase in the complexity of control systems, so that to elaborate and implement robust and reliable control logic is not a trivial task. In order to minimize the risks due to programming errors and to permit a formal method for modelling DES, Ramadge & Wonham (1989) introduced the Supervisory Control Theory (SCT), which guarantees optimal control logic (nonblocking and minimally restrictive) for these systems.

A Discrete Event System (DES) consists of a system with discrete states that are driven by events. In other words, its state evolution depends on the occurrence of discrete asynchronous events over time (Cassandras & Lafortune, 2008). Discrete Event Systems are quite common in the industry nowadays and the events may be classified as uncontrollable and controllable. Examples of controllable events are the start and end of an operation and examples of uncontrollable events are the activation and deactivation of a presence sensor.

The Supervisory Control Theory (SCT) is already widespread in the academic environment, using the automata theory as a base to model the control systems. However, such a theory is not common in the industrial environment. Therefore the resolution of supervisory control problems has been done without the usage of a formal procedure.

The SCT allows the solution of control problems in a systematic manner. This technique guarantees that the resulting supervisor will satisfy the specifications imposed by the designer, avoiding general control issues, such as blocking. Besides, due to its heuristic nature, the SCT facilitates the code writing changes before implementation in a controller, in case there is some inclusion/exclusion of devices or changes in the system layout.

Although the SCT provides an automatic method to synthesize control systems for DES, when analysing the monolithic supervisor obtained from the SCT, it is difficult to visualize the process dynamics in an easy way as the system complexity grows. That occurs due to the large number of states and no distinction about what kind of event, uncontrollable or controllable, has priority to occur when the supervisor is in a certain state. Furthermore, the implementation of this supervisor in a controller will require considerable non-volatile memory, neither being an elegant solution nor justifying the adoption of a formal method. PLC (Programmable Logic Controller) implementation of DES supervisory control was discussed in many works, as in (Ariñez et al., 1993; Lauzon, 1995; Leduc & Wonham, 1995; Leduc, 1996; Lauzon et al. 1997; Fabian & Hellgren, 1998; Dietrich et al., 2002; Hellgren et al., 2002; Liu & Darabi, 2002; Music & Matko, 2002; Queiroz & Cury, 2002; Chandra et al., 2003; Hasdemir et al., 2004; Manesis & Akantziotis, 2005; Vieira et al., 2006, Noorbakhsh & Afzalian 2007; Hasdemir et al., 2008; Silva et al., 2008; Leal et al., 2009; Uzam et al., 2009; Moura & Guedes, 2010).

In brief, other works presented methodologies using extended automata with variables in an attempt to minimize the exponential growth of states resulting from the automata composition, such as (Chen & Lin, 2000; Yang & Gohari, 2005; Gaudin & Deussen, 2007; Skoldstam et al., 2008), amongst others.

This chapter intends to propose a formal methodology to model control systems for industrial plants through extended automata called Mealy state machines (Mealy, 1955) and subsequent implementation in Programmable Logic Controllers (PLCs) using the Ladder language. An algorithm proposed by Possan (2009), which explores the benefits of SCT to design a supervisor, is used to convert the automaton which represents the supervisor to a finite state machine with outputs. Another algorithm is then used to reduce the state machine to have a simplified structure implemented in a PLC. The simplified state machine representation is a different way of viewing a DES, resulting in a systematic way to implement the source code in a controller with reduced memory usage. The code implementation takes into consideration some common issues found in synchronous controllers, such as PLCs.

A Mealy state machine is a finite state machine with outputs, composed of an oriented graph where the nodes are called states and arcs are called transitions. It is a powerful model to represent the behavior of processes in general.

The chapter is structured as follows: Section 2 introduces the proposed methodology; Section 3 covers the monolithic approach defined by the SCT; Section 4 shows a system to be

used as example to illustrate the modelling and implementation; Section 5 presents in detail the transformation algorithm to obtain a Mealy state machine; Section 6 relates to the state machine simplification procedure; Section 7 presents common issues found during supervisor implementation in synchronous controllers while Section 8 describes how to implement the simplified state machine in PLC using the Ladder language. Finally, Section 9 covers the conclusion.

2. Methodology for the control system design

Figure 1 shows an overview of the proposed methodology. It starts with the supervisor synthesis. The synthesis is done based on the SCT and a monolithic supervisor is obtained. The supervisor is then used as input for the transformation algorithm to obtain the state machine. The machine is then simplified to have a reduced number of state transitions. The simplified machine, on the other hand, represents a model to generate the code for a controller (PLC, microcontroller or any other data processing unit). This chapter is focused on the state machine implementation using the Ladder language for PLCs.

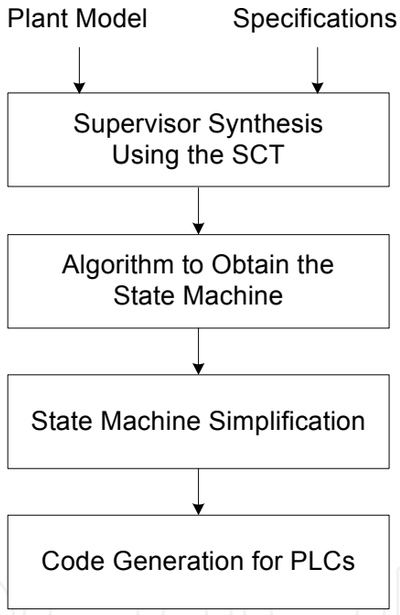


Fig. 1. Proposed methodology

The synthesis of the supervisor and the obtaining and simplification of the state machine will be described in the next three sections.

3. Supervisor synthesis using the SCT

The monolithic approach for the synthesis of an optimal supervisor (nonblocking and minimally restrictive) is based on three main steps:

a. obtain a model for the physical system (plant) to be controlled;
b. obtain a model which represents the specifications to be respected; and
c. synthesize a nonblocking and optimal control logic.

The plant model is built through the synchronous composition (Cassandras & Lafortune, 2008) of all the existing subsystem models in the system. The same procedure is done to build the specification model. The plant and the target language, obtained through the synchronous composition of the plant with the specification, will be used as input to obtain the monolithic supervisor. This procedure can be done using the computational tool named *Grail for Supervisory Control* (Reiser et al., 2006).

In the SCT, the plant is assumed to spontaneously generate events. The supervisor observes the string of events generated by the plant and might prevent the plant from generating a subset of the controllable events, thus disabling them. However, the supervisor has no means of forcing the plant to generate an event, as shown in Figure 2-a.

In practice, the modelled behavior of the plant does not correspond exactly to the real behavior due to the assumption that controllable events are not generated by the plant, as presumed by the SCT. This is because in most real systems, the events modelled as controllable correspond to commands that actually must be generated by the control system. These commands must be sent by the controller to the actuators because they would not occur spontaneously. Thus, the implementation is performed according to the structure proposed by Queiroz & Cury (2000) to keep such coherence. Figure 2-b shows the representation for real systems, where the electric signals coming from the sensors (responses from the plant) correspond to the observed (uncontrollable) events while the electric signals sent to the actuators (actions in the plant) correspond to the disabled (controllable) events.

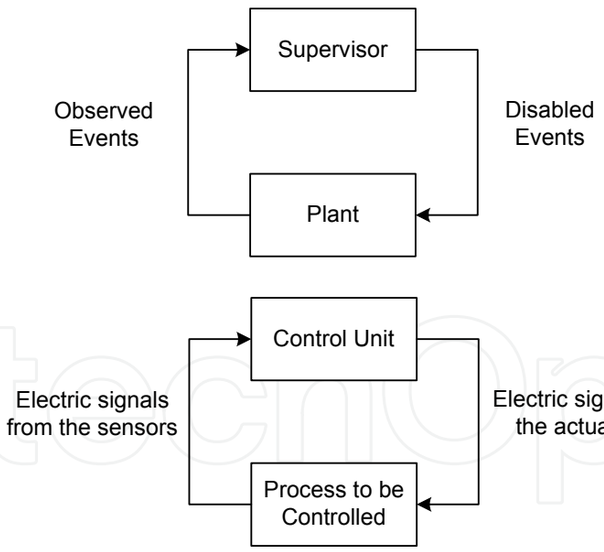


Fig. 2. Monolithic approach: (a) Ramadge-Wonham Framework, (b) Similar representation for a real system

4. A motivation example

In this section, a manufacturing system is used to demonstrate the proposed methodology. This system is composed of three apparatus and two intermediary buffers with a capacity of one which are available between the apparatus, as illustrated in Figure 3.

The apparatus are represented by *Ai*, where *i = 1, 2, 3* and the buffers are represented by *Bj*, where *j = 1, 2*.

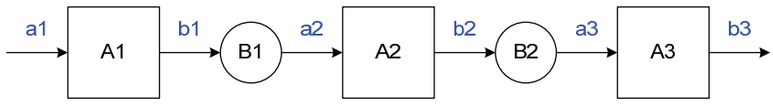


Fig. 3. Manufacturing system

The controllable events that correspond to the start of the apparatus' operation are represented by a_x, while the uncontrollable events that correspond to the end of operation are represented by b_x, where $x = 1, 2, 3$.

The plant and specifications are modelled using automata and the controllable events are represented with a dash. The behavior of each apparatus (or subsystem) can be modelled by the automaton shown in Figure 4. Notice that state *0* is double circled. This is a marked state that represents a completed task.

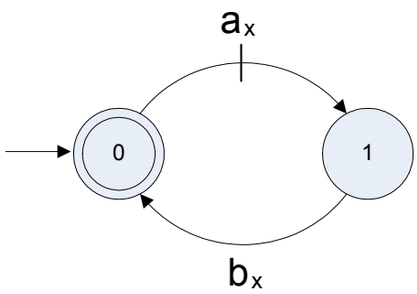


Fig. 4. Automaton for the apparatus Ax

When modelling this plant, only the apparatus were taken into consideration. The buffers were considered only in the in the control specification model.

The specifications for this system are restrictions of coordination to avoid overflow (the apparatus finishes its task but the output buffer is already full) or underflow (the apparatus starts working without any item to fetch from the input buffer). Those restrictions point out the idea that it is necessary to alternate b_1-a_2 and b_2-a_3, respectively. This means that the start of operation for an apparatus (event a_{x+1}) will only be allowed when its input buffer is loaded (event b_x), as shown in Figure 5.

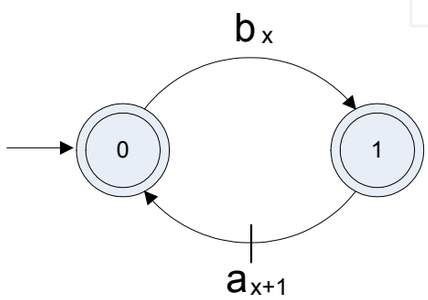


Fig. 5. Automaton for the control specifications

The synchronous composition of the plants with the specifications will result in the target language (Ramadge & Wonham, 1989). The calculation of the minimally restrictive and nonblocking supervisor is based on an iterative process which identifies and eliminates bad states in the automata that models the target language.

The monolithic supervisor found with the usage of *Grail* (Reiser et al., 2006) has 18 states and 32 state transitions.

The supervisor can be reduced for a later comparison with the state machine. This procedure is done to simplify the supervisor size and also the number of transitions (Su & Wonham, 2004). An algorithm for supervisor reduction is used to obtain fewer states and transitions than the original supervisor (Sivolella, 2005).

Figure 6 shows the automata which represents the reduced supervisor. The disabled events for each state are represented by red dashes.

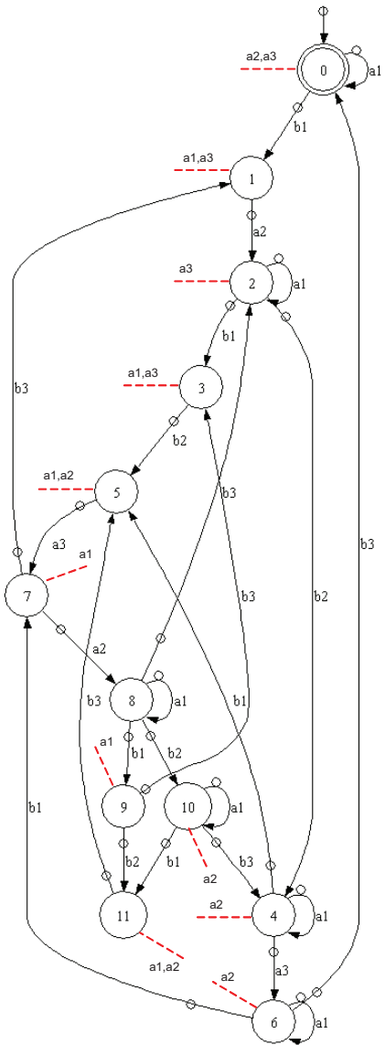


Fig. 6. Reduced supervisor and disabled events for the manufacturing system

Notice that the reduced supervisor has 12 states and 20 state transitions. The supervisor reduction algorithm creates self-loop transitions. However, these transitions are not relevant because they do not cause a state change.

Transitions among the states can occur either by controllable events (a_x) or by uncontrollable events (b_x). In case the designer intends to implement the code in the controller based on this model, he will have to decide on the kind of event to give priority. Furthermore, it is not trivial to visualize what are the active apparatus and what events modelled as uncontrollable are expected to occur at a certain state of the supervisor.

5. Algorithm to obtain the state machine with outputs

The state machine obtained with the proposed algorithm consists of a Mealy machine (Mealy, 1955). In a Mealy machine, a transition can have one or more output actions (set one or more controllable events) and any output action can be used in more than one transition. The output actions are not associated with the states, which are passive. Thus, the actions can be associated with more than one state.

A simple example of a Mealy machine is shown in Figure 7. Transitions and actions are separated by dashes. It is a machine with two states where, when in State 1, Transition 1 makes the machine to go from State 1 to State 2 and takes Action 1. When in State 2, Transition 1 plus Transition 2 makes the machine go from State 2 to State 1 and Action 2 is taken.

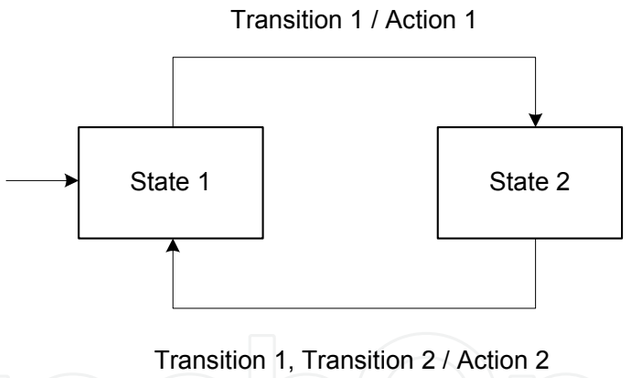


Fig. 7. Example of a Mealy machine

In this proposal, the transition between two states is performed by means of one or more uncontrollable events in the system. For each transition, an output action may be generated.

The algorithm proposed to create the state machine works in an iterative process, looping through the input data to obtain the states, transitions and actions that compose a finite state machine. As soon as all the input data is processed, the state machine is completed. The algorithm is shown in Figure 8.

Input data for the algorithm is the information about the plant, the supervisor and the list of disabled events. Output data are the states, transitions and actions which compose the state machine.

In the transformation process from the supervisor to the state machine, the output actions correspond to the controllable events in SCT while the transitions among the states correspond to the uncontrollable events.

The initialization considers the initial states of the input data. These data compose the starting point of the state machine representing the condition where the physical process has not started.

The next step is to create a states queue. The queue is required to store the states that are being obtained iteratively to be treated after the treatment of a current state has finished. The queue consists of a First In, First Out (FIFO) structure. A while loop is suggested to treat all the states available until the states queue becomes empty.

```
1: Read the supervisor (S) automata and the disabling map
(DM)
2: Split the supervisor automata according to its events
controllability – Su (uncontrollable) and Sc (controllable)
3: Create the initial state for the machine = initial state of
supervisor automata
4: Create a list with the next states which are obtained from
the current states that need to be processed
5: while (States List > 0) do
6:      Read state from the list and consider as a current
state
7:      Create a Transitions List for this state
8:      for (Transitions List) do
9:           Read Su, MD, Sc
10:          if (Sc evolved) then
11:               Update MD e Sc
12:          else
13:               Create a new next state
14:          end if
15:          if (State was already created) then
16:               do nothing
17:          else
18:               Add the next state to the States
                  List according to the next state
                  from the supervisor automata
19:          end if
20:     end for
21:     Decrement States List
22: end while
23: Remove unreachable states
24: Save the Monolithic Mealy State Machine
```

Fig. 8. Algorithm to obtain the state machine

For each state, a transition queue is created as well. A "for loop" is suggested to treat all the valid transitions for that specific state until all the transitions available in the queue are processed.

In order to create the list of valid transitions for a certain state in the state machine, it is first necessary to divide the plant and the supervisor in two parts according to the controllability of the events and their transitions. Thus G_u and S_u will have the list of automata state transitions due to uncontrollable events. The same happens for the controllable part.

The data reading sequence begins at the uncontrollable part (G_u and S_u). After that, the algorithm evaluates the disabled events for the actual state of the supervisor. This will define if the controllable part of the system (G_c and S_c) can evolve or not.

In other words, the resulting state machine gives priority to the occurrence of uncontrollable events, waiting to receive some response from the physical system to make a decision about what controllable events to disable.

The valid transitions for a certain state are the uncontrollable events that create state evolutions from the current active states in the uncontrollable part of the plant and supervisor. The combination of more than one uncontrollable event is also considered as a transition.

After the uncontrollable part is processed, the controllable part is processed. The controllable disabled events are evaluated according to the current state the supervisor is in. While the disabled events are forbidden to occur, the remaining ones may originate the actions.

The valid actions for a certain state are the controllable events that are not disabled at the supervisor state and cause states evolution in the controllable part of the plant and the supervisor.

When an action occurs, the algorithm checks if the controllable part of the supervisor has evolved. If so, then the disabled events for the destination state are evaluated in order to verify if another action may occur. This step assures that all the actions possible to occur for the same transition are processed. It means that more than one action can occur for the same transition.

If S_c has not evolved, then a new state is created. This state is compared with the other states and if it already exists, it is ignored. Otherwise, it is added to the states queue to be treated later.

For each transition due to the uncontrollable events, G_u and S_u obtained from the SCT evolve. The same happens for the controllable part in the occurrence of an action.

This methodology is consistent with the Mealy machine approach, where the outputs (actions) depend on the current state and valid inputs (transitions).

After the procedure for creating a transition and corresponding actions is finished, the algorithm will treat the remaining transitions available in the transitions queue. When all the transitions in the queue are treated, the algorithm will evaluate the next state available in the states queue. These iterative processes are performed until the states queue is empty. This means that the finite state machine has been completed.

The state machine for the manufacturing system with the proposed algorithm is shown in Figure 9, composed of 8 states and 22 state transitions.

The states are named according to the apparatus that are operating at a certain point in time and the buffers that are full. The transitions are represented by the uncontrollable events, and the taken actions, if any, are separated from the transitions by a slash (/). The disabled events are represented by red dashes. Notice that in this model the transitions may occur due to more than one uncontrollable event, and the actions, if any, may be due to more than one controllable event.

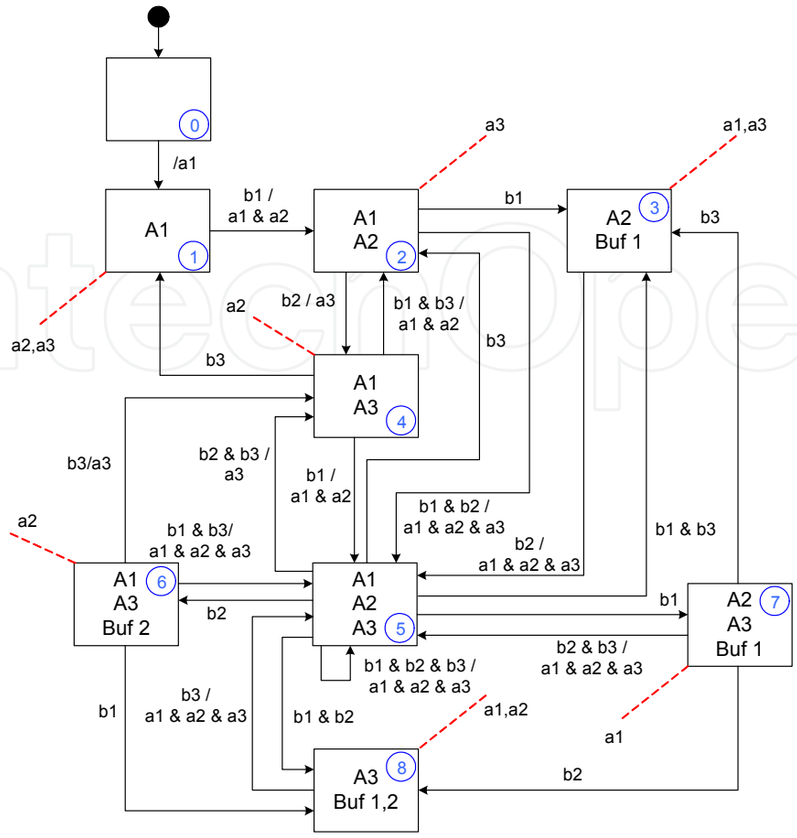


Fig. 9. Mealy state machine

6. State machine simplification

When the state machine model considers the treatment of more than one uncontrollable event, this may result in the disadvantage of an exponential growth of transitions, depending on the number of plants modelled and how many may be enabled at the same time. The number of transitions created for a certain state is on the order of $2^n - 1$, where n represents the number of uncontrollable events present in the model and possible to occur for that state. Therefore, for larger and more complex systems, the code size would be affected significantly to satisfy that condition, being a convincing reason for not using such a methodology. An alternative solution for that is to consider a reduced state machine where the state transitions are represented only by transitions that result in actions. Transitions that do not result in actions are represented by self-loops inside their current state. For example, for state 2 of the state machine represented in Figure 9, transition b_1 could be represented as a self-loop. Although the state machine evolves to state 3, there is no action taken during this transition.

Although they do not result in actions, these transitions are important to represent the plant dynamics and cannot be simply disregarded during the implementation process. The controller program must capture their occurrence and store it in some internal variable to be used in the decision-making process when other transitions occur. The algorithm presented in Figure 10 describes the process to reduce a monolithic Mealy machine.

```
1: Read the Mealy machine
2: for State = from i to n do % where n is the number of states
3:         for Transition = from i to t do % where t is the
number of transitions from the current state
4:                 if (Transition results in action) then
5:                 Keep the transition in the reduced machine
6:                 else
7:                 Create a self-loop represented by dashed
                   lines in the current state
8:                 end if
9:         end for
10: end for
11: Remove unreachable states
12: Save the Reduced Monolithic Mealy State Machine
```

Fig. 10. Algorithm to reduce the state machine

It is important to emphasize that such a procedure to reduce the Mealy machine does not necessarily result in a minimal state machine.

Figure 11 shows the reduced state machine for the manufacturing system. This state machine contains only four states and seven transitions. The transitions illustrated by solid lines represent the occurrence of an action, regardless of whether the transition occurred from one state to another. The transitions illustrated by self-loops in dashed lines inside a current state represent that, although a transition has occurred, an action is not fired.

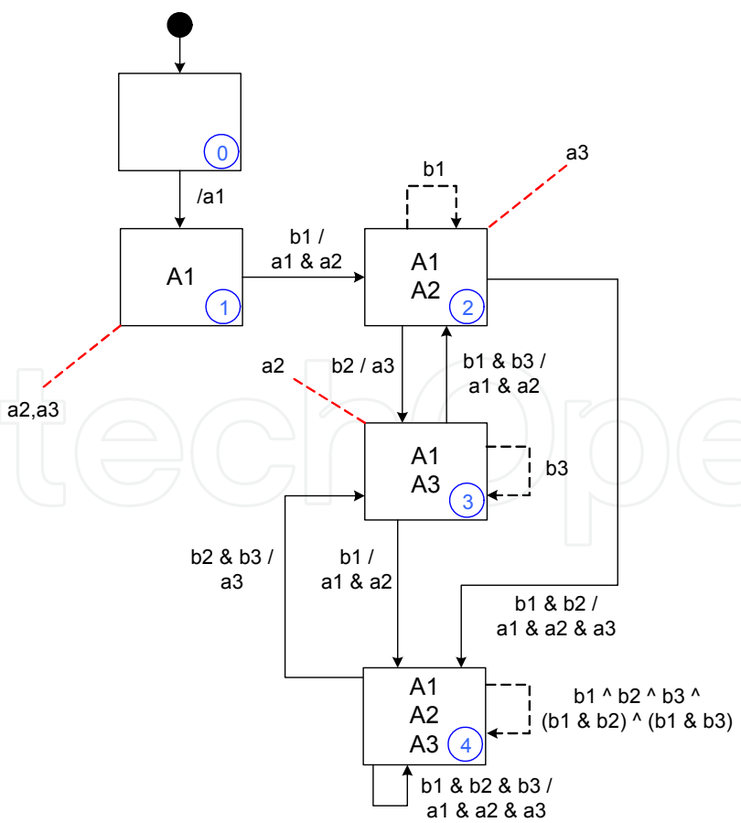


Fig. 11. Reduced Mealy state machine

Consider for instance state *2* of the reduced state machine. If transition b_2 occurs, the state machine evolves to state *3*. In the case that transition b_1 occurs, the model illustrates that situation as a self-loop represented by dashed lines, which means that the practical implementation in the controller must guarantee the proper storage of this information in some internal variable.

When event b_2 occurs with transition b_1 already enabled, transition b_1 & b_2 will be activated, so that the state machine evolves to state 4 and actions a_1, a_2 and a_3 are taken.

The disabled actions shall be represented in the corresponding states to illustrate the control actions forbidden to occur. The states which became unreachable if compared to the original state machine are eliminated by this reduced model.

Transitions due to more than one uncontrollable event and action due to more than one controllable event are still represented in this model. That is relevant information which helps the designer when he intends to implement the control system, so that the program allows several events to be executed inside the same scan cycle of the synchronous controller.

In addition, a new logic operator appears in this model. The operator "^" represents an *exclusive or* condition. In order to understand its function, consider state *4* of the reduced state machine, for example. If any of the transitions listed in the self-loop with dashed lines occur, namely, b_1 ^ b_2 ^ b_3 ^ (b_1 & b_2) ^ (b_1 & b_3), none of the actions will be taken and those transitions will remain enabled. Actions will be taken only when the transitions b_2 & b_3 or b_1 & b_2 & b_3 become valid. The operator "^" appears only for the states with a self-loop represented by dashed lines.

7. Issues with Implementing supervisors in synchronous controllers

According to (Fabian & Hellgren, 1998), "the supervisor implementation is basically a matter of making the Programmable Logic Controller (PLC) behave as a state machine". However, that is not a simple task. Certain issues appear during the implementation process in synchronous controllers, such as PLCs and computers. Those problems exist regardless of the model used to represent the supervisors, by means of automata, Petri nets or colored Petri nets (Basile & Chiacchio, 2007). Such problems are explored in the structure presented here and a solution is presented where possible, in accordance with the Ladder language definition described by the IEC-61131-3 (1993) standard.

7.1 Causality

The SCT considers that all events are generated spontaneously by the plant and that the supervisor tracks the sequence of events generated by the plant and also acts to disable the controllable events in order to avoid any infringement of the control specifications. However, in most practical applications, the controllable events are not generated spontaneously by the physical plant, but only the feedback due to sent commands. So, the question "who generates what?" must be answered (Fabian & Hellgren, 1998), or in other words, who is responsible for the generation of certain kind of events, the supervisor or the plant? (Vieira, 2007).

The supervision scheme proposed by the SCT was already illustrated in Figure 2-a. In this structure, the plant is the responsible for events generation, both controllable and uncontrollable, while the supervisor is just an observer and disables a set of controllable events to satisfy control requirements.

However, in most practical applications, the events modelled as controllable correspond to commands that, indeed, shall be generated by the PLC and sent to the actuators, because they do not occur spontaneously. The plant generates only the uncontrollable events, as a feedback to the stimulus sent by the PLC through the controllable events. Figure 2-b describes the control structure usually employed in practice (Malik, 2002).

In this chapter, the causality problem is solved by representing the model using state machines with outputs instead of automata. Therefore, when this transformation is performed, the SCT changes to a model with inputs and outputs, as suggested by Malik (2002). The signals originated by the sensors correspond to the uncontrollable events while the signals generated by the actuators correspond to the controllable events.

7.2 Event detection

The synchronous nature of PLCs may create issues during the detection of uncontrollable events as described below.

7.2.1 Signals and events

Some implementation problems are due to the lack of an easy way to translate supervisors based on discrete event systems, which are symbolic, asynchronous and occur in discrete instants of time in a synchronous universe and are based on signals such as those from the PLC (Fabian & Hellgren, 1998). In order to avoid a discrepancy between the theory and practice, a signal cannot generate more than one uncontrollable event during an operational cycle of the PLC (Basile & Chiacchio, 2007).

7.2.2 Avalanche effect

The PLC signals may assume boolean values and are sampled periodically. Thus, in order to implement supervisors according to the SCT in PLCs, the events are associated with changes in the PLC input signals, which may cause what is called the *avalanche effect*. This effect occurs when a value change in an input signal is registered as an uncontrollable event and makes the software jump to an arbitrary number of states during the same scan cycle of the PLC. This may occur specifically if a certain uncontrollable event is used to trigger several state transitions in a list, creating behavior similar to an avalanche.

Figure 12 shows an example of the avalanche effect for a conventional supervisor implementation in PLC. The supervisor transitions from state *0* to state *2* with the occurrence of event b_1. It should transition from state *1* to state *2* only with the occurrence of a new event b_1, or transition from state *1* to state *3* with the occurrence of an event b_2. However, the implementation proposed on the right side of Figure 12 does not restrict that event b_1 permits the transition from state *1* to state *2* in the same scan cycle of the PLC. So, the avalanche effect introduces a transition directly from state *0* to state *2*, so

that, even if event b_2 happened, it would not have any effect on the controller dynamics. This is clearly unsatisfactory.

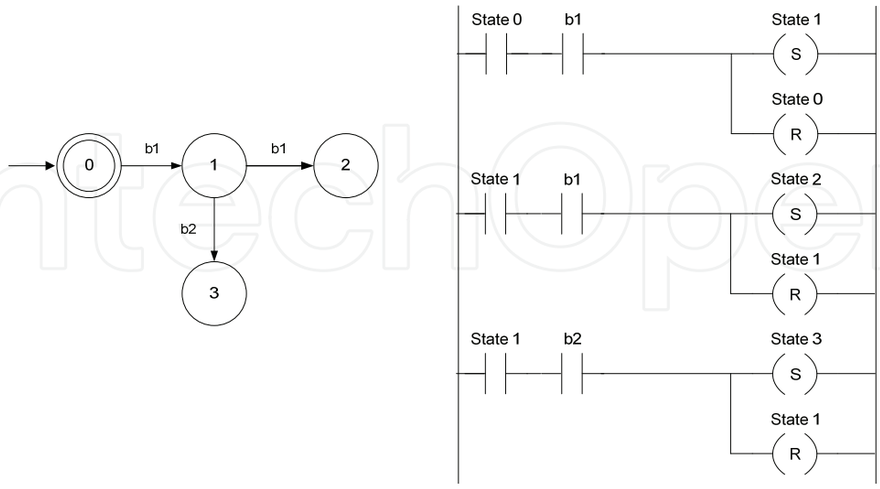


Fig. 12. Example of the Avalanche Effect

In order to solve this problem, Vieira (2007) proposes a procedure where a variable is associated with all the transitions available for the supervisor. That variable is activated every time a new state transition occurs, being deactivated at the end of the section of source code where the supervisor structure is implemented. Every state transition includes the negation of this variable as a condition. Such a solution solves the problem, but has the disadvantage of allowing the occurrence of only one uncontrollable event inside the same PLC scan cycle.

The solution proposed in this chapter consists of the deactivation of the variable corresponding to the event which occurred as soon as a state transition due to that event happens. In this way, the supervisor deactivates that specific event which occurred, allowing transitions due to other events available in the model still to happen. During a scan cycle, if events related to asynchronous specifications occur, they will be treated in this same cycle. Such a proposal solves the avalanche effect problem and permits several uncontrollable events to be treated inside the same PLC scan cycle. Moreover, the constraint that more than one event is treated in the same scan cycle results in another problem called *loss of information* (Vieira, 2007). Such a problem occurs in situations where several events happen and only one is treated per scan cycle. Then, the information related to the occurrence of the remaining events is lost. That issue does not occur in the methodology proposed here due to the possibility of treating all other events which occurred in the same scan cycle. This is possible because there is not just one general variable which is activated when a state transition occurs, but a specific treatment for each event, where the other events remain enabled to fire transitions.

7.2.3 Inability to recognize the order of events

This simultaneity problem was covered first by Fabian & Hellgren (1998) considering the implementation of supervisors in PLCs. When two or more input signals change their values

between two input readings, those changes will be stored as a simultaneous change of uncontrollable events, so that it is not possible to identify which one occurred first, because the PLC performs a synchronous reading of its inputs.

It is noticeable that signal changes may be simultaneous or not, but the events will always be stored as simultaneous during the reading. That problem is called *simultaneity* (Fabian & Hellgren, 1998). Thus, in order to avoid implementation problems, the supervisor shall have a control action which does not depend on the different interleaving of the uncontrollable events. Fabian & Hellgren (1998) define such a property as *interleave insensitivity* of uncontrollable events. These authors proposed an algorithm to detect if a supervisor is interleaving insensitive. If not, the order in which the uncontrollable events occur cannot be recognized if the controller utilized is synchronous.

8. Code generation for PLCs

The scan cycle of a program in the PLC follows the sequence: input read, control logic execution and output write. That synchronous behavior of the PLC forces the outputs to be updated only at the end of the scan cycle. Due to that, the activation of the actuators requires a specific treatment. Looking at the state machine's structure, it may happen that as soon as it finishes the operation, an apparatus may be requested to start a new operation cycle inside the same scan cycle in which the previous operation had just been finished. If this happens during a scan cycle, that variable would assume a low logic level (end of operation) and return to a high logic level (start of operation). However, as the PLC writes the values stored in its internal memory to the physical outputs only when its execution cycle is finished, it does not recognize the process of operation end/start, so that its physical output will be kept active all the time during the scan cycle. In order to avoid that, variables are added to represent the evolving of the plants and guarantee the proper synchronism during the system dynamics. Those variables are called *Plant i*, with *i* varying from *1* to *n*, where *n* is the number of plants modelled in the global system. That variable is enabled every time an apparatus finishes its operation. This procedure guarantees that the apparatus is not requested to start operating again inside the same scan cycle when its operation end is detected. It will be turned on again only in the next scan cycle.

As in the state machine model, for the implementation, the variables b_i represent the uncontrollable events (transitions) while variables a_i (actions) represent the controllable events.

The Ladder code can be split into five blocks, called by a main organizational block in the following order: initialization, inputs, transitions/actions, disabling and outputs. They implement the reduced state machine shown in Figure 11.

8.1 Initialization

The initialization starts the state machine and puts it into its initial state, and enables the controllable event a_1 in order to start the process, as shown in Figure 13. Other variables, such as the ones responsible for uncontrollable events *bi* and the evolution of plants *Plant i*, are disabled.

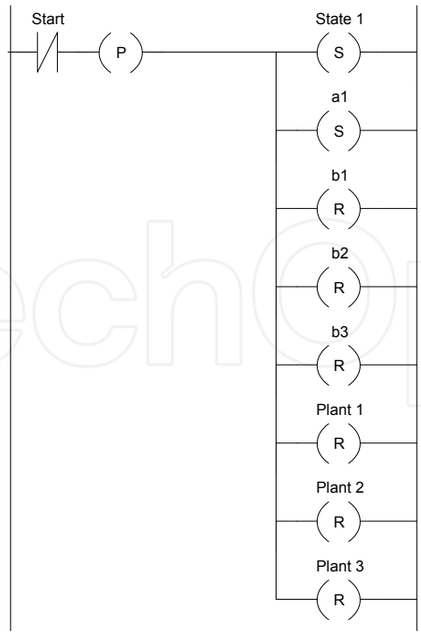


Fig. 13. Initialization

8.2 Inputs

The variables responsible for transitions b_i will be activated in the controller only when an edge rise occurs (from logic level 0 to 1) in the corresponding inputs. Therefore, a pulse detector is required for each input in order to signal the corresponding event, as shown in Figure 14. The variables related to the uncontrollable events are activated in this block. During the program execution, if they result in some action, they will be disabled. If they result only in self-loops where no action is taken, as represented by dashed lines in the state machine, these variables remain active until the moment that some transition which results in an action occurs later.

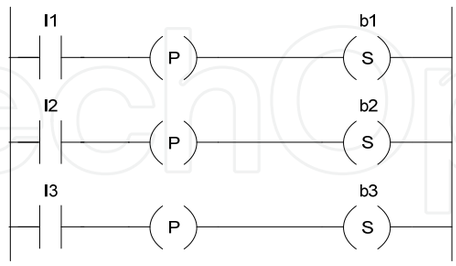


Fig. 14. Inputs

8.3 Transitions/actions

The requirement for a transition to occur is that the state machine must be in a certain state and one or more uncontrollable events that result in some action happens. If these requirements are satisfied, then the next state is activated and the current one is deactivated. The uncontrollable events responsible for the transition are deactivated to avoid occurrence

of the avalanche effect (Fabian & Hellgren, 1998). When an uncontrollable event happens for a certain plant, the variable corresponding to that plant, *Plant i*, is activated in order to avoid that the action corresponding to the start of the operation for that apparatus occurs during the same scan cycle, being treated only in the next scan cycle. This is due to the specific treatment required for the actions, as described previously. The actions, if any, will be activated to allow the corresponding plant to evolve in the same scan cycle or only in the next scan cycle if forbidden to evolve in the current cycle. Figure 15 shows this block for the manufacturing system. Verify the representation of the reduced state machine shown in Figure 11 in order to compare the theoretical model with the practical one. Consider for instance that the state machine is in state *3*. Here, two transitions are possible: due to both events b_1 and b_3 or only due to event b_1. The transition b_1 & b_3 (due to events b_1 and b_3) must always appear first in the Ladder diagram due to this problem of simultaneity (Fabian & Hellgren, 1998). This is because in the case that events b_1 and b_3 occur and the transition only due to event b_1 is implemented first, the latter will be executed in the program and the variable b_1 will be deactivated. In this way, the transition due to events b_1 and b_3 will never happen in practice.

Consider now that the state machine is in state 4 and the transition b_1 & b_2 & b_3 occurs. The state machine remains in the same state and, as all the apparatus finish their operation, no action will be taken in the current scan cycle. However, as the actions a_1, a_2 and a_3 are activated, in the next scan cycle all the apparatus will be turned on again. Thus, such a methodology guarantees that it is necessary to wait only one scan cycle for the enabled actions to be taken.

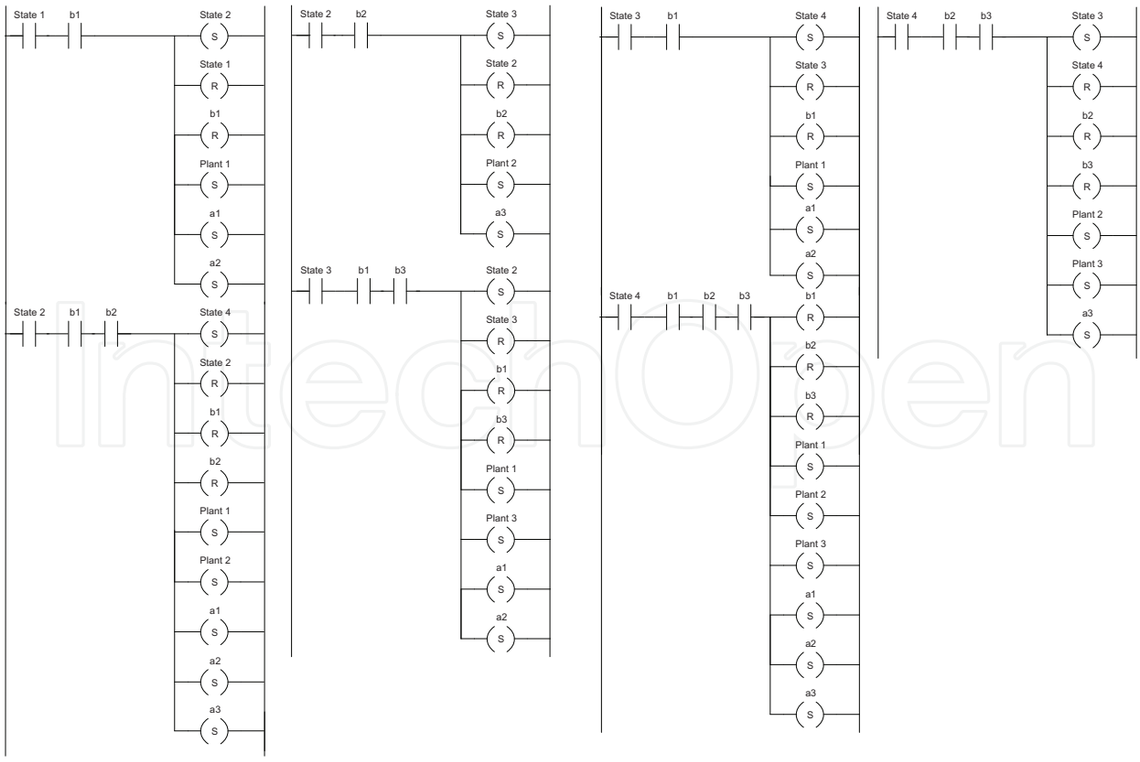


Fig. 15. Transitions/Actions

8.4 Disabling

This block is responsible for disabling the control actions in the machine states. This means that if the state machine reaches a certain state, the actions related to this state are forbidden to occur, being disabled. Figure 16 shows the disabling block for the manufacturing system. The first two rungs on the Ladder code represent the states where the corresponding control actions are disabled. Furthermore, the disabling may occur if some uncontrollable event is signalled but it does not allow an action to be taken (represented by a self-loop with dashed lines in the reduced state machine), so that the start of operation in the corresponding plant is forbidden. In the Ladder diagram, it is enough to disable the action related to a certain *Plant i* when the uncontrollable event *bi* occurs. In other words, when b_1 occurs, it is enough to disable *Plant 1*. When b_2 occurs, it is enough to disable *Plant 2*, and so on. This is because although the event *bi* occurred, it did not effectively generate a transition in the state machine, and therefore, the plant was not disabled, as shown in the remaining rungs of the Ladder diagram.

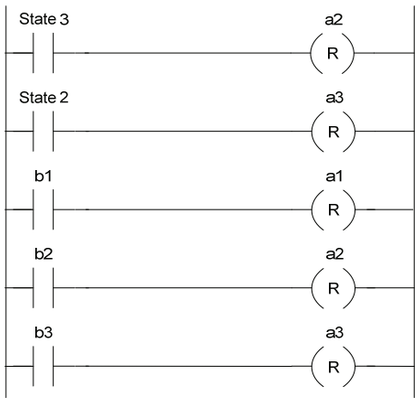


Fig. 16. Disabling

8.5 Outputs

A physical output is activated only if the action related to a controllable event is taken and its corresponding variable, *Plant i*, is not enabled, as shown in Figure 17. If such situation occurs, the coil Q_i which represents the physical output of the PLC will be energized. Yet, at the end of the program, the variables *Plant i* will be disabled in order to return to the initial condition before starting a new scan cycle.

Besides being a simplified implementation model, this solution has the advantage of not restricting more than one uncontrollable event to be treated in the same PLC scan cycle.

In order to have a better understanding of the logic implementation of this supervisory control, consider first that the start of operation is given by the action a_1. After a few PLC scan cycles, the apparatus A1 finishes its operation and the transition due to the uncontrollable event b_1 is generated. According to the reduced state machine, the actions a_2 and a_1 are enabled to occur and the state machine transitions to *state 2*. However, as *Plant 1* finished its operation, only the physical output Q_2 is activated in the same scan cycle. The physical output Q_1 will be activated only in the next scan cycle. Next, the state machine will behave differently depending on which apparatus finishes its operation first. If apparatus

A2 finishes its operation, then transition b_2 will be generated, action a_3 will be taken and the state machine transitions to *state 3*. If apparatus A1 finishes its operation, then transition b_1 will be generated resulting in a self-loop with dashed lines inside *state 2*, because no action is taken. If the PLC identifies the changes in the input signals corresponding to the end of operation in both apparatus A1 and A2 (transitions b_1 and b_2, respectively), then actions a_1, a_2 and a_3 will be taken and the state machine transitions to *state 4*. Similarly, the remaining transitions and actions follow the same dynamics, as illustrated in the reduced state machine model presented in Figure 11.

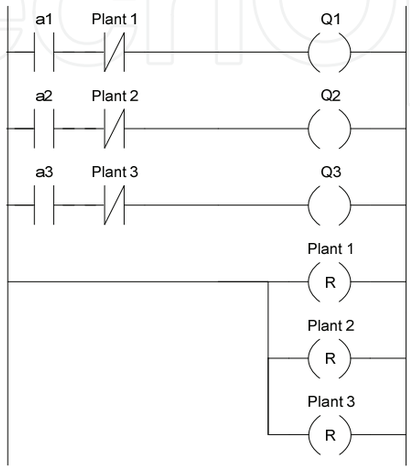


Fig. 17. Outputs

9. Conclusion

The proposed methodology presents a new model to represent supervisors for Discrete Event Systems (DES). The Supervisory Control Theory (SCT) is a convenient methodology to obtain supervisors from simpler models, where automata are useful to represent plants and control specifications so that, starting from simpler models, supervisors may be represented by state machines. In general, the approach based on state machines consists of a model rich in information. It is appropriate to emphasize the importance of the modelling process for the supervisors, because if some control specification changes or a new subsystem is added to the plant, it is necessary to remake the synthesis for the state machine.

The proposed algorithm to transform the automata which represents a supervisor in a state machine allows a reduction in the number of states in the model. That reduced approach can be used as a reference to implement the control system in a data processing unit. It is necessary only to take into account the implementation aspects related to the controller used, based on the constructive aspects of its hardware. In this chapter, solutions were described to avoid the problems usually found when implementing supervisors in synchronous controllers, using a PLC as a target with the source code implemented in the Ladder language.

The example of a manufacturing system demonstrates some aspects related to the optimization in the code size generated resulting in an economy in the non-volatile memory usage and the possibility of treating several events inside the same scan cycle of the

controller. For large systems, this approach results in an improvement of the temporal dynamics of the control when several input signal changes and several actions shall be taken in the same scan cycle, ensuring synchronism and minimizing problems due to communication delays.

10. Acknowledgment

The authors thank the financial support provided by CNPq and FAPESC, and also thank Whirlpool Latin America and the UDESC for their support in the realization of this work.

11. References

Ariñez, J.F.; Benhabib, B.; Smith, K.C. & Brandin, B.A. (1993). Design of a PLC-Based Supervisory-Control System for a Manufacturing Workcell, *Proceedings of the Canadian High Technology Show and Conference*, Toronto, 1993.

Basile, F. & Chiacchio, P. (2007). On the Implementation of Supervised Control of Discrete Event Systems. *In: IEEE International Transactions on Control Systems Technology,* Vol. 15, No. 4, pp. 725 – 739, ISSN: 1063-6536

Cassandras, C.G. & Lafortune, S. (2008). *Introduction to Discrete Event Systems.* 2nd Ed. Springer Science, ISBN-13:978-0-387-33332-8, New York, USA.

Chandra, V.; Huang, Z. & Kumar, R. (2003). Automated Control Synthesis for an Assembly Line Using Discrete Event System Control Theory. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, Vol. 33, No. 2, (May 2003), pp. 284-289, ISSN 1094-6977.

Chen, Y. L. & Lin, F. (2000). Modeling of Discrete Event Systems using Finite State Machines with Parameters. *In: IEEE International Conference of Control Applications,* Anchorage, Alaska, USA.

Fabian, M. & Hellgren, A. (1998). A PLC-based implementation of supervisory control for discrete systems. *In: Proceedings of the 37th IEEE Conference on Decision and Control,* Vol. 3, pp. 3305-3310.

Gaudin, B. & Deussen, P.H. (2007). Supervisory Control on Concurrent Discrete Event Systems with Variables. *In: Proceedings of the IEEE American Control Conference,* pp. 4274-4279, ISSN: 0743-1619, New York, NY, USA.

Harel, D. (1987). Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming,* Vol. 8, No. 3 (June 1987), pp. 231-274.

Hasdemir T., Kurtulan, S., & Gören, L. (2004). Implementation of local modular supervisory control for a pneumatic system using PLC. In: *Proceedings of the 7th International Workshop on Discrete Event Systems (WODES'04)*, pp. 27-31, Reims, France.

Hasdemir, T.; Kurtulan, S.; & Gören, L. (2008). An Implementation Methodology for Supervisory Control Theory. *International Journal of Advanced Manufacturing Technology*, Vol. 36, No. 3, (March 2008), pp. 373-385. ISSN 0268-3768.

IEC-61131-3 (1993). International Electrotechnical Commission. Programmable Logic Controllers – Part 3: Programming Languages.

Lauzon, S. C. (1995). *An implementation methodology for the supervisory control of flexible-manufacturing workcells*, M.A. Sc. Thesis, Mechanical Engineering Dept. University of Toronto, Canada.

Lauzon, S. C.; Mills, J. K.; & Benhabib, B. (1997). An Implementation Methodology for the Supervisory Control of Flexible Manufacturing Workcells, *Journal of Manufacturing Systems*, Vol. 16, No. 2, pp. 91-101.

Leal, A. B.; Cruz, D.L.L.; Hounsell, M.S. (2009). Supervisory Control Implementation into Programmable Logic Controllers. In: *Proc. of the 14th IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 899-905, Mallorca, 2009.

Leduc, R. J. (1996). *PLC Implementation of a DES supervisor for a manufacturing testbeb: an implementation perspective*, M.A.Sc. Thesis, Dept. of Electrical and Computer Engineering, Univ. of Toronto, Canada.

Leduc, R. J. & Wonham W. M. (1995). PLC implementation of a DES supervisor for a manufacturing testbed. In: *Proceedings of 33rd Annual Allerton Conference on Communication, Control and Computing*, pp. 519-528, University of Illinois.

Liu, J. & Darabi, H. (2002). Ladder logic implementation of Ramadge-Wonham supervisory controller, *Proceedings of the 6th International Workshop on Discrete Event System (WODES'02)*, pp. 383-389. ISBN: 0-7695-1683-1. Zaragoza, Spain, October 2002.

Malik, P. (2002). Generating Controllers from Discrete-Event Models. *In : Proceedings of the Modelling and Verifying Parallel Processes Movep 02*. pp. 337-342, Nantes, France.

Manesis, S. & Akantziotis, K. (2005). Automated synthesis of ladder automation circuits based on state-diagrams. *Advances in Engineering Software*, 36, pp .225–233.

Mealy, G.H. (1955). A Method to Synthesizing Sequential Circuits. *Bell Systems Technical Journal.* pp. 1045-1079.

Moura, R. & Guedes, L.A. (2010). Control and Plant Modeling for Manufacturing Systems using Basic Statecharts. *In: Programmable Logic Controller*, Guedes, L.A. (Ed), pp. 33-50, INTECH, ISBN 978-953-7619-63-3, Croatia.

Music, G. & Matko, D. (2002). Discrete event control theory applied to PLC Programming, *Automatica*, Vol 43, 1-2, pp. 21-28.

Noorbakhsh, M. & Afzalian, A. (2007). Implementation of supervisory control of DES using PLC. In: *Proc. 15th Iranian Conference on Electrical Engineering (ICEE)*, Tehran, Iran.

Possan, M.C.P. (2009). *Modelling and Implementation of Supervisory Control Systems Based on State Machines with Outputs* (in Portuguese). Thesis (Master Degree), Santa Catarina State University, Joinville, Brazil.

Queiroz, M.H. & Cury, J.E.R. (2000). Modular supervisory control of large scale discrete-event systems. *Discrete Event Systems: Analysis and Control*. Kluwer Academic Publishers (*Proc. WODES 2000),* Ghent, Belgium, pp. 1-6).

Queiroz, M. H. de & Cury, J. E. R. (2002). Synthesis and implementation of local modular supervisory control for a manufacturing cell, *Proc. of the 6th Int. Workshop on Discrete Event Systems WODES*, pp. 1-6. ISBN: 0-7695-1683-1. Zaragoza, Spain, October 2002.

Ramadge, P.J. & Wonham, W.M. (1989). The control of discrete event systems. *In: Proceedings of IEEE, Special Issue on Discrete Event Dynamics Systems*, Vol. 77, No. 1, pp. 81-98.

Reiser, C.; Da Cunha, A.E.C. & Cury, J.E.R. (2006). The Environment Grail for Supervisory Control of Discrete Event Systems. *In: Proceedings of the 8th International Workshop on Discrete Event Systems,* Michigan, USA, pp. 1-6.

Silva, D. B.; Santos, E. A. P.; Vieira, A. D. & Paula, M. A. B. (2008). Application of the supervisory control theory in the project of a robot-centered, variable routed

system controller, *Proceedings of the 13th IEEE International Conference on Emerging Technologies and Factory Automation – ETFA'08*, pp. 1-6.

Sivolella, L. F. (2005). *Contributions for Supervisors Reduction for Discrete Event Systems* (in Portuguese). Thesis (Master Degree), IME, Rio de Janeiro, Brazil.

Skoldstam, M. & Akesson, K. & Fabian, M. (2008). Supervisor Control Applied to Automata Extended with Variables - Revised. Technical Report. Chalmers University of Technology, Gothenburg, Sweden.

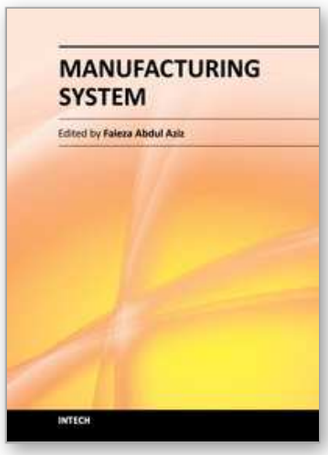
Su, R. & Wonham, W. M. (2004). Supervisor reduction for discrete-event systems, *Discrete Event Dynamic Systems*, Vol. 14, No. 1, pp. 31-53.

Uzam, M.; Gelen, G.; & Dalci, R. (2009). A new approach for the ladder logic implementation of Ramadge-Wonham supervisors, *Proc. of the 22nd Int. Symp. on Information., Commun. and Automation Technologies*, pp. 1-7. ISBN 978-1-4244-4220-1, Bosnia 2009.

Vieira, A.D. (2007). *Method to Implement the Control of Discrete Event Systems with the Application of Supervisory Control Theory* (in Portuguese). PhD. Thesis, UFSC, Florianópolis, Brazil.

Vieira, A. D.; Cury, J. E. R. & Queiroz, M. (2006). A Model for PLC Implementation of Supervisory Control of Discrete Event Systems. *Proc. of the IEEE Conf. on Emerging Technologies and Factory Automation*, pp. 225–232. Czech Republic, September 2006.

Yang, Y. & Gohari, P. (2005). Embedded Supervisory Control of Discrete-Event Systems. *In: IEEE International Conference on Automation Science and Engineering*, pp. 410-415, Edmonton, Canada.



Manufacturing System

Edited by Dr. Faieza Abdul Aziz

ISBN 978-953-51-0530-5
Hard cover, 448 pages
Publisher InTech
Published online 16, May, 2012
Published in print edition May, 2012

This book attempts to bring together selected recent advances, tools, application and new ideas in manufacturing systems. Manufacturing system comprise of equipment, products, people, information, control and support functions for the competitive development to satisfy market needs. It provides a comprehensive collection of papers on the latest fundamental and applied industrial research. The book will be of great interest to those involved in manufacturing engineering, systems and management and those involved in manufacturing research.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen