DESIGNING DEPENDABLE LOGIC CONTROLLERS USING THE SUPERVISORY CONTROL THEORY

Jean-Marc ROUSSEL(1), **Alessandro GIUA**(2)

(1)LURPA, ENS de CACHAN, 61 avenue du président Wilson, F-94235 CACHAN Cedex, France.
jean-marc.roussel@lurpa.ens-cachan.fr

(2)DIEE, Università di Cagliari, Piazza d'Armi, 09123 Cagliari, Italy.
giua@diee.unica.it

Abstract: *In this paper we deal with the problem of designing a controller for a discrete event system. We argue that the classical approach of supervisory control theory (SCT) can be used as an essential step of such a procedure. However, some of the features that make supervisory control an attractive paradigm to solve theoretical problems are often a major source of difficulty in implementing a controller: such is the case, for instance, of the abstraction level usually considered in SCT. We define a method to obtain the correct abstraction level and present a procedure to design a controller using SCT. This approach is applied to a simple but realistic example: an automatic gate.*

Keywords: Logic control systems, Supervisory control theory, Dependabilility, Implementation, Programmable Logic Controller (PLC).

1. INTRODUCTION

The automatic design of logic control devices for discrete event systems is a problem that has received a lot of attention in the last 40 years since the first Programmable Logic Controllers (PLC) appeared on the market. Several ad hoc design approaches and examples are discussed in the literature, but it is difficult to generalize them and human ingenuity is still an essential component of the design procedure although formal approaches for the design of logic controllers go back to the 80's (Crockett, *et al.*, 1987; Zhou, *et al.*, 1992).

Supervisory control theory (SCT) is an approach based on formal languages that, has influenced much of the research on discrete event systems, since the publication of the seminal works by Ramadge and Wonham (1989). A large body of theoretical results have appeared since then, and it was expected that eventually it would provide the basis for a technical transfer to the industry. Unfortunately, there is still a gap between the theoretical development and the number of applications of SCT in the industry, which are still far from what most of the discrete event community expected fifteen years ago.

There was an early attempt to enrich the theory with primitives that should help in the implementation phase, such as the notion of forcible event (Brandin and Wonham, 1994), or to apply it to the design of control logic for PLC (Brandin, 1996). However, ten years later the implementation of supervisory controllers is still an open issue that seems to be attracting the interest of many researchers (Balemy *et al.*, 1993; Akesson and Fabian, 1999; De Queiroz and Cury, 2002; Dietrich, *et al.*, 2002; Fabian and Hellgren, 1998; Flochova, 2003; Hellgren, *et al.*, 2002; Holloway *et al.*, 2000a; Holloway, *et al.*, 2000b; Jeron, *et al.*, 2003; Leduc, *et al.*, 2001; Liu an Darabi, 2002).

We believe that some key features that make SCT such an attractive theoretical framework are often a major source of difficulty in implementing a controller. We mention three of them.

- Firstly, SCT is essentially a theory for "safeness" control, i.e., for restricting the behavior of the plant to satisfy a "safety specification" that specifies which evolutions of the plant should not be allowed (*what the plant should not do*). However, in most

cases we have "liveness specifications" that specify which evolutions of the plant should occur in certain given conditions (*what the plant should do)*.
This appears obvious if one looks at the notion of controllability of an event, that is based on the possibility of preventing an event occurrence. In most cases, however, an event needs also to be forced. This problem was discussed by Brandin and Wonham giving rise to the notion of forcible event (1994) and by Holloway *et al.* (2000b).
Another aspect of the issue is the fact that the notion of optimality in SCT is based on the idea of *maximal permissive control policy.* This is a good paradigm if one wants to enforce a "safety specification" by means of a supervisor. A supervisor is a special controller that in normal conditions should do nothing, whereas it has to intervene when the plant is about to enter a dangerous (or forbidden) state. In most control applications, on the contrary, a control law consists in choosing a particular action without leaving any "choice" if possible (the choice is used only to model the nondeterministic effect of disturbance but no choice among controllable actions is desired. This problem was discussed by Dietrich *et al.* (2001) giving rise to the notion of "implementation" of a supervisor, i.e., of a particular controller that enables only one controllable event at each time.

- In SCT the modelling focus is on the state of the system and its evolution: the open loop system is first described, then the model of the supervisor is kept at a level of detail such that it can be used to derive the model of the controlled system — by standard operators, such as the concurrent composition or the computation of the supremal controllable sublanguage.
  However the purpose of modelling a discrete event system is the design of a control structure to drive its state space trajectories. This problem could be approached in a different way. Given a physical device that we will call the *operating unit*, i.e., the machine in this example, we could model the *control unit* that effectively drives the operating unit according to the orders received from the s*upervisor*, i.e., from the external agent that chooses which sequences of operations are to be executed next. This control unit can be specified in terms of inputs and outputs.
- Finally, while it is true that the SCT provides a general automatic synthesis rule based on standard operators once the plant model and the specification language are given, formalizing a specification may often be an exceedingly difficult task, that can only be solved by trial and error. Personally, while working on a few simple test cases, we observed that it was almost impossible to write a correct specification unless a solution for the control problem was clearly oulined.
  Usually, this problem is solved by choosing a high-level of abstraction. A higher-level description often leads to a SCT problem that is easy to solve but may not be the right level for converting this solution into a control code. Thus the solution obtained at a higher-level may be useless for implementation. This problem was discussed in the literature giving rise to the notion of "hierarchical control" where the solution obtained at a higher-level may help to derive a lower-level controller (Leduc, *et al.*, 2001), or to the notion of "conformance" (Jeron, *et al.*, 2003).

We nevertheless believe that SCT is a keystone approach for automatic design of logic controllers. But it is also necessary to complement it with a formal methodology to obtain plant and specification models at a proper level of abstraction. In this paper, we present a technique to verify if the abstraction-level chosen for the plant model is correct. We also propose a design procedure to obtain a final automaton on which the forcible events are automatically identified.

The paper is structured as follows. In Section 2, we present an example as a case-in-point. In Section 3 we present a procedure to obtain the correct level of abstraction for the plant model. In Section 4, we present a method to express the specifications according to the plant model proposed. In section 5, we give a technique to code the final automaton in an industrial controller. We felt that a background section to present SCT was not necessary: the reader is referred to the book written by Cassandras and Lafortune (1999) for details.

2. DESCRIPTION OF THE CASE STUDY

2.1 Structure of the system

The selected case taken from (Roussel and Faure, 2004) deals with the control of an automatic gate for a car park. The overall system could be partitioned into two parts: the plant to be controlled and the control unit (figure 1). The boundary between the two parts is imposed by the technology, and more precisely, by the choice of inputs and outputs of the industrial controller. From a technical point of view, the inputs and the outputs of the control unit are electrical signals (which can be modeled by booleans) from sensors or for actuators.

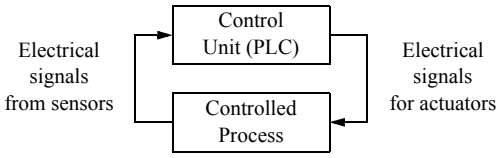

Fig. 1 Decomposition of the system

In this case study, the plant was composed of several elements:

- a gate with 2 limit switches to indicate when the gate is fully open or fully closed,
- an electrical motor with 2 contactors to control the direction (one per direction),
- a receiver for the user's remote controls,
- a sensor to detect the presence of a vehicle in front of the gate.

This choice of technology imposes the inputs and the outputs of the Control Unit (figure 2).

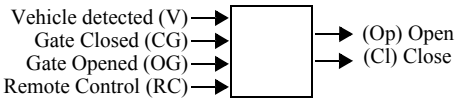

Fig. 2 Inputs and outputs of Control Unit (PLC)

2.2 Control specifications

The desired behaviour of the plant may be expressed by the set of specifications given hereafter in plain natural language. Among these 7 specifications, the first three are related to vivacity requirements (what must be done to perform the expected task). Specification P4 expresses a safety requirement. Specifications P5 and P6 express constraints coming from actuators and the last one is an assumption on the correct operation of the sensors (the problem of sensors monitoring is not dealt with in this study).

P1 When the remote control is activated, the gate opens.
P2 When the gate is open with no request from the user or no detection of a car, the gate closes.
P3 While the gate is not totally closed, the detection of a car causes the reopening of the gate.
P4 The gate must never be simultaneously controlled to open and to close.
P5 An open gate can not be controlled to open.
P6 A closed gate can not be controlled to close.
P7 The gate is never simultaneously open and closed.

2.3 Problems to solve

The code implements a procedure to compute the value of all output signals (on/off) as a function of the state of the controlled system. This state cannot be directly observed but can be reconstructed by means of the available sensor signals if a model of the plant is inserted into the controller. In this approach we model the system by a simple automaton whose change of state is triggered by the occurrence of particular events corresponding to the rising or falling edge of an input/output signal. This model is constructed following the SCT approach.

Notations:
- Parallel composition is denoted by $G = G_1 \| G_2$.
- Supremal controllable sublanguage of H based on G is denoted by $S = Supcon(H, G)$

3. DESIGN OF THE PLANT MODEL

We consider that this step is the most difficult task and that it requires particular care. The desired model must describe the behaviour of the plant with an abstraction level adapted to our objective: the definition of the system control laws.

In the examples presented in the literature the level of abstraction is often too high to allow a correct definition of the control code. A common problem consists in the fact that there is not a clear separation between the model of the plant and the model of controller: the plant model in fact implicitly contains significant aspects that belong to the controller.
To avoid this problem, we propose the following procedure.

3.1 Methodology

The procedure we propose is composed of four steps:

1 *Identification of the boundary of the model*. To obtain it, it is necessary to partition the global model in two parts: the plant and the control unit (figure 1). The boundary of each part is imposed by the choice of inputs and outputs of the industrial controller.

2 *Definition of the list of events*. To model the behaviour of a boolean variable with events, it is necessary to associate 2 events to each boolean (Zaytoon and Carré-Ménétrier, 2001): one for the change from 0 to 1 (rising edge) and one for the change from 1 to 0 (falling edge).
As the values of the inputs are imposed by the plant, the corresponding events are *uncontrollable*.
As the values of the outputs are imposed by the controller, the corresponding events are *controllable*.
Only events that correspond to input/output should be used to define the model of the plant at step 3.

3 *Definition of the behaviour of the plant.* To avoid a model with too high an abstraction level, we suggest to begin with a model for each element (an elementary component) without synchronisation. In a second step, the elementary models are composed by means of synchronous product. The overall result of all proposed synchronisations must be graphically controlled to manually detect inconsistency.

4 *Test for absence of control parts in the plant model*.
From a technical point of view, the plant has no means to restrict the behaviour of the control unit. The control unit could change the values of its outputs as one likes. The proposed plant model must integrate this feature. To test this point, it is necessary to analyse each state of the plant model in order to verify if the evolution of each output is possible in the current state. If such it is not the case, a self-loop with the forgotten event must be added.

This step is very important because these errors are easy to make and they have important consequences.

3.2 Application to the case study

In section 2, we outlined how steps 1 and 2 can be carried out for the considered example. To each boolean B, we have associated 2 events (">B": change from 0 to 1, "<B": change from 1 to 0).

The model of the plant is given by the 4 automata presented on figure 3[1] :

- *Gate:* the gate has 3 states (GC: Gate fully Closed, GO: Gate fully Opened, GhO: Gate half-Opened). The changes of states can be directly detected thanks to the limit switches.
- *Motor:* the motor has 3 states (Stop, Open, Close). The change of states is imposed by control unit by the changes of values of outputs. When the motor is active, it is impossible to change the direction because of the presence of security element in the electrical circuit.
  Interactions between gate and motor are included in this model. The position of the gate can change only if the motor is activated in the correct direction.
  The selfloop with ">op", "<op", ">cl" or "<cl" events are necessary to avoid restricting the behaviour of the most permissive control system.
- *Vehicle Sensor:* There are only 2 states (Vehicle detected, No Vehicle detected).
- *Remote control:* There are only 2 states too.

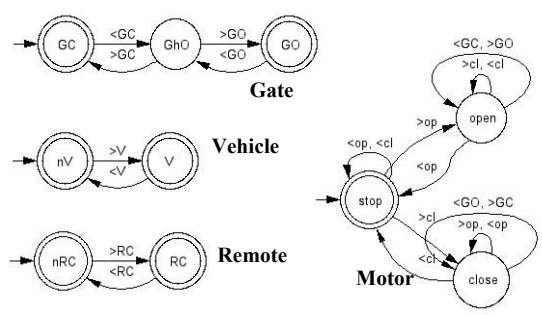

Fig. 3 Plant Model of the automatic gate

The model of the plant is obtained as follows[2]:

$$\text{Plant} = (\text{Gate} \parallel \text{Motor} \parallel \text{Remote} \parallel \text{Vehicle})$$

4. DESIGN OF THE CONTROL MODEL

To write specifications, interactions between the control unit and the controlled process, should be clearly expressed. In the literature, this aspect is only broached during the implementation step. We think that this point must be precised earlier.

1. The graphical representation of automata was obtained automatically with Graphiz (for details: http://www.research.att.com/sw/tools/graphviz/)
2. The composition was obtained with umdes (for details: http://www.eecs.umich.edu/umdes/)

4.1 Modelling of the control unit

From a technical point of view, a unit control could be considered as an infinitely reactive system. It can distinguish all input events and can always calculate all the consequences of each event. We propose to model it with an automaton with 2 states (cf. figure 4) — Read: Data acquisition, Exe: Program Execution. The change from Read to Exe depends on an uncontrollable event. In state "Exe", one or more controllable events could be sent. The change from Exe to Read is conditioned by a controllable event "end". This event signals the end of the program execution. This event is used to simplify the specification written.

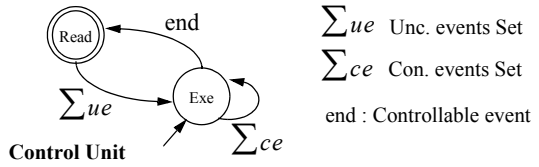

Fig. 4 Model of the control system: generic part

This model must be completed with models of inputs and outputs. We propose to model each element by an automaton with 2 states (one for each value of the boolean). For an output, the initial state corresponds to the 0 value. For an input, the initial state must be fixed according to hypotheses taken for the plant model.

We denoted G the automaton, obtained as follows:

$$G = (\text{Plant} \parallel \text{Control Unit} \parallel \text{Inputs} \parallel \text{Outputs})$$

This automaton G depicts the behaviour of the automatic gate without any control law. This behaviour must be restricted to obtain the only desired part S. We must now define the H automaton with which:

$$S = Supcon(H, G)$$

4.2 Expression of the specification

The specification could be separated in two parts:

- the "safety specification" that specifies which evolutions of the plant should not be allowed (*what the plant should not do*).
- the "liveness specifications" that specify which evolutions of the plant should occur in certain given conditions (*what the plant should do)*.

In several cases, to satisfy a "safety specification" it is not sufficient to prevent an event occurrence. It might be necessary to force a controllable event to appear. For example, for the specification P5, it is necessary:

- to prevent the event occurrence of ">op" (the motor starting) when the gate is fully opened,

- but also to force the event occurrence of "<op" (the motor stop) at the end of the gate opening.

With the proposed model for the Control Unit, we have a global solution to express this type of specifications. We suggest to prevent the "end" event in all dangerous states (cf. figure 5). Each "safety specification" could independently be expressed and composed together after.

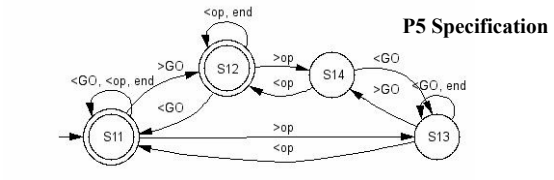

Fig. 5 Safety specification between inputs and outputs

To express "liveness specifications", it is often necessary to force starting events and prevent stop events. We propose to use the same approach. In our tests, we have noted that the "liveness specifications" could not be express independently (cf. figure 6).

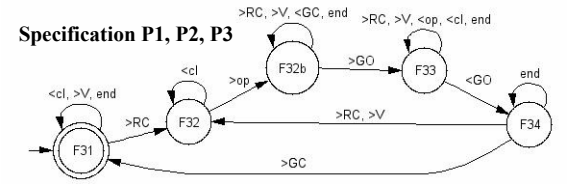

Fig. 6 All liveness specifications

The proposed model for the control unit, permits to link together several controllable events, between an uncontrollable event and the "end" event. The obtained control law must be stable for each output. To obtain this result, it is necessary to prevent two changes of the same output, as shown in figure 7.

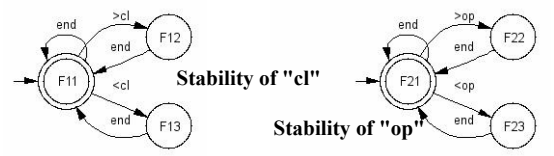

Fig. 7 Stability of the outputs

4.3 Application to the case study

For the case study, the complete specification (H) is composed of 8 automata. The final automaton S ($S = Supcon(H, G)$) is composed of 57 states and 85 transitions. For sake of brevity, these automata are not given in this paper. Graphical and textual descriptions of them can be obtained at http://www.diee.unica.it/~giua/IFAC05.

5. IMPLEMENTATION

To be correctly implemented in the control unit, the automaton S must be *deterministic*. Namely, this means that it does not contain any state from which the "end" event and another different controllable event are both enabled. The detection of these errors is immediate by a simple inspection of the graph. If S is not deterministic a new specification must be introduced to remove this undesirable degree of freedom.

The automaton S could also include states from which two (or more) controllable events are both enabled. This case, that we call *choice*, is different from the non-determinism previously described. It simply implies that there exist two (or more) different ways of implementing the control law. In this case, it is sufficient to implement only one solution.

For the case study, the automaton S contains 2 nondeterministic states but no choice. To remove the undesirable nondeterminism we introduce a new specification and use it to compute a new automaton S'. The new specification H' and the automaton S' can be found in the above mentioned web page.

Standard techniques could be used to transform a deterministic and choice-free automaton into a Mealy machine than can be directly implemented into the control unit. For the case study, the Mealy machine contains only 16 states and 42 transitions (figure 8).

6. CONCLUSIONS

We have shown a method to use SCT to obtain an implementation for a control unit. This method is based on a specific model for unit controls. This model permits to determine which controllable events must be forced and to control if the final automaton contains non-deterministic parts. In this paper we have briefly outlined the proposed approach and have applied it to the design of a controller for an automatic gate.

There are several issues that deserve to be further explored. While working on a few simple test cases, we have observed that it was almost impossible to write a correct specification of liveness unless a solution for the control problem is clearly oulined. We propose to complete this approach with other techniques, like task specifications as proposed in (Holloway, *et al.*, 2000b).

REFERENCES

Akesson, K., M. Fabian, (1999). Implementing supervisory control for chemical batch processes. In: *Proceedings of the IEEE Conf. Control Applications,* Kohala Coast, Hawaii, USA, 1999, pp. 1272-1277.

Balemi, S., G.J. Hoffmann, P. Gyugyi, H. Wong-Toi, G.F. Franklin, (1993). Supervisory control of a rapid thermal multiprocessor. In: *IEEE Trans. on Automatic Control*, **38-7**, 1040-1059

Brandin, B.A., (1996). The real-time supervisory control of an experimental manufacturing cell. In: *IEEE*

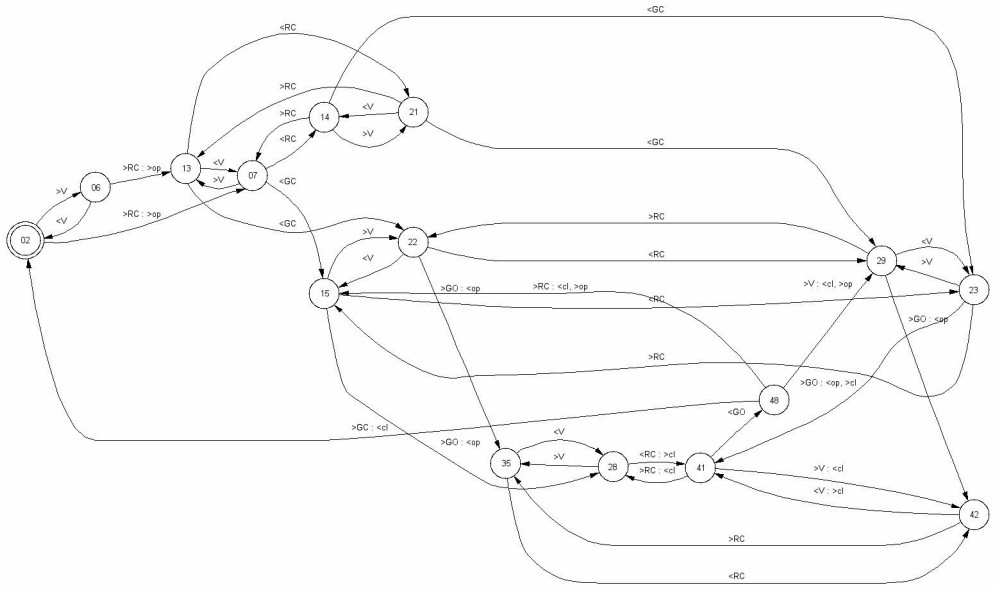

Fig. 8 Mealy machine implemented into Control Unit

Trans. on Robotics and Automation, **12-1**, 1-14.

Brandin, B.A., W.M. Wonham (1994). Supervisory control of timed discrete-event systems, In: *IEEE Trans. on Automatic Control*, **39-2**, 329-342.

Cassandras, C.G., S. Lafortune (1999). *Introduction to Discrete Event Systems,* Boston, Kluwer Academic Publishers.

Crockett, D., A. Desrochers, F. Dicesare, T. Ward (1987). Implementation of a Petri Net Controller for a Machining Workstation, In: *Proceedings of the IEEE Conf. Robotics and Automation,* Raleigh, North Carolina, USA, April 1987, pp. 1861-1867.

De Queiroz, M.H., J.E.R. Cury (2002). Synthesis and implementation of local modular supervisory control for a manufacturing cell, In: *Proceedings of 6th Int. Workshop on Discrete Event Systems*, Zaragoza, Spain, October 2002, pp. 377-382.

Dietrich, P., R. Malik, W.M. Wonham, B.A. Brandin (2002). Implementation Considerations in Supervisory Control. In: Synthesis and Control of Discrete Event Systems, B. Caillaud, X. Xie, Ph. Darondeau and L. Lavagno (Eds.), pp. 185-201, Kluwer.

Fabian, M., A. Hellgren (1998). PLC-based Implementation of Supervisory Control for Discrete Event Systems, In: Proceedings of 37th IEEE Conf. on Decision and Control, Tampa, Florida, USA, December 1998, pp. 3305-3310.

Hellgren, A., B. Lennartson, M. Fabian (2002). Modelling and PLC-based implementation of modular supervisory control Discrete Event Systems, In: *Proceedings of 6th Int. Workshop on Discrete Event Systems*, Zaragoza, Spain, October 2002, pp. 371-376.

Flochova, J. (2003). A Petri net based supervisory control implementation. In: *Proceedings of IEEE Int. Conf. on Systems, Man and Cybernetics,* Washington, DC, USA, October 2003, pp. 1039-1044.

Holloway, L.E., A. Callahan, J. O'Rear, X. Guan (2000a). Spectool: Automated Synthesis of Control Code for Discrete Event Controllers. In: *Proceedings of 5th Int. Workshop on Discrete Event Systems*, Ghent, Belgium, August 2000, pp. 383-389.

Holloway, L.E., X. Guan; R. Sundaravadivelu, J. Ashley JR. (2000b). Automated synthesis and composition of taskblocks for control of manufacturing systems. In: *IEEE Trans. on Systems, Man and Cybernetics, Part B*, **30-5**, 696-712.

Jeron T., H. Marchand, V. Rusu, V. Tschaen (2003). Ensuring the conformance of reactive discrete-event systems using supervisory control. In: *Proceedings of 42nd IEEE Conf. on Decision and Control*, Maui, Hawaii, USA, December, 2003, pp. 2692-2697.

Leduc, R.J., B.A. Brandin, W.M. Wonham, M. Lawford (2001). Hierarchical interface-based supervisory control: serial case. In: *Proceedings of 40th IEEE Conf. on Decision and Control*, Orlando, Florida, USA, December, 2001, pp. 4116-4121.

Liu, J., H. Darabi (2002). Ladder logic implementation of Ramadge-Wonham supervisory controller. In: *Proceedings of 6th Int. Workshop on Discrete Event Systems*, Zaragoza, Spain, October 2002, pp. 383-389.

Ramadge, P.J., W.M. Wonham (1989). The control of discrete-event systems. *Proceedings of the IEEE*, **77**, 81-97

Roussel, J.-M., J.-M. Faure (2004). Designing dependable logic controllers using algebraic specifications. In: *Proceedings of 7th Int. Workshop on Discrete Event Systems*, Reims, France, 22-24 September, 2004, pp. 313-318.

Zaytoon, J., V. Carré-Ménétrier, (2001), Synthesis of control implementation for discrete manufacturing systems. In: *International Journal of Production Research*, **39-2**, 329-345.

Zhou, M.C., F. Dicesare, D. Rudolph (1992). Design and Implementation of a Petri Net Based Supervisor for a Flexible Manufacturing System, In: *Automatica*, **28-6**, 1199-1208.