ISSN 0005–1144
ATKAAF 43(1–2), 21–28 (2002)

Gašper Mušič, Drago Matko

Discrete Event Control Theory Applied to PLC Programming

UDK 681.518.5:004.4
IFAC IA 4.2;2.8

Professional paper

In the paper we present an implementation method for controllers designed by discrete event control theory. Controllers are implemented by standard programmable logic controller and IEC 61131-3 compliant programming software.

Key words: discrete event systems, programmable controllers, supervisory control

1 INTRODUCTION

Discrete event control theory was introduced in a series of papers by Ramadge and Wonham and their co-workers [1, 2, 3]. There the control theory for discrete event systems was formulated through the concept of supervisory control. The controllability conditions were studied and the optimal control was formulated in the sense of maximally permissive controller, which does not restrict the behaviour of the system more than necessary. Modular approach was also presented.

Several other researchers continued the development of the basic theory with the studies of observability, normality, hierarchical control, and the number of published theoretical papers is substantial.

On the other hand, the reports on application of the supervisory control are rather rare. One of the first among them is the paper of Balemi et al. [4] describing the application in the semiconductor industry. Other papers report the applications in protocol conversion, transaction execution in database systems, automated highway systems, etc. Application of supervisory control theory to control synthesis for an assembly line is reported in [5].

In this paper we investigate the application of supervisory control theory to design a control program that will be implemented by a programmable logic controller (PLC). PLCs are one of the most commonly used implementation platforms and the majority of industrial automation solutions are currently based on PLCs.

The traditional approach to PLC programming is mainly concerned with the application of different programming languages. Severe efforts to unify the diverse programming techniques resulted in the international standard IEC 61131-3 [6]. The standard defines common elements and specific syntax of four programming languages (Instruction List, Structured Text, Function Block Diagram, Ladder Diagram) plus additional language for structuring the code (Sequential Function Chart).

While many efforts have been taken to improve the programming techniques, less has been done on the design of the control logic itself. The key to the success of the controller program lies in the correctness of the underlying logic. An approach commonly used in the computer science is to verify the program code by formal techniques. Recently, several papers have been published on the verification techniques for the PLCs, a survey may be found in [7]. The main idea of the verification approaches is to verify the correctness of the code that has been programmed in the usual way.

What we investigate here is the complementary approach, i.e. a systematic design procedure that would result in an automatically generated code, correct by design. The procedure consists of several design phases, from system modelling, followed by specification of control requirements to control synthesis and implementation. The supervisory control theory is applied in the control synthesis phase while implementation is performed within the IEC 61131-3 compliant programming environment. A similar approach is reported in [5]. The main difference is in implementation where we use a programmable logic controller instead of a personal computer and a dedicated software.

The paper is structured as follows. In the next section a brief overview of the discrete event control theory is given. Then the proposed design procedure is described and a simple application example is presented at the end of the paper.

2 DISCRETE EVENT CONTROL THEORY

The supervisory control theory was introduced in [1] as an attempt to build up a control theory for discrete event systems. The process that is controlled is modelled as a deterministic finite state automaton [8], interpreted as a generator of a formal language $L(G)$.

2.1 Formal languages

One of the possible representations of dynamics of discrete event system is to write possible event sequences the system can generate. Let symbol σ represent an event in the system. The system can generate a finite set of distinguished events. Related symbols form a nonempty finite set Σ, also named an alphabet of the system.

A *string*, also called trace or word is a finite set of symbols, e.g. $s=\sigma_1\sigma_2\sigma_3\sigma_4$. Length of a string $|s|$ is a nonnegative integer, equal to the number of symbols forming the string. A string with length 0 is an empty string denoted as ε. The concatenation of strings: $s=s_1s_2$ is the string of symbols of s_1 followed by the string of symbols of s_2. The empty string is the identity element for the operation of concatenation: $\varepsilon s = s\varepsilon = s$.

The set Σ^* is a set of all finite strings of elements of Σ including the empty string ε. The *language* is defined as a subset of Σ^*. Standard set operations union, intersection, set difference, complement (relative to Σ^*) are defined over the languages of Σ^*.

If $s't=s$ and $s, s', t \in \Sigma^*$, then s' is a prefix of s. A prefix closure of a language $L \subseteq \Sigma^*$ is defined as $\overline{L} = \{s \in \Sigma^*; \exists t \in \Sigma^*, st \in L\}$. $\overline{L}$ is again a language: $L \subseteq \overline{L}$. L is prefix closed if $L = \overline{L}$.

2.2 Automaton as a generator of a formal language

A deterministic automaton is defined as a five--tuple

$$G = (X,\ \Sigma,\ \delta,\ x_0,\ X_m) \tag{1}$$

where X is a set of states, Σ is a set of symbols, associated with events in the automaton. $\delta : X \times \Sigma \to X$ is a state transition function of G and is in general a partial function on its domain. x_0 is the initial state of the automaton G, X_m is a subset of states, called a set of marker states.

The set of marker states X_m enables to designate a set of states with special meaning, e.g. states where different tasks in the system are completed. It is a common request that a system is capable of returning to one of the marker states at any moment.

The state transition function δ of a generator G is extended from $X \times \Sigma$ to $X \times \Sigma^*$ in a recursive manner: $\delta(x,\varepsilon)=x$, $\delta(x,s\sigma)=\delta(\delta(x,s),\sigma)$, for $s \in \Sigma^*$ and $\sigma \in \Sigma$ whenever $x'=\delta(x,s)$ and $\delta(x',\sigma)$ are defined.

A generator G may be represented as a directed graph with a set of nodes X, where there exist a connection $x \to x'$, labeled σ, for every triple (x,x',σ), such that $x'=\delta(x,\sigma)$. The generator G is interpreted as a device, which enters state x_0 when switched on and then changes its state following the graph. A symbol is generated at every transition. The transitions between states occur spontaneously, asynchronously, and in an instant. The described model does not include any event selecting mechanism nor time.

The language generated by the generator G is

$$\mathcal{L}(G) = \{s \in \Sigma^*;\ \delta(x_0,s) \text{ is defined}\} \tag{2}$$

The language marked by G is

$$\mathcal{L}_m(G) = \{s \in \mathcal{L};\ \delta(x_0,s) \in X_m\} \tag{3}$$

$\mathcal{L}(G)$ is interpreted as a set of all finite event sequences that may occur in the automaton. $\mathcal{L}_m(G)$ is a subset of event sequences that end in marker states. It is not required that the generator stops there, it may continue with the symbol generation. If G is a generator, the language $\mathcal{L}(G)$ is prefix closed: $\mathcal{L}(G)=\overline{\mathcal{L}(G)}$; which is not always true for $\mathcal{L}_m(G)$.

If the set of states X is finite, G is a finite automaton. The class of finite automata is particularly interesting for implementation. The class of languages that may be represented by finite automata is the class of regular languages.

2.3 Composition of automata

Complex models may be built by composing simpler automata. Two basic composition operations exist: product, denoted by $\times$, and parallel composition, denoted by $\|$. Operation $\|$ is often called synchronous product and $\times$ is sometimes called totally synchronous product.

Denote a pair of automata as

$G_1=(X_1,\Sigma_1,\delta_1,x_{01},X_{m1})$ and $G_2=(X_2,\Sigma_2,\delta_2,x_{02},X_{m2})$.

In the product $G_1 \times G_2$, transitions in the two automata must always be synchronised on a common event, that is an event in $\Sigma_1 \cap \Sigma_2$. Other events cannot occur at all. In the parallel composition $G_1 \| G_2$, the two automata are only synchronised on common events, while other events may occur whenever possible. The composed automaton is

$$G_1 \| G_2 = Ac(X_1 \times X_2,\ \Sigma_1 \cup \Sigma_2,\ \delta,\ (x_{01},x_{02}),\ X_{m1} \times X_{m2}) \tag{4}$$

where Ac denotes the accessible part of the automaton, i.e. automaton where only the states that can be reached from its initial state are kept. The state transition function in the composed automaton is defined as:

$$\delta((x_1,x_2),\sigma) = \begin{cases} \delta_1(x_1,\sigma),\delta_2(x_2,\sigma) & \text{if } \delta_1(x_1,\sigma) \text{ and } \delta_2(x_2,\sigma) \text{ defined} \\ \delta_1(x_1,\sigma),x_2 & \text{if } \delta_1(x_1,\sigma) \text{ defined, } \sigma \notin \Sigma_2 \\ x_1,\delta_2(x_2,\sigma) & \text{if } \delta_2(x_2,\sigma) \text{ defined, } \sigma \notin \Sigma_1 \\ \text{undefined} & \text{otherwise} \end{cases} \quad (5)$$

When $\Sigma_1 = \Sigma_2$, all events must occur synchronously therefore $G_1 \| G_2 = G_1 \times G_2$.

2.4 Supervisory control

The supervisory control concept deals with a discrete event system whose behaviour is restricted by an external controller called supervisor.

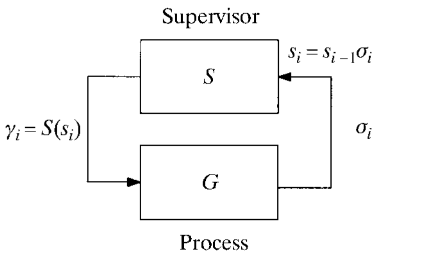

Fig. 1 Feedback loop of supervisory control

The supervisor (Figure 1) does not uniquely determine the next event to occur in a system; it merely monitors events generated by the system and determines the set of allowable events that can occur at any instant (γ_i in Figure 1). In this way the supervisor actually intervenes only in cases when some undesired process behaviour is about to take place. The control effect is restricted to prevention of certain events in the system.

The supervisor is computed based on the »open--loop« system model that can be given as a finite automaton. Commonly, the open-loop model (G in Figure 1) represents a locally controlled process; it therefore includes the process and the local control mechanisms (Figure 2).

Supervisory controller action is to define a set of enabled events that are permitted to occur with regard to the sequence of the past events. The events that are not included in the set of enabled events are disabled. Not every event can be disabled. The set of events is therefore divided into a subset of controllable and a subset of uncontrollable events: $\Sigma = \Sigma_c \cup \Sigma_u$, $\Sigma_c \cap \Sigma_u = \emptyset$.

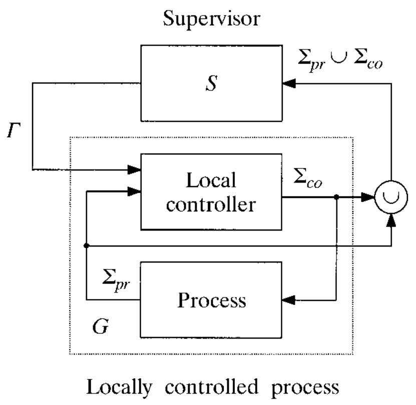

Fig. 2 Supervisory control

The uncontrollable events are either generated by the process itself and cannot be controlled or must not be blocked by an external agent due to the safety of other requirements. The process events are denoted by Σ_{pr} in Figure 2 while Σ_{co} denotes the events generated by the local controller. It follows that $\Sigma_{pr} \subseteq \Sigma_u$ and $\Sigma_c \subseteq \Sigma_{co}$.

Supervisory control synthesis methods enable the computation of the supervisor that is maximally permissive. That means the resulting closed-loop system meets the demands about the system behavioural restrictions, while the supervisor never tries to block an uncontrollable event and at the same time does not restrict the system more than necessary. The key issues are the concept of controllability [1] and a concept of supremal controllable sublanguage [2].

The main problem of the related design procedures is the large number of states in automata that represents real systems. The testing of controllability and calculation of supremal controllable sublanguage involve searching over the state space of an automaton. The so-called state explosion may be avoided by hierarchical and modular approaches.

3 SUPERVISORY CONTROL SYNTHESIS OF LOGIC CONTROLLER

The local controller in Figure 2 is supervised by the supervisor, which limits its behaviour. This supervisor is unnecessary in the case when the behaviour of the controller is defined as a sublanguage of the supervised behaviour of the system. This brings us back to the original setup of Ramadge and Wonham (Figure 1) but with the different interpretation of the controller S – a logic controller.

The logic controller activity is not limited to monitoring the process events and providing a set of enabled events. It is extended to actively trigger events that result in the state changes of the actuating elements of the process. The controller therefore actively drives the process through a desired event sequence.

Note that this does not present any contradiction with the original supervisory control framework. In the closed loop of the supervisory control there is no implication on the causal order of events or about the event triggering mechanism. The only requirement is that the events in the process and the controller must be synchronised [9]. Since the controller is designed to generate a subset of the allowed behaviour, all the control specifications are fulfilled.

The proposed design procedure is schematically shown in Figure 3 and described in the following sections.

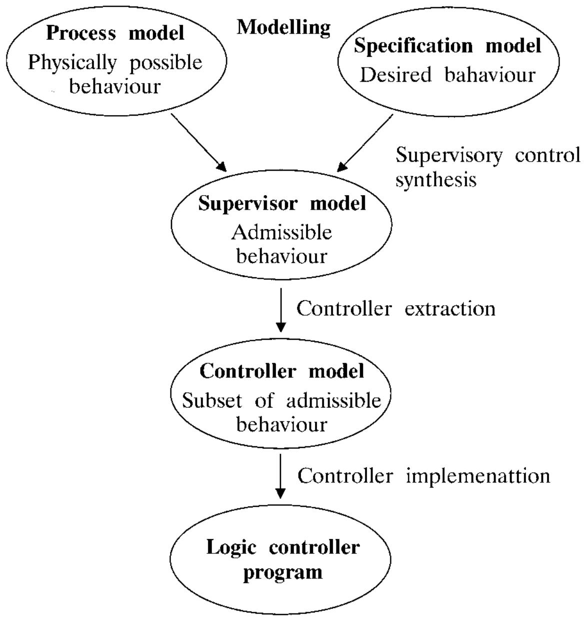

Fig. 3 Design procedure

3.1 Modelling

Modelling of the process aims to capture all the possible event sequences in the system that has to be controlled. Looking from the viewpoint of the logic controller, process events are changes in the states of sensing and actuating elements. Due to physical setup of the system, only a subset of all sensors and actuators are directly related. According to this relation the process is decomposed into a set of subsystems that we call devices.

Every device is modelled independently as a finite automaton. Only those event sequences are included in the model that are physically possible. E.g., a pneumatic piston equipped with two limit switches at both ends and an electro-pneumatic valve on the pressure supply may not generate a sequence of two switch state changes without a change of the state of the valve in between.

In general, change of the state of an actuator is physically possible at any moment (events Σ_{co} in Figure 2) while sensor events (denoted by Σ_{pr} in Figure 2) depend on the state of the actuators.

Even if certain sequence of actuator events is physically possible it may not be allowed (e.g. for safety reasons). This requirement, however, is not part of the process model but belongs to the control specification.

The complete model of the process may be obtained by parallel composition of device models. The number of states in such a model increases exponentially with the number of devices so modular approach [3] to the control design must be adopted.

3.2 Specifications

Control specifications define the desired behaviour of the process. This is related to desired system operation, e.g. completing of tasks in the specified order, and additional measures to assure safety, co-ordinate subprocesses, prevent deadlocks, etc.

Therefore the specifications are split up in two parts. The first part deals with the sequential specification and defines prescribed order of tasks. It consists primarily of events related to sensor readings. The second part involves prevention of undesired behaviour. It is composed of the so-called interlocks, and includes primarily events related to actuator states.

Every specification is modelled as a single automaton that generates desired event sequences. Similarly as with the process model, different parts of the control specifications are combined with parallel composition of the related automata. When modular approach is used, only specifications related to a single device are used at a time.

3.3 Supervisory control synthesis

Once the model of the process and the specification model are obtained the supervisor is designed in a formal way, according to procedures described in [1, 2]. The result is an automaton that may be used to implement the supervisor. In our approach, we do not implement the supervisor directly but extract a logic controller first.

3.4 Logic controller extraction

The logic controller must generate a subset of the behaviour that is allowed by the supervisor – the admissible behaviour. It is extracted by inspecting the state transition graph of the automaton model of the admissible behaviour.

In every node of the graph, a set of enabled controllable events is determined. One of them is chosen as the preferred event. The criteria the preferred event is chosen upon depend on the designer. One criterion may be, e.g., to try to complete the tasks (reach the marker states) in the minimal number of steps.

The table of chosen controllable events is built up and stored for later use in the implementation phase. The arcs related to chosen controllable event and all the uncontrollable events are retained while other may be deleted from the state transition graph. Note that this is not absolutely necessary since the choice among the events is based on the control table.

3.5 Logic controller implementation

The result of the controller extraction is a finite state model of control logic and the table of events that should be generated in its states. This model may be coded in a program of a logic controller in a rather straightforward way. The standard IEC 61131-3 provides different programming languages that may be used for this purpose. Since the required set of operations in our case is very basic, the choice of programming language is more or less a matter of programmer's preference. Here we show the implementation by ladder diagram, which is one of the classical tools for programming logic controller.

Beside the programming languages, the standard IEC 61131-3 provides also means for structuring the code in the modular and hierarchical way. When using modular approach as suggested above, the so-called function blocks may be of advantage. For every device, a function block is defined. Such a block in a general form is shown in Figure 4.

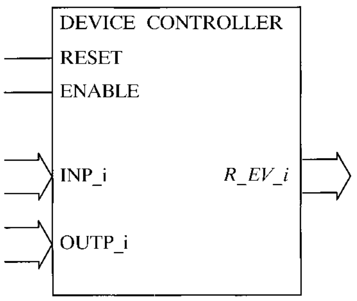

Fig. 4 Function block representing controller for a single device

Inputs to the function block are current states of all the sensors and actuators related to particular device. Additional inputs are »Reset«, which returns the implemented automaton in the initial state, and »Enable« which controls whether state transitions inside the block are allowed or not.

The body of the function block is implemented by ladder diagram consisting of several sections. Typical program sections are shown in Figures 5 to 8.

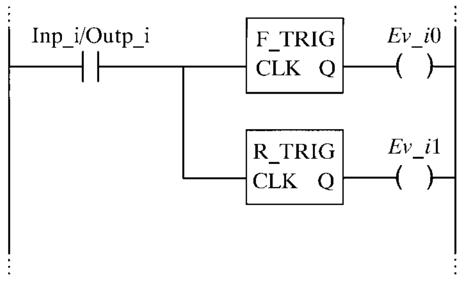

Fig. 5 Event detection section of the ladder diagram

Figure 5 shows a part of the event detection section, where events are expressed as transitions of the states of sensors and actuators.

Figure 6 shows the reset section, where the initial state of the automaton is established when a »Reset« input equals 1.

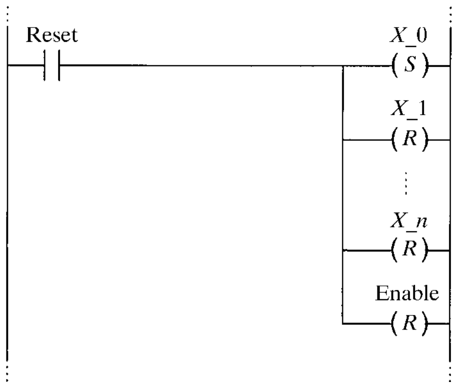

Fig. 6 Reset section of the ladder diagram

Figure 7 shows a part the state transition section, where the actual state transition logic is programmed. The »Enable« signal is reset after any successful state transition, which assures only one state transition occurs in a single program cycle and no output event is lost. In case of simultaneous events the priority of state transitions is established by correct sequencing of the rungs of the ladder diagram. The top-most rung corresponds to the state transition with the highest priority.

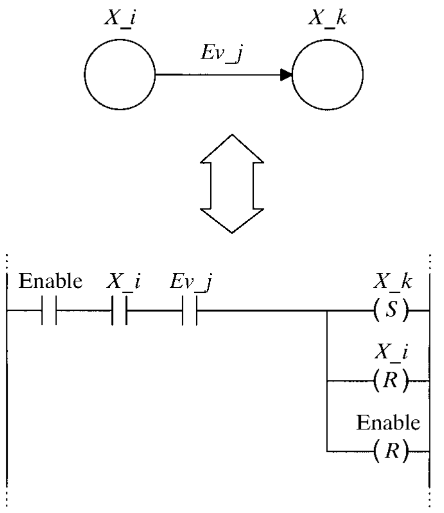

Fig. 7 State transition section of the ladder diagram

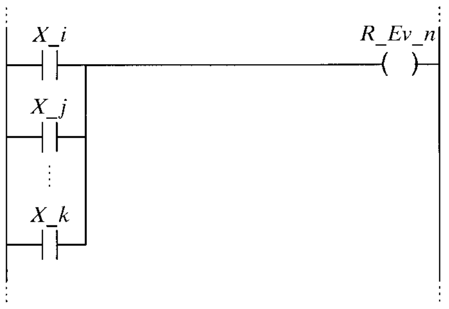

Fig. 8 Output section of the ladder diagram

Finally, the output section is shown in Figure 8. The outputs of the function block are set or reset according to the internal states of the block. The set of states that participate in generation of a particular event is determined by the control table. The outputs are interpreted as requests for actions. The actual change in the state of the actuator depends on the agreement of eventual other function blocks that deal with the same actuator. This concept enables a true modular approach, with possibility of interlocking and coordination above the device level. Since state transitions inside blocks depend on actual state of the actuators, the requested action that is blocked by external agent does not influence the correct state transition sequence in the block. The possibility of deadlock, which is inherent in such an architecture, must be avoided in the supervisory control design stage.

4 EXAMPLE

The application of the above described concept is illustrated by a simple example. Figure 9 shows a part of a laboratory model of a modular production system. The system consists of five working stations, controlled by five programmable logic controllers [10]. Two of the stations are shown in Figure 9.

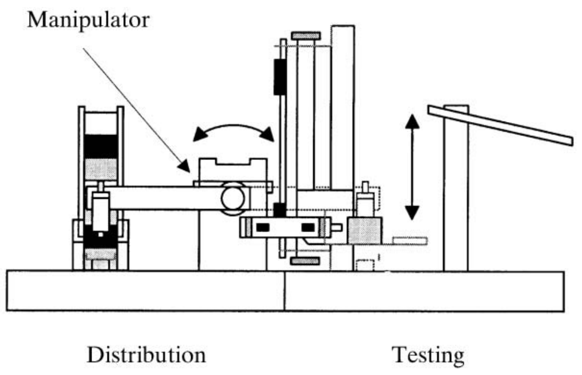

Fig. 9 Manipulator in the modular production system

We design a simple control logic that will move the arm of the manipulator, which moves the workpieces between the two stations. The arm is equipped with two limit switches to signal the left (sl) and right (sr) position. There is also a bidirectional pneumatic gear that moves the arm. The movement is initiated by opening one of the corresponding electro-pneumatic valves (ar – movement right, al – movement left). If both valves are closed or opened, the arm holds its position.

Model of the arm is shown in Figure 10. Events are labelled by the label of the related sensor/actuator followed by the subscript indicating the transition of the corresponding signal (1 – transition from 0 to 1, 0 – transition from 1 to 0). The initial state is designated by arrow pointing to the state while marker states are designated by arrows pointing out from the states.

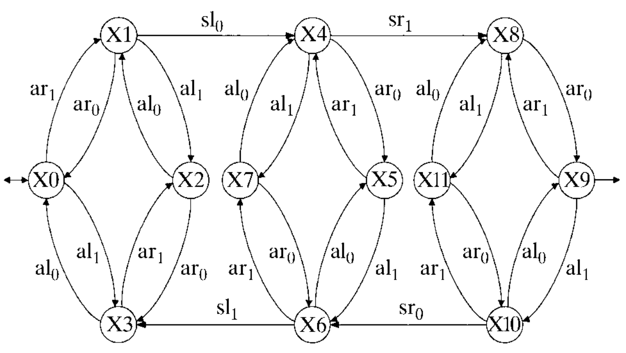

Fig. 10 Model of the manipulator arm

Any sequence of state changes of the actuators is possible, while changes of the sensor states depend on the movement of the arm caused by the particular sequence of actuator events.

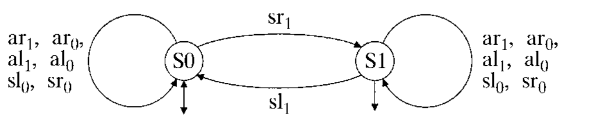

Fig. 11 Sequential specification for the manipulator arm

The sequential specification is very simple in this example; we only want to move the arm cyclically between the two end positions (Figure 11). The initial state is assumed on the left while both states are marker states indicating that any end position can be interpreted as the completed task.

An additional specification may express the requirement that both electro-pneumatic valves must not be opened at the same time. An exception to this is the initialisation phase, where this is actually required in order to fill the gear with the air. The aim of this is to provide sufficient damping of the movement.

This requirements are modelled by the interlock specification in Figure 12. Note that a change of the state of a sensor is used as an indication that the initialisation phase has terminated. After this, activation of an actuator must be followed by its deactivation, before the other one may be activated.

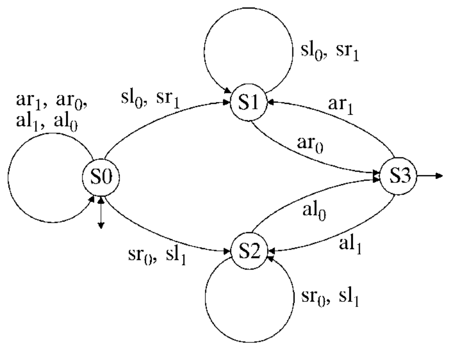

Fig. 12 Interlock specification for the manipulator arm

The process model and the specification are used as input to the synthesis of the supervisor. All the sensor events are considered as uncontrollable and the maximally permissive supervisor is searched for. The corresponding algorithm was implemented in Matlab according to [2]. The result is the automaton model of the admissible behaviour (Figure 13) that may be used for supervisor implementation.

The controller model is extracted from the automaton in Figure 13 by taking a single path through its state transition graph. The result of this procedure is shown in Figure 14.

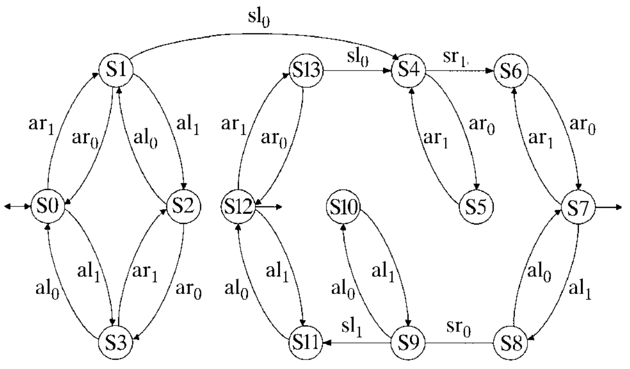

Fig. 13 Supervisor of the manipulator arm

Finally, the model of the controller is implemented in the ladder diagram as described in the previous section. The related diagram is not shown here for space limitation.

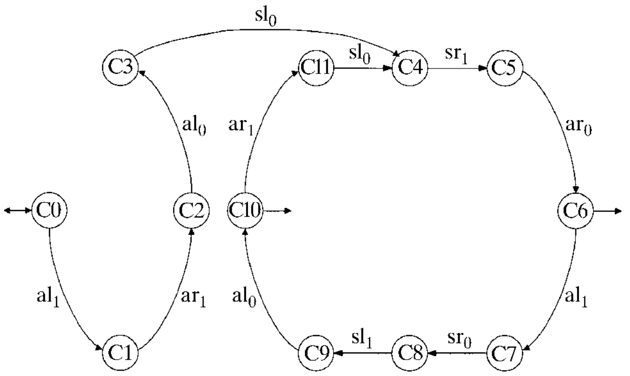

Fig. 14 Logic controller of the manipulator arm

5 CONCLUSIONS

The design procedure presented in this paper is well suited to design of the basic control logic for individual devices in manufacturing systems. Except the modelling stage, most of the design steps may be automated and this is one of the issues of the future work.

The other issues that are currently investigated are related to the interactions between devices. The main design problem here is to avoid deadlocks, which may require another supervisory layer of the control logic. The complexity of such a supervisor may be considerable, what limits the practical applicability of the proposed method. On the other hand, coordination between stations has already been studied in the Petri net framework [10] and the possibilities of integration of the two approaches are also investigated.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the financial support of the Ministry of Education, Science and Sport of the Republic of Slovenia under grant J2-2417-1538-01, which was used to finance part of this work.

REFERENCES

[1] P. J. Ramadge, W. M. Wonham, **Supervisory Control of a Class of Discrete Event Processes**. SIAM J. Control and Optimization, vol. 25, no. 1, pp. 206–230, 1987.

[2] W. M. Wonham, P. J. Ramadge, **On the Supremal Controllable Sublanguage of a Given Language**. SIAM J. Control and Optimization, vol. 25, no. 3, pp. 637–659, 1987.

[3] P. J. Ramadge, W. M. Wonham, **Modular Feedback Logic for Discrete Event Systems**. SIAM J. Control and Optimization, vol. 25, no. 5, pp. 1202–1218, 1987.

[4] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin, **Supervisory control of a rapid thermal multiprocessor**. IEEE Transactions on Automatic Control, vol. 38, no. 7, pp. 1040–1059, 1993.

[5] V. Chandra, S. R. Mohanty, R. Kumar, **Automated Control Synthesis for an Assembly Line Using Discrete Event System Control Theory**. Proc. of the American Control Conf., pp. 4956–4961, Arlington VA, June, 2001.

[6] ..., IEC, International Electrotechnical Commission, **Programmable Controllers – Part 3: Programming Languages**, Publication 61131.3, 1993.

[7] G. Frey, L. Litz, **Formal methods in PLC programming**. Proc. of the IEEE SMC 2000, Nashville, TN, 2000.

[8] C. G. Cassandras, S. Lafortune, **Introduction to Discrete Event Systems**. Kluwer Academic Publishers, 1999.

[9] W. M. Wonham, **Notes on Control of Discrete Event Systems**. University of Toronto, 1999.

[10] G. Mušič, D. Matko, **Petri Net Based Control of a Modular Production System**. Proc. IEEE International Symp. on Industrial Electronics, vol. 3, pp. 1383–1388, Bled, Slovenia, 1999.

[11] G. Mušič, D. Matko, **Petri Net Based Supervisory Control of Flexible Batch Plants**. Prepr. 8th IFAC Symp. on Large Scale Systems: Theory & Application, vol. II, pp. 989–994, Rio Patras, Greece, 1998.

Primjena teorije upravljanja diskretnim događajima u programiranju PLC-ova. U radu su prikazane realizacije regulatora zasnovanog na teoriji upravljanja diskretnim događajima. Regulatori su realizirani pomoću standardnih programirljivih kontrolera i u skladu s IEC 61131-3 programskom podrškom.

Ključne riječi: sustavi diskretnih događaja, programirljivi kontroleri, nadzorno upravljanje

AUTHORS' ADDRESS:

Asst. Prof. Dr. Gašper Mušič
Prof. Dr. Drago Matko
Faculty of Electrical Engineering, University of Ljubljana,
Tržaška 25, 1000 Ljubljana, Slovenia.
E-mail: gasper.music@fe.uni-lj.si, drago.matko@fe.uni-lj.si.

Received: 2002–10–05