15th Triennial World Congress, Barcelona, Spain

MODEL BASED PROGRAMMABLE CONTROL LOGIC DESIGN

Gašper Mušič [*,1] **Drago Matko** [*] **Borut Zupančič** [*]

[*] *Faculty of Electrical Engineering, University of Ljubljana, Slovenia*

Abstract: The paper presents a methodology of designing control logic that is implemented by industrial programmable logic controllers. The approach is based on discrete-event model of a plant to be controlled and a set of interlock and sequential specification models. Supervisory control theory is used to test the controllability of the specifications and in the final stage, to derive a model of the admissible behaviour of the system which serves as a specification for the sequential part of the controller. A laboratory scale modular assembly line case study is presented to illustrate the practical issues of the approach. *Copyright ©2002 IFAC*

Keywords: Programmable logic controllers, Discrete-event systems, Supervisory control, Manufacturing systems.

1. INTRODUCTION

Programmable logic controllers (PLC) are often referred as a workhorse of industrial automation. They form one of the most commonly used implementation platforms and the majority of industrial automation solutions are currently based on PLCs.

In the search for the more effective methods of programming industrial logic controllers most of the recent developments are focused on standardization of programming languages (IEC, 1993). 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 correctness of the program code by formal techniques, while the code is programmed in the usual way.

What is investigated here is a 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, specification of control requirements to control synthesis and implementation. The supervisory control theory (Ramadge and Wonham, 1987) is applied for the control synthesis while implementation is performed in the IEC 61131-3 compliant programming environment.

The control logic design approach used here was first presented in (Chandra *et al.*, 2001). We introduce some modifications in implementation stage where several interlock supervisors and a separate sequencing controller are implemented in order to accommodate to various operating modes. The main specific of our implementation is that a standard PLC is used instead of a personal computer and a dedicated software.

The paper is organized as follows. In Section 2 the discrete event modelling and related theory is briefly reviewed. The control design procedure is described in Section 3. Different control layers are explained and a simple application example is given. Some implementation issues are discussed at the end of the paper.

2. MODELLING

Processes to be controlled are modelled as generators of formal languages (Kumar and Garg, 1995; Cassandras and Lafortune, 1999). Such a generator may be represented in a form of a finite automaton with a partial transition function, i.e., a transition structure where, in general, only a subset of a total set of events can occur at each state.

[1] Partially supported by the Ministry of Education, Science and Sport of the Republic of Slovenia, grant J2-2417-1538-01.

2.1 *Generators and formal languages*

A deterministic generator is defined as a five-tuple

$$G = (X, \Sigma, \delta, x_0, X_m) \qquad (1)$$

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

A generator G may be represented as a directed graph with a set of nodes X. An arc $x \to x'$, labeled σ exists whenever $x' = \delta(x,\sigma)$. The generator G is interpreted as a device, which enters state x_0 when switched on and changes its state following the graph. A symbol is generated at every transition. The transitions occur spontaneously and asynchronously. The model does not include any event selecting mechanism nor time.

A finite set of generator symbols, e.g. $s = \sigma_1\sigma_2\sigma_3\sigma_4$ is called a string. An empty string is denoted as ε. The concatenation of strings: $s = s_1 s_2$ is the string of symbols of s_1 followed by the string of symbols of s_2 ($\varepsilon s = s \varepsilon = s$). The set of all finite strings of elements of Σ including the empty string ε is denoted by Σ^*.

The language is defined as a subset of Σ^*. If $s't = s$, $s, s', t \in \Sigma^*$, 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 $L = \overline{L}$.

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.

The language generated by the generator G is $L(G) = \{s \in \Sigma^*;\ \delta(x_0,s) \text{ is defined}\}$. The language marked by G is $L_m(G) = \{s \in L;\ \delta(x_0,s) \in X_m\}$. If G is a generator, the language $L(G)$ is prefix closed: $L(G) = \overline{L(G)}$; which is not always true for $L_m(G)$. $L(G)$ is interpreted as a set of all finite event sequences that may occur in the automaton. $L_m(G)$ are the sequences that end in marker states.

2.2 *Composition of generators*

Complex models may be built by composing simpler generators. Two basic composition operations exist: product, denoted by $\times$, and parallel composition, denoted by $\|$ (Cassandras and Lafortune, 1999).

Denote a pair of generators 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 generators 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 generators are only synchronised on common events, while other events may occur whenever possible. The composed generator is

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

where Ac denotes the accessible part of the generator. When $\Sigma_1 = \Sigma_2$, all events must occur synchronously therefore $\Sigma_1 = \Sigma_2 \Rightarrow G_1 \| G_2 = G_1 \times G_2$.

2.3 *Modelling for logic control*

When modelling a process for the purpose of controlling it by a logic controller the observable events are related to controller input and output signals. Two events may be assigned to every binary input/output signal, denoting changes of the signal from 'off' to 'on' (0 to 1) and vice versa.

A model of the process should capture all possible signal changes related to it. In general, there is no physical limitation on changes of the signals driving the actuators of the process. Although there is a common request that certain actuating signals are not allowed in a particular state of the process this may be treated as a part of a control specification and not a property of the process. On the other hand, possible changes of the sensor signals depend on the process state, i.e., on the past sequence of the input/output signal changes. 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.

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 are called process devices. Every device is modelled independently as a generator. 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 to the control design is preferred.

2.4 *Modelling example*

The modelling approach is illustrated by modelling a part of a modular assembly line. The system consists of five working stations, controlled by five programmable logic controllers (Mušič and Matko, 1999). Two of the stations are shown in Fig. 1.

We focus on the arm of the manipulator, which moves the workpieces between the two stations. The arm is equipped with a gripping device, consisting of a vaccum holder, electro-magnetic valve and a low pressure sensor. The valve is operated by two binary signals, one for opening and one for closing the valve. In case both signals are 'on', the state of the valve corresponds to the signal that has been switched on first. The sensor detects when a grip is firm enough to carry a workpiece. In case the workpiece falls during the transport, the sensor detects the fall.

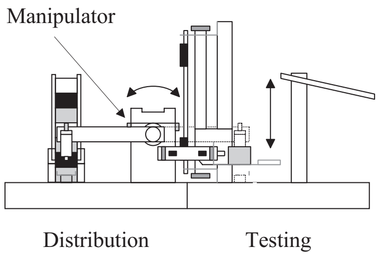

Fig. 1. Part of the modular production system

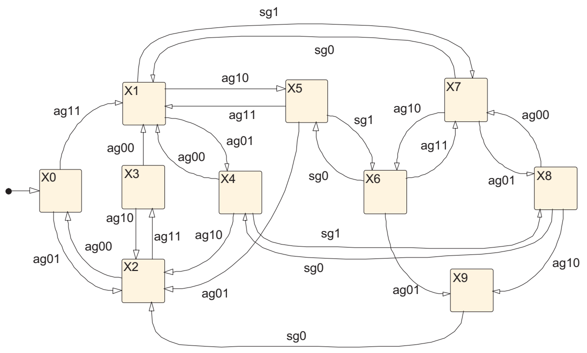

Fig. 2. Model of the gripping device

The gripping device is modelled as a generator shown in Fig. 2. Events are labelled by the label of the related sensor/actuator (sg - grip sensor, ag1 - vacuum on, ag0 - vacuum off) followed by the number indicating the transition direction 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 no marker states are designated. At this point we assume all states are marked. The language generated and marked by the device is prefix-closed.

3. CONTROL LOGIC DESIGN

A design procedure leading to control logic for a single device has been presented in (Mušič and Matko, 2001). It is based on the approach reported in (Chandra *et al.*, 2001) where the logic controller is designed using the supervisory control theory (Ramadge and Wonham, 1987). The theory enables to calculate the model of the admissible behaviour of a process and a control mechanism that will guarantee such a behaviour. In the presented approach a subset of the admissible behaviour is extracted to derive a model of a logic controller. Here the idea is further developed to tackle the problem of interacting devices. Modular approach (Wonham and Ramadge, 1988) is used to reduce computational complexity.

3.1 *Supervisory control*

The supervisory control concept deals with a discrete event system whose behaviour is restricted by an external controller called supervisor. The supervisor (Fig. 3) does not uniquely determine the next event to occur in the 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 Fig. 3). In this

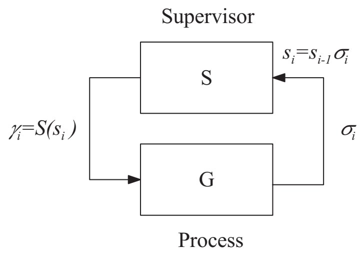

Fig. 3. Supervisory control

way the supervisor actually intervenes only in cases when some undesired process behaviour is about to take place. The supervisor is computed based on the 'open-loop' system model.

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. The set of events in the system is divided into a subset of controllable and a subset of uncontrollable events: $\Sigma = \Sigma_c \cup \Sigma_u,\ \Sigma_c \cap \Sigma_u = \emptyset$. 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.

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 (Ramadge and Wonham, 1987) and the concept of supremal controllable sublanguage (Wonham and Ramadge, 1987).

3.2 *Control design*

The proposed control structure is presented in Fig. 4. One of the key points of the approach is that the specifications are split up in two parts. The first part involves prevention of undesired behaviour. It is composed of the so-called interlocks that implement measures to assure safety, co-ordinate subprocesses, etc.

The second part deals with the sequential specification and defines prescribed order of tasks. It is related to desired system operation. The sequencing part of the control logic is only synthesized after the interlock part has been designed. This increases the flexibility of the proposed solution since only the upper layer has to be redesigned when changes in the system operation are required.

3.2.1. *Base interlock layer* In the first layer, every device is supervised by a local supervisor, which imposes device's own interlock specifications. These include various rules about switching the actuator signals that are independent of other devices. The main reason to introduce this supervisory layer is a higher

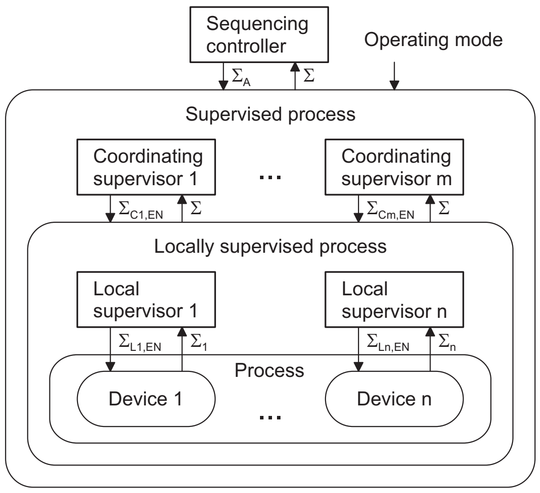

Fig. 4. Proposed control structure

modularity of resulting control logic. Local supervisors, when implemented, may be re-used, whenever a device of the same type appears in the system. When the system is in operation the local supervisors are always active, irrespective of the operating mode.

3.2.2. *Interlock layer* Next layer is composed of the co-ordinating supervisors that control the interactions among devices. In order to prevent deadlocks the non-blocking property of the composed system must be maintained. In supervisory control theory the non-blocking property of the supervisor is expressed by the requirement that the nonempty specification language $K \subseteq L_m(G)$ is relative-closed ($L_m(G)$ closed):

$$\overline{K} \cap L_m(G) = K \tag{2}$$

As in (Chandra *et al.*, 2001) we assume the set of marked states is determined by the sequential specification only. All the interlock specifications K_{spec} are prefix-closed ($K_{spec} = \overline{K}_{spec}$). Specification language K is obtained by intersection $K = K_{spec} \cap L_m(G)$ (product of related generators) and since $L_m(G) = \overline{L_m(G)} = L(G)$ we have $K = \overline{K}$ and relative-closure property trivially holds for a single specification.

It remains to check the controllability (Ramadge and Wonham, 1987) of the specification, i.e., the specification language K must satisfy the following condition:

$$\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K} \tag{3}$$

Next, the effect of the composition of the designed supervisors has to be examined. According to (Wonham and Ramadge, 1988) a conjunct of two controllable and relative-closed specifications K_1 and K_2 is controllable and relative-closed whenever the two specifications are non-conflicting:

$$\overline{K}_1 \cap \overline{K}_2 = \overline{K_1 \cap K_2} \tag{4}$$

Again, when both specifications are prefix-closed, the non-conflicting property always holds. This allows to implement the interlock supervisors modularly and to conjunct their outputs, i.e., an event σ_i is enabled, if it is enabled in all supervisors. In case a supervisor deals with a subset Σ_i of a total set of events Σ it is assumed that it always enables all events in the set $\Sigma - \Sigma_i$.

During the system operation the co-ordinating supervisors are normally active both in manual and automatic mode to ensure safe operation. In some specific operating modes, e.g., maintenance mode the co-ordinating supervisors may be disabled.

3.2.3. *Sequencing layer* The sequencing controller plays a different role than the interlock supervisors. Instead of permitting or disabling the occurrence of events in the system it has to actively trigger events that result in the state change of the actuating elements of the process. The controller actively drives the process through a desired event sequence. We implement the controller as a subset of the admissible behaviour satisfying the global sequential specification of the system. This brings us back to the original setup of Ramadge and Wonham (Fig. 3) but with the different interpretation of the controller S - a logic controller.

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.

As in previous cases the specification has to be controllable and relative-closed. The admissible behaviour model is obtained by product of the plant model and the specification. In case the specification is not controllable a supremal controllable sublanguage is calculated. The 'closed-loop' model is used as the plant model for checking the controllability and relative-closure of the sequential specification. The model incorporates the actual plant and all the interlock supervisors. The computing effort to perform controllability check and the calculation of supremal controllable sublanguage depends on the state space size of the model. With the careful selection of the interlock specifications the number of states of the 'closed-loop' model may be significantly reduced comparing to 'open-loop' model of the plant.

The sequencing controller is extracted by inspecting the state transition graph of the generator of the admissible behaviour. In every node of the graph, a set of enabled events is determined. One of them is chosen as the preferred event, which may be controllable or uncontrollable. Transitions related to controllable events (except the eventually chosen preferred event) are deleted from the graph while all transitions related to uncontrollable events are retained in the graph. 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 a minimal number of steps. Finally, inaccessible states and related transitions are removed from the graph. This way a new generator is obtained - the controller.

3.3 *Example*

To illustrate the approach a simple control logic is designed that operates the manipulator, shown in Fig. 1. The arm of the manipulator is equipped with a gripping device, model of which was described in the previous section (Fig. 2). Next we model the arm itself. It consists of a bidirectional pneumatic gear that moves the arm, two electro-pneumatic valves to control the movement, and two limit switches to signal the left (sl) and right (sr) position. Each valve is operated by a binary signal (ar - movement right, al - movement left) and is opened, when the signal is 'on'. If both valves are closed or opened, the arm holds its position. Model of the arm is shown in Fig. 5. Events are labelled in the same way as in Fig. 2, all states are marked.

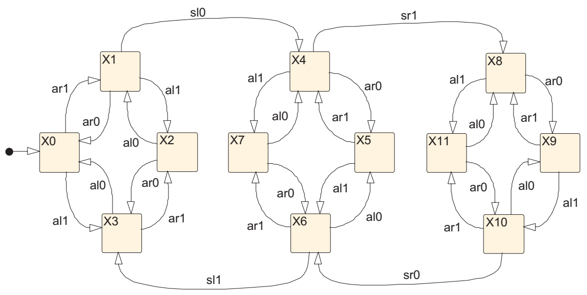

Fig. 5. Model of the manipulator arm

The basic interlock layer is designed for each device independently. Local interlock specifications for the gripper are shown in Fig. 6. The first specification defines that events ag11 and ag01 may not follow each other meaning the signals ag1 and ag0 are not allowed to be 'on' at the same time. Second specification defines that the vaccum valve on the gripper must be closed (ag01) before it may be opened again (ag11). A similar interlock specification for the arm is shown in Fig. 7. The specification defines that left (al) and right (ar) movement may not be started at the same time.

All specifications are controllable and generators shown in Figs. 6 and 7 are implemented as the local supervisors for corresponding devices. The control effect of such a generator is interpreted as follows: an event σ_i is disabled in state S_j when $\delta(S_j, \sigma_i)$ is not defined. The model of the locally supervised gripper has 7 states and 11 transitions and corresponding model of the arm has 9 states and 16 transitions.

Interlock specifications controlling the interaction between the arm and the gripper are shown in Fig. 8. First specification defines that the grip may be initiated only before the arm starts moving to the right and after it comes to the left position. Second specification defines that releasing the grip is not allowed during arm movement except when the workpiece falls. Last specification prevents start of the movement when vacuum is being switched on or off. All specifications are controllable and related generators are directly implemented in the control logic. The 'closed-loop' model of the supervised process has 43 states and 112 transitions (parallel composition of locally supervised devices - the 'open-loop' model - has 63 states and 211 transitions).

Finally, the sequential specification is shown in Fig. 9. Arrows pointing out of the states S0 and S3 indicate that these are marker states. S0 corresponds to the left position of the manipulator and S3 to the right position. Choice of the marker states guarantees that

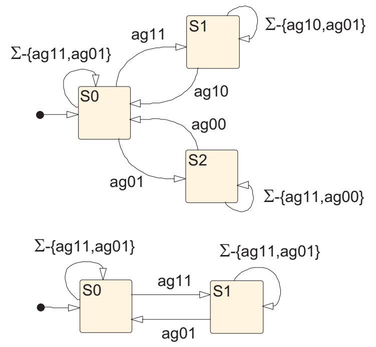

Fig. 6. Local interlock specifications for the gripper

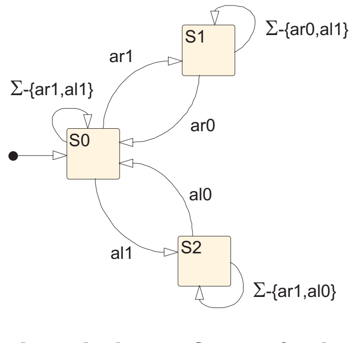

Fig. 7. Local interlock specification for the arm

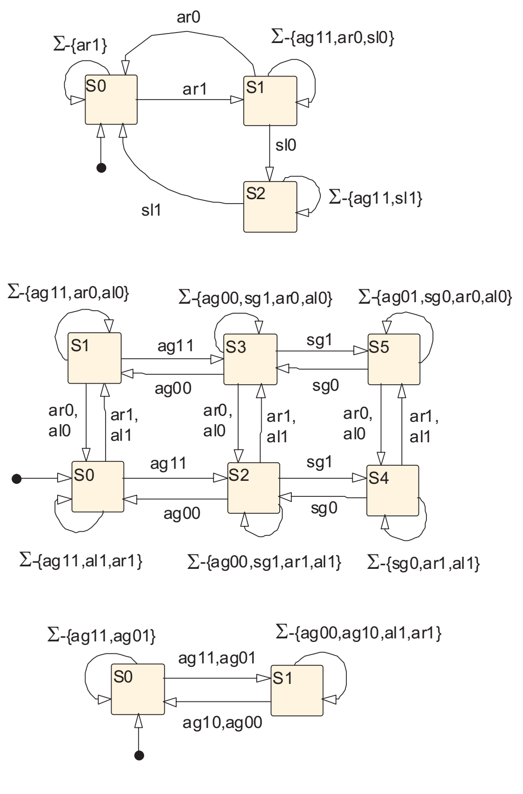

Fig. 8. Interlock specifications

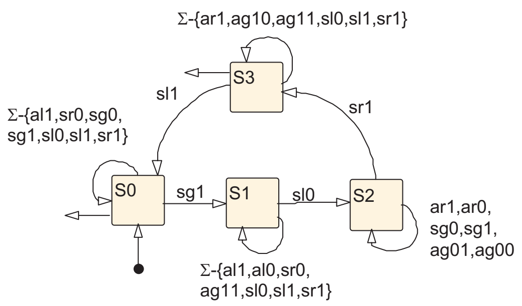
Σ-{ar1,ag10,ag11,sl0,sl1,sr1}
S3
sl1
sr1
Σ-{al1,sr0,sg0, sg1,sl0,sl1,sr1}
S0
sg1
S1
sl0
S2
ar1,ar0, sg0,sg1, ag01,ag00
Σ-{al1,al0,sr0, ag11,sl0,sl1,sr1}

Fig. 9. Sequential specification

the manipulator will actually move between the two working stations.

The specification is not controllable, therefore the supremal controllable sublanguage is calculated. Corresponding model of admissible behaviour has 37 states and 73 transitions. The sequencing controller is extracted that consists of 28 states and 44 transitions. It is not shown here for space limitations.

4. IMPLEMENTATION

The obtained generator models can be implemented by any of the standardized languages for programmable logic controllers (IEC, 1993). Ladder diagram was used to implement the control logic for the presented example. The basic idea of coding the state machine into a ladder diagram is shown in Fig. 10.

A state transition is triggered by an event flag *Ev_j* that signals the occurrence of event. Afterwards, the enable flag is reset to assure that only a single state transition is made in one PLC program scan. The problem of simultaneously appearing events is solved by maintaining an event queue. The rising and falling edges of the related I/O signals are detected and memorized. Then the event flags are set one at a time.

The readability of the program can be improved by the use of separate function blocks for the local supervisors related to a particular device. Another set of function blocks can be used for co-ordinating supervisors and a separate function block may be used to implement a sequencing controller. A general form of the later was proposed in (Mušič and Matko, 2001), a similar form is used for the supervisors. The main difference is the interpretation of the block outputs. The outputs of the supervisors are used as the enabling signals for the events while outputs of the sequencing controller are used as event triggers.

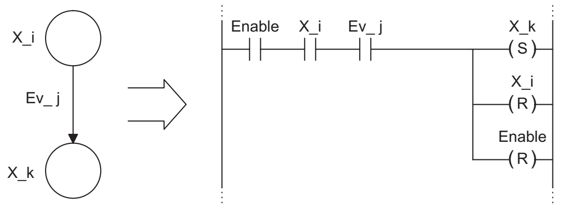

Fig. 10. Translation of a state transition into a rung of the ladder diagram

To reduce the number of connections among function blocks every supervisor block receives only signals related to events that participate in the transitions between states of the related state machine, i.e. events that only appear in selfloops on the states are not considered. Similarly, only those events are taken into account on the block outputs that are disabled at least at one of the states of the supervisor or triggered by at least one of the states of the sequencing controller.

5. CONCLUSIONS

The presented approach enables a relatively high automation of the control synthesis for the manufacturing systems. Once the model of the plant and the specification models are developed an appropriate computer tool may perform all the necessary calculations and even generate the control code. A set of functions was developed in Matlab to perform the supervisory control synthesis. Editor of the Matlab discrete-event simulation tool Stateflow was used to draw all the required models. An automatic generator of the code for the PLC is planned for the future work.

REFERENCES

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

Chandra, V., S.R. Mohanty and R. Kumar (2001). Automated control synthesis for an assembly line using discrete event system control theory. In: *Proceedings of the American Control Conference*. pp. 4956–4961. Arlington VA.

IEC (1993). *Programmable Controllers - Part 3: Programming Languages*. International Electrotechnical Commission, publication 61131.3. Geneva.

Kumar, R. and V.K. Garg (1995). *Modeling and Control of Logical Discrete Event Systems*. Kluwer Academic Publishers. Norwell MA.

Mušič, G. and D. Matko (1999). Petri net based control of a modular production system. In: *Proc. IEEE International Symp. on Industrial Electronics*. Vol. 3. pp. 1383–1388. Bled, Slovenia.

Mušič, G. and D. Matko (2001). Discrete event control theory applied to plc programming. In: *Proc. 9th Mediterranean Conference on Control and Automation*. pp. 1383–1388. Dubrovnik, Croatia.

Ramadge, P.J. and W.M. Wonham (1987). Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization* **25**, 206–230.

Wonham, W.M. and P.J. Ramadge (1987). On the supremal controllable sublanguage of a given language. *SIAM J. Control and Optimization* **25**, 637–659.

Wonham, W.M. and P.J. Ramadge (1988). Modular supervisory control of discrete event systems. *Mathematics of Control, Signals and Systems* **1**, 13–30.