PETRI NET CONTROL OF SYSTEMS UNDER DISCRETE-EVENT SUPERVISION

G. Mušič, D. Matko

Faculty of Electrical Engineering, University of Ljubljana,
Tržaška 25, SI-1000 Ljubljana, Slovenia
e-mail: gasper.music@fe.uni-lj.si, drago.matko@fe.uni-lj.si

Keywords: Petri nets, automata, supervisory control, reachability analysis, programmable logic controllers.

Abstract

Synthesis and verification of discrete control logic by a combined application of automata based supervisory control theory and Petri nets is addressed. While supervisory control theory is well suited to specification and design of the interlock part of the control logic, the sequencing part can be more easily described by Petri net models. The basic idea presented in the paper is to design basic interlock logic by the use of supervisory control theory and then to use the derived model of admissible behaviour as a process model. The sequencing part is designed in a form of a Petri net model which is verified against the process model. The basic property of interest is the absence of blocking. To study the interaction of the two models we use an extension of Place/Transition nets which includes external inputs and outputs, i.e., the Real-time Petri nets (RTPN). In order to verify the nonblocking of the controller a new kind of reachability analysis is proposed, which considers all possible changes of the controller input and output signals.

1 Introduction

Programmable logic controllers (PLCs) are the primary implementation platform used in industrial automation. While the functionality of PLCs is continuously expanding, the discrete control logic remains the core of their operation. For a long time, PLCs have been programmed in a rather intuitive way using specialised graphical programming languages such as ladder diagram. Recently, much attention has been given to formal methods and their application in design and verification of PLC programs [5].

A PLC may be treated as a kind of discrete event system, which changes its state (and outputs) in response to changes on its inputs and time. The process under control, which is connected to the controller by means of binary signals only, is viewed as the same kind of system from the controller. In the control theory there is a well established field - the supervisory control theory - covering different aspects of control of the logical discrete event systems, i.e., systems where the ordering of events is of the primary concern [1]. If we are not interesting in the time of the operation, systems controlled by PLCs fall into this category. Within the supervisory control theory the controller action is interpreted as a mechanism of enabling and disabling events in the system. The theory enables an algorithmic synthesis of a supervisor, given a process model and a specification model [1, 11]. The theory uses the automata modelling framework, where an automaton is interpreted as a generator of a formal language. The synthesis of supervisory controllers by the use of Petri nets has also been studied, e.g. [6].

Despite the sound theoretical foundation the application of the above results to PLC programming is not straightforward. The fundamental issue of investigation within the supervisory control theory is the restriction of the system's behaviour. This is well suited for designing interlocks that present a significant part of discrete control logic. For the sequential part, however, this seems less appropriate although some applications are reported (e.g. [2, 9]).

Instead of specifying allowed event sequences the behaviour of controlled system is more naturally described by a kind of flowchart. For this purpose, an international standard IEC 61131-3 [7] defines the Sequential Function Chart, a specialised graphical programming language used for structuring PLC programs. It is similar to Grafcet [4], which can be interpreted as a special kind of Petri net. This makes Petri net framework a good candidate for specifying event sequences in a more compact notation.

In the paper we study a combined approach, where the supervisory control theory is used to design the interlock part of the control logic. The sequential part is designed by Petri nets, but not in the sense of supervisor synthesis. Petri nets are used in a sense of formal specification that is verified against the model derived during the interlock synthesis. The remainder of the paper is structured as follows. Section 2 gives basic notions of automata models and supervisory control theory. The class of Petri nets used for modelling the sequential specification is described in section 3. Section 4 presents the proposed approach of verification of the Petri net specification model. A simple example is given to illustrate the approach.

2 Automata and supervisory control

Processes to be controlled may be modelled as generators of formal languages [1]. 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.

A deterministic generator 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 generator. $\delta : X \times \Sigma \rightarrow X$ 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, i.e., states with a special meaning.

The generator G is interpreted as a device, which enters state x_0 when switched on and changes its state according to its state transition function. A symbol $\sigma_i \in \Sigma$ 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. The language generated by the generator G is $\mathcal{L}(G)$. It is interpreted as a set of all finite event sequences (strings) that may occur in the automaton. The language marked by G is denoted by $\mathcal{L}_m(G)$ and consist of event sequences that end in marker states.

The supervisory control concept deals with a discrete event system whose behaviour is restricted by an external controller called supervisor. The supervisor (Fig. 1) 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. 1). 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.

The supervisor is computed based on the 'open-loop' system model and a specification model. 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 and the concept of supremal controllable sublanguage [1, 11].

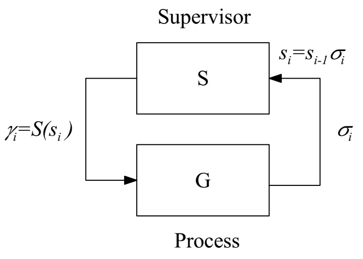

Figure 1: Supervisory control

3 Real time Petri nets

Petri nets as a tool for modelling and specification of manufacturing systems are described in a number of sources, such as [3, 8]. A Place/Transition Petri net can be described as a bipartite graph consisting of two types of nodes, places and transitions. Nodes are interconnected by directed arcs. State of the system is denoted by distribution of tokens (called marking) over the places. Formally, a Petri net is a five-tuple PN $= (P, T, I, O, m_0)$, where

- $P = \{p_1, p_2, \ldots, p_k\}, k > 0$ is a finite set of places,
- $T = \{t_1, t_2, \ldots, t_l\}, l > 0$ is a finite set of transitions (with $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$),
- $I : P \times T \rightarrow \mathbf{N}$ is an input function that specifies weights of arcs directed from places to transitions,
- $O : P \times T \rightarrow \mathbf{N}$ is an output function that specifies weights of arcs directed from transitions to places,
- $m : P \rightarrow \{0, 1, 2, \ldots\}$ is a marking, m_0 is the initial marking.

For the purpose of logical modelling required in sequential control specification and synthesis we use the class of ordinary Petri nets. This means all the non-zero arc weights are equal to one. The switching rule of an ordinary Petri net is given as follows:

i) a transition is enabled if each of the input places of this transition contains at least one token,

ii) an enabled transition may or may not fire, which depends on an additional interpretation,

iii) a firing of a transition is immediate and removes a token from each of the input places of the transition and adds a token to each of the output places of the transition.

For the purpose of simulation and possible implementation by industrial controllers, the input/output interpretation can be added to resulting models. One of such extensions is a class of Petri nets called *Real-Time Petri Nets* or RTPN [10]. Physical output functions are assigned to places and physical input and timing variables are assigned to transitions of the PN model. Formally, an RTPN is defined as an eight tuple $RTPN = (P, T, I, O, m_0, D, Y, Z)$ where

- (P, T, I, O, m_0) is as defined before;
- $D : T \rightarrow \mathbf{R}^+$ is a firing time-delay function;
- $Y : T \rightarrow \mathbf{B}$ is an input signal function, where $\mathbf{B}$ is the set of Boolean expressions on input signals;
- $Z : P \rightarrow A \times \{0, 1\}$ is a physical output function, where A is the set of output signals.

Several properties of Petri net models have been defined and investigated by different authors. For the purpose of modelling of industrial processes the most important properties are liveness, boundedness (safeness) and reversibility. Definition and meaning of these properties can be found in [8].

4 Verification of a Petri net control specification

The aim of the verification is to answer a question whether a specification model is correct. This is done by examination of various properties of the model, such as stability, absence of deadlocks, etc. In our case we limit our investigation to the study of a single property, i.e., we check if the Petri net specification is not blocking the system operation.

The non-blocking property of a Petri net is traditionally regarded as the absence of deadlocks and closely related to a concept of liveness. A Petri net is said to be *live* when it is possible to ultimately fire any transition of the net by progressing through some firing sequence, starting from any marking that is reachable from a given initial marking [8]. A live Petri net guarantees a deadlock-free operation.

When examining the Petri net specification of a logic controller, the property of liveness is not enough to ensure the non-blocking operation due to external inputs and outputs and their interrelations. We can say the liveness of a PN is a necessary but not sufficient condition for the non-blocking operation of a related controller.

To examine the possible blocking of the controller the relation between inputs and outputs must be taken into account. In other words, instead of analysing the 'open-loop' model of the controller we have to study the 'closed-loop' model of the control system. Such an approach may be considered a model-based approach to verification, according to classification in [5].

4.1 Modelling a process under control

The key to success of such a verification approach is a model of the process under control, which is not readily available in many cases. In certain cases, however, we can use models that are developed during initial stages of the control logic design.

A multistage approach to design of the control logic is schematically shown in Fig. 2 [9]. 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. The sequencing part of the control logic is only synthesized after the interlock part has been designed.

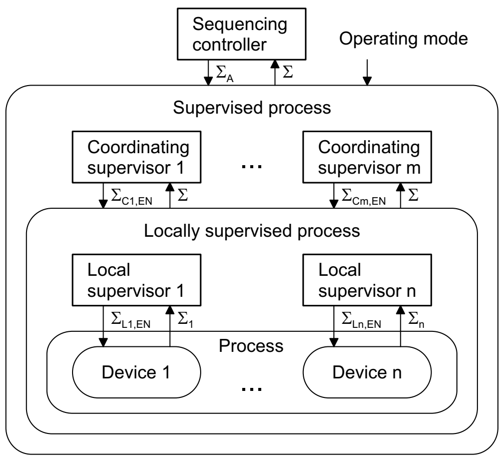

Figure 2: Proposed control structure

The set of interlock supervisors may be designed within a framework of supervisory control theory. Result of the synthesis of the interlock part is a model of admissible behaviour, i.e., the model of all possible event sequences in the controlled system that comply with the interlock specification. This model can be used as an 'open-loop' process model when designing the sequencing part of the control logic.

4.2 Petri net specification of the sequencing part of the control logic

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 a state change of the actuating elements of the process. The controller actively drives the process through a desired event sequence. The design of the sequencing part of the control logic may also be performed within the supervisory control theory, e.g. [2, 9], but the design approach is not that straightforward as with the interlock part. Here we explore an alternative way where we formalize the specification by the Petri net. The notation of RTPN is used to include controller input and output signals into a specification model.

Other modelling and specification approaches, e.g. finite automata, could be used to perform this task. As we require the Petri net models of the control logic are bounded, or even safe in most cases, the descriptive power of such a model is the same as that of a finite automaton. The main advantage of the Petri net approach is that the representation of the state of the system is distributed over places of the net. This enables a compact representation of concurrency and synchronisation, which makes Petri net models easy to understand. Another advantage of the Petri net representation is the straightforward path from the developed specification models to the industrial implementation. This is due to the closed relationship between SFC, Grafcet and Petri nets, which enables a SFC to be directly redrawn from a Petri net model and some of the classical properties of Petri nets can be applied also to SFCs [3, 4].

4.3 Verification of a RTPN

A basic analysis approach for the study of the so-called behavioural properties of Petri nets is the construction of the coverability tree or coverability graph [8]. For a bounded Petri net the coverability tree is called the reachability tree and the coverability graph is called the reachability graph, since they contain all possible reachable markings. In this case all the behavioural properties (inluding liveness) can be checked by exploring the reachability tree or reachability graph.

The main disadvantage of this approach is that it is an exhaustive method since all possible markings of the net have to be enumerated. It is therefore appropriate for relatively small-size models.

The first step in the proposed verification approach is the construction of the reachability graph of the Petri net. Only the (P,T,I,O,m_0) part of the RTPN is considered at this step. Safeness and liveness of the net are checked from the graph.

In the second step, inputs and outputs of the RTPN and the above described model of the admissible behaviour are taken into account. The reachability graph is expanded by all event sequences possible between any two transition firings. This considers both changes of the controller outputs (controllable events) due to defined output signal function of the RTPN as well as changes of the controller inputs (uncontrollable events) that result from the process dynamics. It is assumed that an attempt to perform the output function is always made before firing any enabled transition of the RTPN.

Events that are considered here are defined as changes of a state of a binary signal. Therefore, two events are related to any input or output of the controller. During the construction of the graph, an information on the state of the signals has to be kept, based on the past occurrences of related events. In the case the process model does not contain all the events related to controller inputs/outputs, the unmodelled events are not considered during the construction of the graph.

In the described way a new kind of reachability graph is derived. A set of nodes is associated with every reachable marking and the transitions between the nodes are of two types:

– transitions of a RTPN connect nodes associated with distinct markings,
– transitions related to events in a model of admissible behaviour connect nodes associated with the same marking.

Since the derived graph includes input and output events we will call it the *IO-reachability graph* of a RTPN under supervision. It must be noted that we only consider ordering of events, while timing information of a RTPN is omitted.

Finally, the IO-reachability graph is used to analyse a potential blocking of a controller. Here we apply the following definition.

Definition: A control specification given as a RTPN is said to be nonblocking under supervision, when a corresponding IO-reachability graph:

(i) contains all transitions of the RTPN, i.e. every transition appears at least once as a label of an edge in the graph and

(ii) may be interpreted as a nonblocking automaton, given $X_m = \{x_0\}$. In the interpretation of the graph as an automaton, transitions of a RTPN are considered as additional events in the system. □

For an exact definition of nonblocking see [1]. Roughly speaking, the automaton is said to be non-blocking, if it is capable to reach a marker state from any reachable state. In our case, since $X_m = \{x_0\}$, the automaton is non-blocking, when it is able to return to initial state from any reachable state. This also implies, that a corresponding RTPN can always return to the initial state. Since we also request that all transitions are contained in the IO-reachability graph, any transition of the RTPN can eventually be fired, starting at any reachable marking. We can say such a RTPN is *live under supervision*.

4.4 Example

To illustrate the proposed concept of verification we present a simple example. We consider a part of a laboratory scale modular production line. The line is composed of five working stations and every working station is further composed of a set of pneumatic pistons, gears, two state sensors, electro-pneumatic actuators, which form a mechanical setup that can be controlled by a PLC to perform a required operation.

To simplify the presentation we will only consider a small part of the distribution station, consisting of a pneumatic piston, that takes a workpiece from the input buffer and a manipulator that transports the workpiece further. The setup is shown in Fig. 3.

The pneumatic piston is equipped by two limit switches, indicating backward (sb) and forward (sf) position. The backward position is also the initial position of the piston. The switches are wired to a power supply and the controller in a standard way so that the controller receives logical 1 when the limit is reached and logical 0 otherwise. The piston has a single actuator (af), it moves in the forward direction when $af = 1$ and in the backward direction when $af = 0$. Movement of the piston is limited to the distance between the two limit switches. The automaton modelling the behaviour of the piston is shown in Fig. 4. Every input/output signal is modelled by two events. One indicates the transition of the signal form 0 to 1, e.g. $af1$, the other from 1 to 0, e.g. $af0$. The change of the state of the actuating signal is possible in any moment, while the possible change of the state of the sensing signal depends on the state of the actuator and the current position of the piston. The initial state of the automaton is designated by arrow pointing to the state while no marker states are designated. At this point we assume all states are marked.

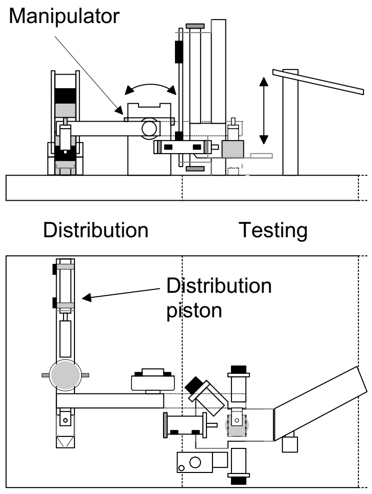

Figure 3: Part of the production line

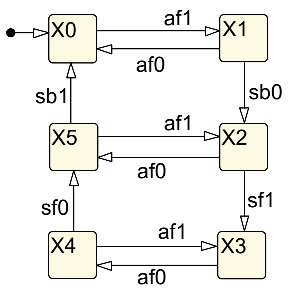

Figure 4: Model of the pneumatic piston

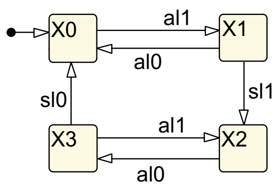

Figure 5: Model of the manipulator

The manipulator consists of a bidirectional pneumatic gear, which moves the arm with a gripping device. The full model may be found in [9]. Here we use a simplified model shown in Fig. 5. We consider only one sensor, which indicates the position of the manipulator at the input buffer (physically this is the left position, therefore *sl*) and one actuator, which forces the manipulator to move toward the input buffer (*al*). We assume the manipulator moves away form the buffer when $al = 0$.

The interlock specifications for the two devices are shown in Fig. 6. The first automaton shown in Fig. 6 models the requirement that the manipulator must be kept away from the piston, when the later is moving. The second automaton models the requirement that the piston must not start moving forward, when the manipulator is taking a workpiece.

The specifications are obviously not controllable, since some of the uncontrollable events are disabled in certain states. Supervisory control theory is applied to calculate the maximally permissive supervisor, which results, when applied to the parallel composition of the above models, in a model of admissible behaviour, shown in Fig. 7. The admissible behaviour model of the piston and the manipulator is a behaviour, which does not violate the constraints imposed by specification in Fig. 6 and at the same time retains as many of the event sequences in the system as possible. In the presented case we actually imposed mutually exclusive operation of the two devices.

Now, consider two sequential specifications, given by two RTPN models shown in Fig. 8. The input/output signal functions for the first specification are as follows: $Y(t_1) = 1, Y(t_2) = sl$, $Y(t_3) = sf$, $Y(t_4) = sb$ AND NOT sl; $Z(p_2) = (al, 1)$, $Z(p_3) = (af, 1)$, $Z(p_4) = (al, 0)$, $Z(p_5) = (af, 0)$. The input signal functions for the second specification are $Y(t_1) = 1$, $Y(t_2) = sf$ AND sl, $Y(t_3) = sb$ AND NOT sl; while output signal functions are the same. No firing delays are defined: $D(t_i) = 0, \forall t_i$. Reachability graphs for the two Petri nets are

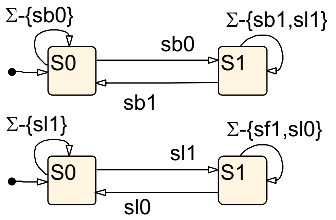

Figure 6: Interlock specifications

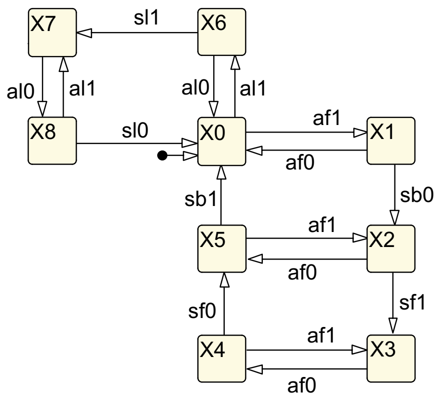

Figure 7: Admissible behaviour

show in Figs. 9 and 10. By the inspection of the two graphs we can prove that both Petri nets are safe and live.

IO-reachability graph for the first sequential specification is shown in Fig. 11. The controller is not blocking. This may be surprising because the interlock supervisor enforces the mutually exclusive operation of the piston and manipulator while the Petri net specifies a parallel operation. The controller is not blocking because the Petri net is drawn in such a way that the two parallel activities only synchronize after both devices are back in the initial state.

If we require the synchronisation of activities in between, the controller blocks, which is clearly seen from the IO-reachability graph for the second sequential specification, shown in Fig. 12. In this case the liveness of the related Petri net is not sufficient for the non-blocking operation of the controller. The blocking is only visible after the construction of the IO-reachability graph.

5 Conclusions

The presented approach enables a detailed analysis of the potential blocking in the control logic that is built on the basis of Petri net specifications. The main advantage of the approach is that the relations among input and output signals of the controller are taken into account, which is not possible by classical methods of the Petri net analysis. The drawback of the approach is the complexity of the resulting IO-reachability graphs, which limits the approach to analysis problems of a small size.

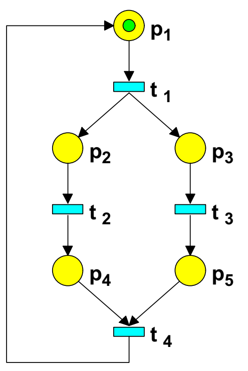

Figure 8: Two sequential specifications

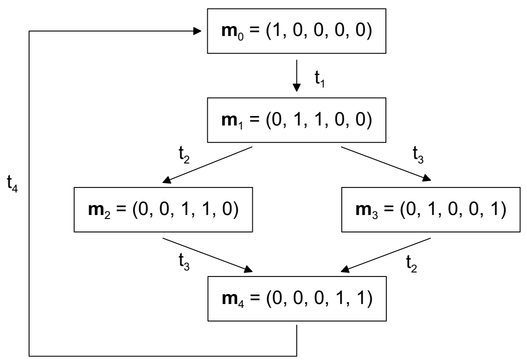

Figure 9: Reachability graph for the first RTPN

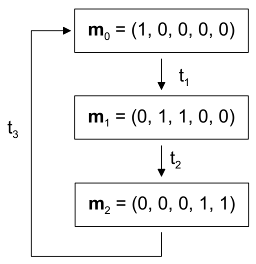

Figure 10: Reachability graph for the second RTPN

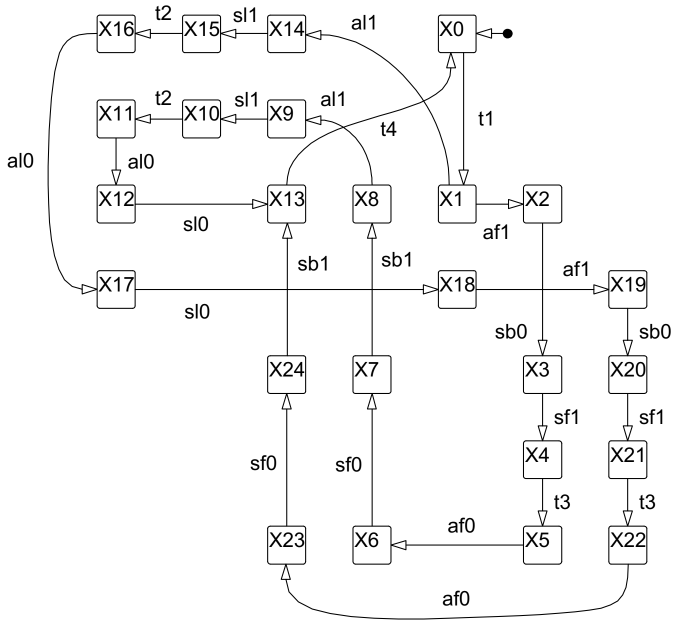

Figure 11: IO-reachability graph for the first sequential specification (non-blocking)

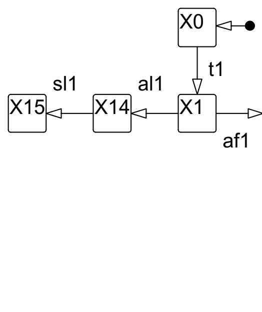

Figure 12: IO-reachability graph for the second sequential specification (blocking)

References

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

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

[3] R. David and H. Alla. ”Petri Nets for Modeling of Dynamic Systems - A Survey”, *Automatica*, **30** (2), pp. 175–202, (1994).

[4] R. David. ”Grafcet: A Powerful Tool for Specification of Logic Controllers”, *IEEE Trans. on Control Systems Technology*, **3** (3), pp. 253–268, (1995).

[5] G. Frey and L. Litz. ”Formal Methods in PLC-Programming”, *Proc. of the SMC’2000*, (2000).

[6] L. E. Holloway, B. H. Krogh, A. Giua. ”A Survey of Petri Net Methods for Controlled Discrete Event Systems”, *Discrete Event Dynamics Systems: Theory and Applications*, **7**, pp. 151–190, (1997).

[7] IEC. ”International Electrotechnical Commission: Programmable Controllers - Part 3: Programming Languages”, publication 61131.3, (1992).

[8] T. Murata. ”Petri Nets: Properties, Analysis and Applications”, *Proc. IEEE*, **77** (4), pp. 541–580, (1989).

[9] G. Mušič, D. Matko, B. Zupančič. ”Model based programmable control logic design”, *Preprints of the 15th Triennial IFAC World Congress*, (2002).

[10] M. Zhou, E. Twiss. ”Design of Industrial Automated Systems Via Relay Ladder Logic Programming and Petri Nets”, *IEEE Trans. on Systems, Man, and Cybernetics - Part C*, **28** (1), pp. 137–150, (1998).

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