COMBINED SYNTHESIS/VERIFICATION APPROACH TO PROGRAMMABLE LOGIC CONTROL OF A PRODUCTION LINE

Gašper Mušič * **Drago Matko** *

* *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. A two stage approach is proposed. In the first stage a set of interlock supervisors is designed based on discrete-event model of the plant and a set of interlock specification models. Supervisory control theory is used to test the controllability of the specifications and to derive a finite automaton representation of the admissible behaviour of the system. In the second stage the model of admissible behaviour is adopted as a plant model and used for the verification of the sequential specification model in a form of a Petri net. The basic property of interest is the absence of blocking. To study the interaction of the two models an extension of Place/Transition nets is used, which includes external inputs and outputs, i.e., the Real-time Petri nets (RTPN). A new kind of reachability analysis is applied, which considers all possible changes of the controller input and output signals. This enables to verify the nonblocking operation of the sequential controller.

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

1. INTRODUCTION

One of the most commonly used implementation platforms in industrial automation consists of programmable logic controllers (PLCs) and related programming software. Recently, much attention has been given to formal methods and their application in design and verification of PLC programs. This is motivated by the growing complexity of the control problems, demands for reduced development time and need for reuse of existing software modules on one hand, and on the other hand by the need for verification and validation procedures related to application of PLCs in safety-critical processes (Frey and Litz, 2000).

The related approaches are either mainly dealing with (automatic) synthesis of the PLC programs, or with formalisation of the specifications and verification of the program against the formal specification.

In the paper we study a combined approach, where the supervisory control theory (Ramadge and Wonham, 1987) is used to synthesize the interlock part of the control logic. The sequential part is then designed by Petri nets (Murata, 1989), which are used in a sense of formal specification that is verified against the model derived during the interlock synthesis. The basic property of interest is the absence of blocking. The motivation for the use of two modelling formalisms is twofold: First, the supervisory control theory is well suited for the interlock design. Interlocks may be conveniently modelled modularly by simple state machines, while complex sequential and partly concurrent behaviour is much more difficult to model this way. Secondly, the Petri net framework provides an intuitive way of modelling operation sequences, while the Petri net based supervisory control methods are less elaborated, especially in terms of event feedback, and little synthesis tools are available. The combined approach exploits the advantages of both frameworks.

2. COMBINED SYNTHESIS/VERIFICATION APPROACH

To be able to present the proposed approach in a formal way, some basic notions of the two paradigms used, i.e. supervisory control theory and Petri nets, are recalled first.

2.1 Automata and supervisory control

A deterministic generator is defined as a five-tuple $G = (X, \Sigma, \delta, x_0, X_m)$ where X is a set of states, Σ is a set of symbols associated with events, $\delta : X \times \Sigma \rightarrow X$ is a state transition function and is in general a partial function on its domain, x_0 is the initial state and X_m is a set of marker states. A symbol $\sigma_i \in \Sigma$ is generated at every transition. A finite set of symbols is called a string. The language generated by G ($\mathcal{L}(G)$) 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 consists of event sequences that end in marker states. Let Σ^* denote a set of all finite strings of elements of Σ including the empty string, and let st denote a concatenation of strings $s, t \in \Sigma^*$. A prefix closure of a language $L \subseteq \Sigma^*$ is then defined as $\overline{L} = \{s \in \Sigma^*; \exists t \in \Sigma^*, st \in L\}$. The automaton is non-blocking, if it is capable to reach a marker state from any reachable state ($\overline{\mathcal{L}_m(G)} = \mathcal{L}(G)$).

The supervisory control concept (Ramadge and Wonham, 1987) deals with restrictions on the behaviour of a discrete event system imposed by an external controller – a supervisor, acting by disabling events. The set of events is partitioned into two disjoint subsets – controllable and uncontrollable events: $\Sigma = \Sigma_c \cup \Sigma_u$, $\Sigma_c \cap \Sigma_u = \emptyset$. The uncontrollable events can not be disabled. The supervisor is computed based on the open-loop system model and a specification model. The key issues are the concept of controllability and the concept of supremal controllable sublanguage (Cassandras and Lafortune, 1999; Wonham, 2003).

2.2 Petri nets and Real-time Petri nets

A Place/Transition Petri net (Murata, 1989; Cassandras and Lafortune, 1999) 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. For the purpose of simulation and possible implementation by industrial controllers, the input/output interpretation can be added. One of such extensions is a class of Petri nets called *Real-Time Petri Nets* (RTPN) (Zhou and Twiss, 1998). Formally, a RTPN is defined as an eight tuple $RTPN = (P, T, I, O, m_0, D, Y, Z)$ 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 a function that specifies weights of arcs directed from places to transitions; $O : P \times T \rightarrow \mathbf{N}$ is a 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. $D : T \rightarrow \mathcal{R}^+$ is a firing time-delay function; $Y : T \rightarrow \mathcal{B}$ is an input signal function, where $\mathcal{B}$ is the set of Boolean expressions on input signals; $Z : P \rightarrow 2^{A \times \{0,1\}}$ is a physical output function, where A is the set of output signals [1] . In the following we will only deal with safe RTPN, i.e. $m(p) \leq 1, \forall p \in P$. The output function of a place sets the related output signals to the specified values when the place is marked.

To enable a detailed analysis of blocking or non-blocking properties of a RTPN that is executed under a restriction of a discrete event supervisor we have to precisely define the firing rule of a RTPN. A firing rule defined in (Zhou and Twiss, 1998) is here adopted with a slight modification.

In a standard Petri net theory a transition $t \in T$ is said to be enabled if $m(p) \geq I(p,t), \forall p \in {}^\bullet t$. Here ${}^\bullet t \subseteq P$ denotes the set of places which are inputs to a transition $t \in T$. For a RTPN we adopt this definition but we call such a transition a *state enabled* transition. A set of state enabled transitions of a RTPN under marking m is $T_e(m) := \{t | t \text{ is state enabled under } m\}$.

Let $v : B \rightarrow \{0, 1\}$ denote an input state where $v(b)$ is the binary value corresponding to a state of the input signal $b \in B$. Similarly, $u : A \rightarrow \{0, 1\}$ denotes an output state. Next we define that a transition $t \in T$ is *input enabled* under an input state v when $eval(Y(t), v) = 1$. Function $eval(e, v)$ denotes an evaluation of the Boolean expression $e \in \mathcal{B}$ by the given input state v. A set of input enabled transitions of a RTPN under input state v is $T_i(v) := \{t | t \text{ is input enabled under } v\}$.

We also define that a transition is *output enabled* when all the preceding control actions have actually been executed. A transition $t \in T$ is output enabled under an output state u when $Z(p) = \{(a_1, i_1), \ldots, (a_n, i_n)\} \Rightarrow u(a_j) = i_j, \forall (a_j, i_j) \in Z(p), \forall p \in {}^\bullet t$. A set of output enabled transitions of a RTPN under output state u is $T_o(u) := \{t | t \text{ is output enabled under } u\}$.

The *firing rule* of a RTPN can now be defined as follows: (*i*) a transition $t \in T$ is enabled if it is state enabled, input enabled and output enabled, i.e., $t \in T_e \cap T_i \cap T_o$; (*ii*) an enabled transition may or may not fire, which depends on the firing time-delay function associated with it: a transition with zero time delay fires immediately when enabled, a transition with non-zero time delay fires immediately after delay $D(t)$ expires (the corresponding timer starts when transition is enabled); (*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. We assume that only a single transition of a PN fires at a time.

[1] This definition of the output function is slightly changed with respect to (Zhou and Twiss, 1998) and (Mušič and Matko, 2003).

Given a marking m, a marking m' is said to be immediately reachable ($m' \in R_1(m)$) if there exists a transition t such that t is state enabled under m and its firing results in m'. A marking m_k is said to be reachable from a marking m_0 ($m_k \in R_\infty(m_0)$) if there exists a sequence $< m_0 m_1 \ldots m_k >$ such that $m_i \in R_1(m_{i-1})$ for $0 < i \leq k$. The notion od reachability can be extended by considering input and output signals of a RTPN. Given a marking m, input state v, and output state u, m' is said to be *immediately reachable under I/O state v, u* if there exists a transition t such that t is state enabled, input enabled, and output enabled under m, v, and u, respectively, and its firing results in the marking m'.

2.3 Events and I/O signals

The feasible set of input/output (I/O) signal patterns is defined by the supervisor S and is implicitly given by the discrete event model of the supervised plant, $H_a = (X, \Sigma, \delta, x_0, X_m)$, which is derived by the supervisory control synthesis procedure. As explained in (Mušič and Matko, 2003) we do not consider blocking at this point therefore $X_m = X$. Language $\mathcal{L}(H_a) = L_a$ generated by H_a contains all admissible event sequences.

An event $\sigma \in \Sigma$ may be regarded either as an external event observed through the change in the state of the corresponding I/O signal or may be actively triggered by the controller. In any case, a change of the input or output state is associated by every event $\sigma \in \Sigma$. This will be denoted by $v' = \delta_v(v, \sigma)$ and $u' = \delta_u(u, \sigma)$. The sets of output and input states are denoted as $U := \{u | A \rightarrow \{0,1\}\}$ and $V := \{v | B \rightarrow \{0,1\}\}$. Next we define a set of total states $W := \{w | w = (x, u, v)\}$.

Considering event sequences that are generated by the model of the admissible behaviour H_a we construct a new total state automaton $H_w = (W, \Sigma, \xi, w_0, W_m)$, where $W \subseteq X \times U \times V$ as defined above, Σ is the set of events composing the admissible behaviour, and ξ is the new state transition function defined as follows:

$$\xi(w, \sigma) = \begin{cases} (\delta(x, \sigma), u', v') & \text{if } \delta(x, \sigma) \text{ defined} \\ \text{undefined} & \text{if } \delta(x, \sigma) \text{ undefined} \end{cases} \quad (1)$$

For convenience, ξ is extended from domain $W \times \Sigma$ to $W \times \Sigma^*$ in the usual way. Initial state w_0 is (x_0, u_0, v_0) and all states are marked, $W_m = W$. We note that $\mathcal{L}(H_w) = \mathcal{L}(H_a) = L_a$, which is evident from (1).

We assume the initial state w_0 of H_w when a corresponding RTPN is marked by the initial marking m_0. The changes of the input/output signal state are driven by the evolution of the two models, the total state automaton model of admissible behaviour and the RTPN model of operational sequences.

Definition 1. A RTPN is deadlock-free under discrete event supervision when for every reachable marking $m \in R_\infty(m_0, w_0)$ there exists a marking m' that is immediately reachable from m, i.e., $m' \in R_1(m, w)$ where $w = \xi(w_0, s); s \in L_a$.

Here $R_1(m, w)$ denotes the set of immediately reachable states of the RTPN under marking m and I/O state v, u, where $w = (x, u, v)$. Similarly $R_\infty(m_0, w_0)$ denotes the total reachable set of the RTPN.

2.4 IO-reachability graph

To be able to analyse the existence or absence of deadlock in the RTPN under discrete event supervision we propose a new kind of reachability graph that enumerates all possible event and transition sequences.

Nodes of the graph are pairs (m, w), where m is a marking of the RTPN while w is the state of the automaton H_w. We start the construction in the initial state (m_0, w_0), where $w_0 = (x_0, u_0, v_0)$. We then search for a set of feasible events. This is a subset of feasible events $\Gamma(x_0)$ of the automaton H_a. More precisely, the set is composed of two subsets. One is the set of all events feasible at x_0 and not generated by the RTPN. Second is the set of events generated by actions of the marked places of the RTPN and defined by the output function Z, which are also feasible at x_0.

Let Σ_{CTRL} denote a set of events triggered by RTPN, and Σ_{SP} a set of events that are not generated by the RTPN ($\Sigma_{SP} = \Sigma - \Sigma_{CTRL}$). Let $\Sigma_A(m)$ denote the set of events generated by actions of the marked places of the RTPN. The set of feasible events Σ_F at H_a in the state x and RTPN marked by m is then given by

$$\Sigma_F(x, m) = \Gamma(x) \cap (\Sigma_{SP} \cup \Sigma_A(m)) \quad (2)$$

Then a node (m_0, w_i) where $w_i = \xi(w_0, \sigma_i)$ is added for $\forall \sigma_i \in \Sigma_F(x_0, m_0)$ and the arc from (m_0, w_0) to (m_0, w_i) is labelled σ_i.

Next the set of immediately reachable markings $R_1(m_0, w_0)$ is determined. Then for every corresponding marking $m_i \in R_1(m_0, w_0)$ a node (m_i, w_0) is added and the arc from (m_0, w_0) to (m_i, w_0) is labelled t_i where t_i is the transition leading from m_0 to m_i. In case of conflicting transitions, all possible firing sequences are enumerated as in standard reachability analysis.

The procedure is repeated for every added node, and duplicate nodes of the graph are merged. The procedure stops when there are no new nodes or all new nodes are duplicate nodes.

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: (*i*) transitions of a RTPN connect nodes associated with distinct markings, (*ii*) 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 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.

It is important to note that since the construction is driven by sequential specification, only a small subset of possible I/O combinations is actually enumerated.

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

Definition 2. A control specification given as a RTPN is 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.

3. A CASE STUDY

To illustrate the proposed design concept we give a simple but realistic example. We consider a part of a laboratory scale modular production line composed of five working stations controlled by five PLCs.

To simplify the presentation we focus on a part of the line, i.e., 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. 1.

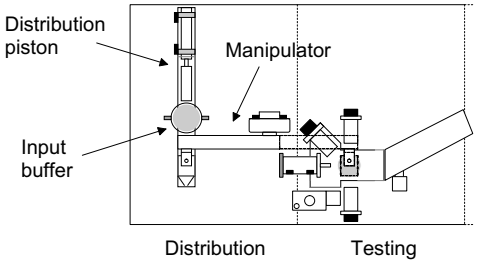

Fig. 1. Part of the production line

The station is decomposed into three devices, besides the distribution piston and arm of the manipulator there is also a gripping device mounted on the arm.

3.1 The interlock part

In the interlock synthesis stage the three devices are modelled as automata, which capture all possible input/output signal changes. In general, there is no physical limitation on changes of the signals driving the actuators of the process. When some limitations are required 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.

To illustrate the modelling concept, the model of the distribution piston is shown in Fig. 2. The piston is equipped by two limit switches, indicating backward

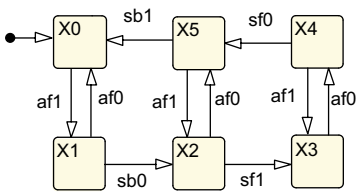

Fig. 2. Model of the pneumatic piston

(sb) and forward (sf) position. The piston has a single actuator (af), it moves forwards when $af = 1$ and backwards when $af = 0$. The movement is limited to the distance between the two limit switches. Events are labelled by the label of the related sensor/actuator followed by the suffix 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. Similar models of the arm and the gripper may be found in (Mušič *et al.*, 2002).

We define all events related to actuators (labels starting with a) as controllable and all events related to sensors (labels starting with s) as uncontrollable. For the purpose of the supervisory control synthesis, the complete model of the process may be obtained by parallel composition of device models or alternatively, the set of supervisors may be synthesized modularly.

Here the same interlock specifications as in (Mušič *et al.*, 2002) and (Mušič and Matko, 2003) are adopted. The only differences are in the simplified representation, and we assume a different initial position of the manipulator. The initial state of one automaton is adjusted correspondingly. The main requirements imposed by the interlocks are prevention of simultaneous activation of counteracting actuators and prevention of mechanical collisions of parts of the production line.

Fig. 3 shows the device local interlock specifications. Event labels are related to I/O signal labels given later in Tabs. 1 and 2. The symbol Σ denotes a set of possible events in the supervised process. The specifications define rules about switching the actuator signals that are independent of other devices. E.g., the upper automaton defines that events $ar1$ and $al1$ may not directly follow each other implying that arm left and right movements may not be active in the same time.

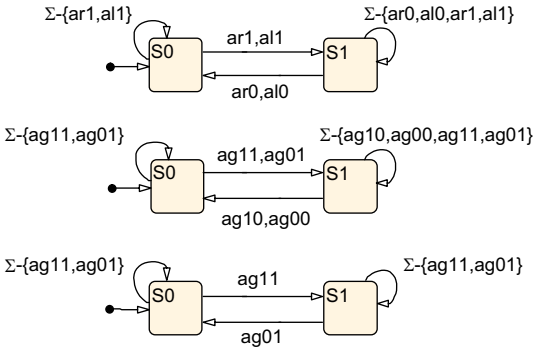

Fig. 3. Device local specifications

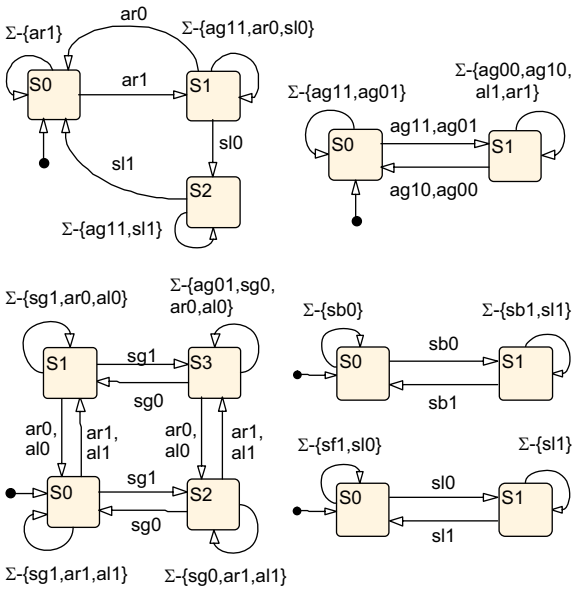

Fig. 4. Device interaction specifications

The interlock specifications controlling the interaction among devices are shown in Fig. 4. The first (top left) specification defines that the grip may be initiated only before the arm starts moving to the right or after it comes back to the left (initial) position. The second (top right) specification prevents the start of the movement when vacuum is being switched on or off. The third (bottom left) specification defines that releasing the grip is not allowed during arm movement except when the workpiece falls. Finally, the last two specifications maintain the interlock between the manipulator and the distribution piston.

The controllability check shows all but the last two specifications are controllable. For these two specifications the supervisory control theory is applied to calculate the maximally permissive supervisor, which complies to the specifications without attempting to block any of the uncontrollable events. This supervisor together with previously defined local and interlock supervisors results, when applied to the parallel composition of the device models, in a model of admissible behaviour. For the given case, it consists of 133 states and 482 transitions. The model captures all possible input/output signal changes that comply with the interlock specifications and are physically possible. It therefore represents an open-loop process model from the viewpoint of sequential part of the control logic.

Note that while a monolithic admissible behaviour model is later used for verification of the sequential logic, the interlock supervisors are implemented modularly. For the specifications that are found controllable, the related automata are directly implemented as a set of function blocks. For the two specifications that are not controllable, the related supervisor model is derived by the supremal controllable sublanguage calculation and the supervisor reduction technique (Vaz and Wonham, 1986).

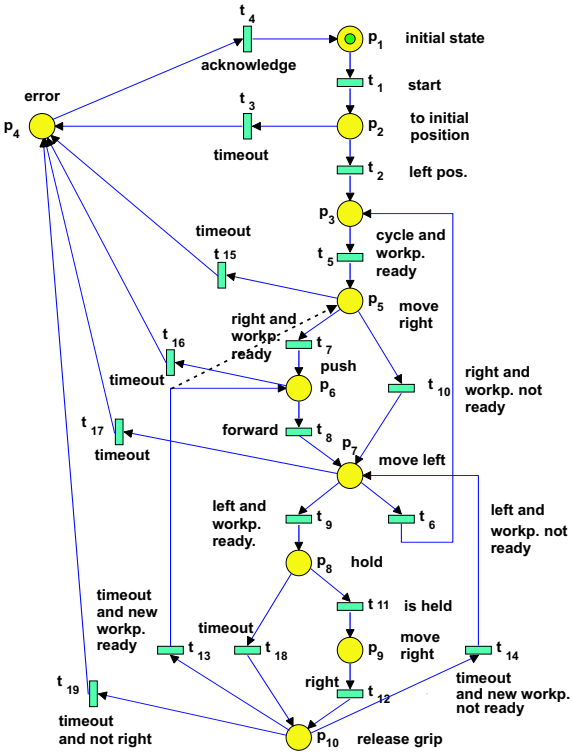

Fig. 5. RTPN sequential specification

3.2 The sequential part

The first step in designing the sequential part is to draw a Petri net that corresponds to desired operation sequence. Such a net for our case is shown in Fig. 5. After receiving a start signal and moving the arm to the initial position (left - to clear the working area of the neighbouring station), the controller checks another start signal and the presence of a workpiece to start the cyclic operation. Then the arm is moved away, the workpiece is pushed from the buffer, the arm is moved back, the gripper is activated and the workpiece is carried right to the next working station where the gripper is released. This concludes a normal working cycle. Alternative paths are provided for cases when there is no workpiece but the arm has to be moved to clear the workspace and for the cases when the requested operation does not terminate in the prescribed time. In the later case an error state is entered, which can only be left after acknowledgement of the error.

Next, input and output signals are adjoined with the transitions and places of the Petri net. This is shown in Tabs. 1 and 2. For the purpose of implementation on a specific type of PLCs, the timeouts are not explicitly assigned to places and transitions (the available type of PLCs does not support timed actions in SFC, which is chosen as a target language), i.e. $D(t_i) = 0, \forall t_i$. An additional input signal is assumed to signal the expiration of every timeout and corresponding timers are assumed to be programmed separately.

A RTPN defined this way is verified against the previously derived open-loop process model. The initial states of the input signals that are not part of the admis-

Table 1. RTPN transition conditions

$Y(t_1)$=start	legend:
$Y(t_2)$=sl	ack - error acknowledgement
$Y(t_3)$=d4s	cycle - start of the cycle
$Y(t_4)$=ack	d05s - 0.5s timeout expired
$Y(t_5)$=cycle	(similarly: d*n*s - *n*s timeout)
$Y(t_6)$=sl AND sb	sf/b - front/back pos. sensor
$Y(t_7)$=sr AND si	sg - workpiece is held
$Y(t_8)$=sf	si - workpiece present
$Y(t_9)$=sl AND sf	sl - left position sensor
$Y(t_{10})$=sr AND NOT si	sr - right pos. sensor
$Y(t_{11})$=sg	start - start of operation
$Y(t_{12})$=sr OR d6s OR NOT sg	
$Y(t_{13})$=d05s AND si AND cycle	
$Y(t_{14})$=d05s AND (NOT si OR NOT cycle)	
$Y(t_{15})$=d6s	
$Y(t_{16})$=d1s	
$Y(t_{17})$=d4s	
$Y(t_{18})$=d2s	
$Y(t_{19})$=d05s AND NOT sr	

Table 2. RTPN place actions

$Z(p_1)$={(l_start, 1),(l_error, 0)}	
$Z(p_2)$={(l_start, 0),(al, 1)}	
$Z(p_3)$={(al, 0)}	
$Z(p_4)$={(l_error, 1),(al, 0),(ar, 0),(af, 0),(ag0, 0)}	
$Z(p_5)$={(ar, 1)}	af - piston forward
$Z(p_6)$={(ar, 0),(af, 1)}	ag0 - release the grip
$Z(p_7)$={(al, 1),(ar, 0),(ag0, 0)}	ag1 - activate the gripper
$Z(p_8)$={(al, 0),(ag1, 1),(af, 0)}	al/ar - arm to the left/right
$Z(p_9)$={(ar, 1),(ag1, 0)}	l_error - error indicator
$Z(p_{10})$={(ar, 0),(ag1, 0),(ag0, 1)}	l_start - initial st. indicator

sible behaviour model may be left undefined or may be fixed at specific value. In our case signals *ack*, $d05s$, $d1s$, $d2s$, $d4s$, $d6s$ and *si* are set undefined, while *start* and *cycle* are assumed to be 1. The initial position of the devices in assumed back and left and gripper released ($sf = 0$, $sb = 1$, $sl = 1$, $sr = 0$, $sg = 0$). The initial state of all output signals is assumed to be 0.

For the given case the constructed IO-reachability graph consists of 47 nodes and 72 transitions. It shows the system operation is blocking after place p_7 is marked. This is because we used the front sensor of the piston to indicate whether a workpiece was present or not ($Y(t_6) = sl$ AND sb, $Y(t_9) = sl$ AND sf). In the same time the interlock specifications require that the manipulator and the piston must not be in the left/front position simultaneously. The supervisor therefore blocks *al* (arm left) signal and since this is the required action of p_7, the operation is blocked.

The blocking situation is solved by replacing sf by *si* and *sb* by NOT *si* in the related transition conditions and moving action $(af,0)$ from p_8 to p_7. A new IO-reachability graph consists of 164 nodes and 312 transitions. It shows the operation is still blocking when the place p_6 is entered through t_{18} and t_{13}. The problem is again the potential collision of the manipulator and the piston - the manipulator is in left position and the forward movement of the piston is blocked. The problem is solved by moving the outgoing arc of t_{13} from p_6 to p_5 (dashed line in Fig. 5) and adjustment of the place actions ($(ag0,0)$ in p_5).

A new analysis shows the operation is now non-blocking. The corresponding IO-reachability graph consists of 108 nodes and 191 transitions. The required changes of the transition conditions and place actions are: $Y(t_6) = sl$ AND NOT *si*, $Y(t_9) = sl$ AND *si*, $Z(p_5) = \{(ar,1),(ag0,0)\}$, $Z(p_7) = \{(al,1),(ar,0),(ag0,0),(af,0)\}$, $Z(p_8) = \{(al,0),(ag1,1)\}$. With the resulting RTPN a code generator is started, which automatically builds a SFC for the given case.

4. 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 approach also 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. Such a prototype tool for automatic code generation in a form of IEC 61131-3 compliant function blocks has already been implemented. Only a small amount of additional programming is then needed to obtain an operating logic controller.

REFERENCES

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

Frey, G. and L. Litz (2000). Formal methods in plc-programming. In: *Proc. of the SMC'2000*.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proc. IEEE* **77**, 541–580.

Mušič, G. and D. Matko (2003). Petri net control of systems under discrete-event supervision. In: *ECC'03 European Control Conference*. Cambridge, UK.

Mušič, G., B. Zupančič and D. Matko (2002). Model based programmable control logic design. In: *Preprints of the 15th Triennial IFAC World Congress*. Barcelona, Spain.

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.

Vaz, A.F. and W.M. Wonham (1986). On supervisor reduction in discrete-event systems. *International J. Control* **44**, 475–491.

Wonham, W.M. (2003). *Notes on Control of Discrete Event Systems: ECE 1636F/1637S 2003-2004*. Systems Control Group, Dept. of ECE, University of Toronto.

Zhou, M. and E. Twiss (1998). Design of industrial automated systems via relay ladder logic programming and petri nets. *IEEE Trans. on Systems, Man, and Cybernetics - Part C* **28**, 137–150.