Petri Net Based Control of a Modular Production System

Gašper Mušič, Drago Matko
University of Ljubljana
Faculty of Electrical Engineering
Tržaška 25, SI-1000 Ljubljana, Slovenia
gasper.music@fe.uni-lj.si

Abstract— **A systematic approach to design of sequential control is presented. It is illustrated by a case study of a two level hierarchical and distributed control design for a laboratory scale modular production system. Our first goal is to show how Petri net models can be used as a tool for sequential control specification that enables systematic development and analysis of specification model. Secondly, we show how the same specification model can be used as an input data for the design of the second, coordination level employing supervisory control concepts. Finally, an implementation method is presented, which enables Petri net supervisor to be separated from the low level process model and implemented in a distributed environment.**

I. INTRODUCTION

Sequential or binary control is the predominant type of control in a large number of automated manufacturing systems. Specifications where a proper operation sequence is of a significant importance are typical for this kind of systems. In such cases, designers are mostly interested in the discrete behaviour of the process that can be described by discrete states and discrete events. This type of a system is characterised as a discrete event dynamic system.

Discrete event systems and related control issues gained a large attention within the research community in the past few years. Many successful applications of developed theoretical tools, such as Petri nets [8], are reported in modelling, design and management of various types of systems.

Much work has been devoted to the supervisory control introduced by Ramadge and Wonham [11]. Their work is based on automata and formal languages. Several supervisory control approaches based on Petri net models have also been developed in the last years. The most of research has been initiated by the academy and not by practical needs. The majority of related papers is therefore strongly theoretical and does not pay much attention to the possible implementation.

In this paper we combine the Petri net modelling framework and supervisory control concept in a sequential control case study that was performed by standard industrial equipment. This equipment consists of programmable logic controllers interconnected by a communication network and corresponding programming software. The main elements of the presented design approach are summarised in the following three sections. Section II briefly presents Petri nets and how they are used as a control specification tool. Supervisory control approach to design of a coordination layer is presented in Section III. Implementation issues are discussed in Section IV. Finally, the case study that combines presented topics is described in Section V.

II. PETRI NETS AND SEQUENTIAL CONTROL

As a large number of manufacturing systems is predominantly discrete the design of corresponding control actions requires a discrete event view of the system. Various discrete event modelling techniques can be used to obtain a model of a plant and a controller.

Petri nets as a tool for modelling and specification of manufacturing systems are described in a number of sources, such as [1], [8], [10]. A 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 $\mathrm{PN} = (P, T, F, W, m_0)$, 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 $P \cap T = \emptyset$),
$F \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs,
$W : F \to \{1, 2, \ldots\}$ is a weight function,
$m : P \to \{0, 1, 2, \ldots\}$ 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 arc weights are equal to one and no time is involved in the firing of transitions. 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 (includes no delay) 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.

A Petri net is *pure*, if it does not contain self loops, i.e. $\forall p_i \in P, \forall t_j \in T, (p_i, t_j) \in F \Rightarrow (t_j, p_i) \notin F$. For such a Petri net the connections among nodes in the net can be uniquely described by an incidence matrix $\mathbf{D}$. For an ordinary Petri net an element of the incidence matrix is defined as

$$d_{ij} = \begin{cases} 1, & \text{if } (t_j, p_i) \in F \\ -1, & \text{if } (p_i, t_j) \in F \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

For the purpose of simulation and possible implementation by industrial controllers, the input/output interpretation can be added to resulting models.

Several properties of Petri net models have been defined and investigated by different authors. For the

Proceedings of the IEEE International Symposium on Industrial Electronics - ISIE '99, Bled, Slovenia, July 12 - 16, 1999, vol. 3, pp. 1383-1388.

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], [10].

One of important related questions is how to derive an appropriate Petri net model. Different Petri net modelling approaches are described in [5], [14]. In our case Petri nets are primarily used for modelling a desired behaviour of the system. We start with a simple model that is refined in a top-down manner until the implementation level is reached. Developed model then serves as a specification of the required state transition structure in the control logic that would, when applied to the system under consideration, result in the desired behaviour.

Even if the Petri net model does not include all the details about controller input/output behaviour, it is advantageous to build such a model first. This enables a clear separation between state transition structure and implementation details. Above that it yields a possibility to analyse certain properties such as safeness, which is required in order for control logic to behave properly. In this way the designer is forced to focus on building a proper state transition structure, which is then used as a base of the sequential control logic.

Other modelling and specification approaches, e. g. finite automata, could be used to perform this task. As we require the Petri net models are bounded, or even safe in most cases, the descriptive power of such models 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. Several states can be therefore represented by a relatively small number of places. This enables a compact representation of concurrency and synchronisation, which makes Petri net models easy to understand.

III. COORDINATION AND SUPERVISORY CONTROL

Manufacturing systems are commonly build in a distributed manner. Resulting subsystems are independent to certain extent but also have to coordinate their actions with their environment. Coordination can include several sorts of activity, e.g. coordination in a sense of minimisation of a system production cycle time or maximisation of its throughput. In this paper we consider the coordination in its simplest sense, i.e. prevention of collisions and similar situations in order to maintain a proper system operation.

The coordination layer acting in this sense can be designed by the use of the supervisory control theory [11]. The supervisory control concept deals with discrete event systems whose behaviour is restricted by an external controller called supervisor.

Supervisory control (Fig. 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 in Fig. 1). In this way the supervisor actually intervenes only in cases when some undesired process

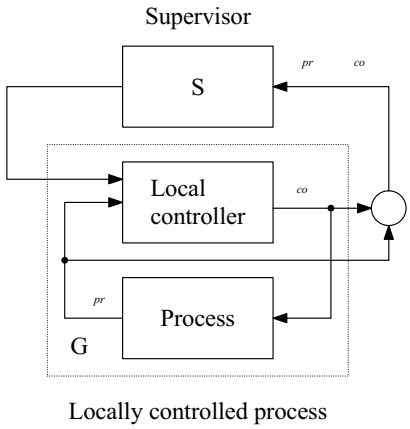

Fig. 1. Supervisory control.

behaviour is about to take place. The control effect is restricted to prevention of certain events in the system that are classified as controllable events.

The supervisor is computed based on the 'open-loop' system model that can be given as a finite automaton or a Petri net. The open-loop model (G in Fig. 1) represents a locally controlled process, it therefore includes the process and the local controller. Supervisory control approaches based on Petri net models are described in [3], [6], [7], [12]. Models of the desired behaviour of the locally controlled system are used as an open-loop model for the design of the supervisor.

In this way the same Petri net models as for control specification can be used to design a second control level that maintains additional system requirements, e.g. performs a simple coordination among subsystems. The set of derived Petri net models of separate subsystems is considered as a single open-loop process model whose behaviour should be restricted by a supervisor. Coordination requirements are translated into restrictions on the marking of the open-loop Petri net model. When the supervisor is designed by the method of place invariants [7], [12], the resulting supervisor consists of several additional places that are connected to existing open-loop Petri net model.

The method of place invariants is particularly interesting, because the resulting supervisory mechanism is computed very easily. The net marking m is denoted by a column vector $\mathbf{m}_p$, whose length equals the number of places P in the net and whose elements are $\mu_i = m(p_i)$. A vector $\mathbf{m}_p$ represents the state of an open-loop model or a plant. By the method of place invariants it is possible to enforce a set of n_c constraints on the plant state $\mathbf{m}_p$. Constraints are written in the form

$$\sum_{i=1}^{k} l_i \mu_i \leq \beta \tag{2}$$

or grouped together as

$$\mathbf{L}\mathbf{m}_p \leq \mathbf{b} \tag{3}$$

The inequality (3) is read with respect to each element on the corresponding left and right hand sides. It is shown in [12] that if the initial marking does not violate

the given set of constraints, (3) can be enforced by a supervisor with the incidence matrix

$$\mathbf{D}_c = -\mathbf{L}\mathbf{D}_p \tag{4}$$

where $\mathbf{D}_p$ is the incidence matrix (1) of the plant. The initial marking of the supervisor is computed by

$$\mathbf{m}_{c_0} = \mathbf{b} - \mathbf{L}\mathbf{m}_{p_0} \tag{5}$$

where $\mathbf{m}_{p_0}$ is the initial marking vector of the plant. The supervisor consists of n_c places that are linked to the existing transitions of the plant. With the addition of supervisory places the overall system is given by

$$\mathbf{D}_s = \begin{bmatrix} \mathbf{D}_p \\ \mathbf{D}_c \end{bmatrix} \qquad \mathbf{m}_s = \begin{bmatrix} \mathbf{m}_p \\ \mathbf{m}_c \end{bmatrix} \tag{6}$$

and every single constraint is transformed to a marking invariant that corresponds to a place invariant [1] of the supervised system.

In fact, every marking constraint results in an additional place, which enforces a marking invariant to the closed loop system. The added place acts as a mutual exclusion mechanism such as parallel or sequential mutual concept introduced in [13]. However, the supervisory control concept enables to consider some transitions in the process model as uncontrollable. These are transitions that are either generated by the process itself and can not 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. This means the resulting closed-loop system meets the demands about the system behavioural restrictions, while the supervisor never tries to block an uncontrollable transition and at the same time does not restrict the system more than necessary.

IV. IMPLEMENTATION ISSUES

Perhaps the most significant advantage of the Petri net representation is the straightforward path from the developed specification models to the industrial implementation. The discrete control logic is most often implemented by programmable logic controllers. The recent IEC 1131.3 standard [4] on programming languages of industrial logic controllers promotes the use of Sequential Function Chart (SFC) for the structuring of the control logic. SFC (also referred as Grafcet) inherited many of its features from the theory of Petri nets. More precisely, a safe interpreted Petri net can be defined such that its input-output behaviour is the same as the input-output behaviour of the SFC [1], [2].

A place in such a Petri net corresponds to a step in the SFC. Transitions and directed links have the same meaning in SFC as they have in Petri net. If an input/output interpretation is added to the transitions and places of the Petri net, we can obtain an equivalent SFC model. There are however two basic differences between such an interpreted Petri net and a SFC. The first difference between Petri nets and SFCs is that the marking of a SFC is Boolean (step is active or inactive) while the marking of a place in a Petri net can be any positive integer. For that reason the conversion of a Petri net to a SFC is only possible when the net is safe (i.e., for any reachable marking, the marking of every place is less than or equal to one).

The second difference between Petri nets and SFCs is that the firing rule of a SFC is different from a Petri net when there is a conflict. A conflict in a Petri net describes the situation when some transitions are enabled by the marking of the same place. This leads to a non-deterministic behaviour since there is no rule to choose which of them will be fired. When such situation emerges in a SFC the transitions are fired according to their priorities to ensure the deterministic behaviour.

Now, if a Petri net is such that any pair of transitions in conflict has transition conditions or receptivities, which can not be true at the same time, the behaviour of the net is deterministic. If such a Petri net is also safe, it is equivalent to a SFC [2]. Its strong relation to Petri net theory then enables a SFC to be directly redrawn from a Petri net model and the classical properties of Petri nets, such as marking invariants, can be applied also to SFCs.

In the proposed approach a model of the desired behaviour represents an unambiguous specification of state transitions in the corresponding control logic. Following the standard we use SFC as a core programming language. As described above, the structure of the Petri net model can be under certain conditions directly transformed into the control logic. In our case, every single specification model is translated into a SFC that is implemented in a local programmable logic controller.

Transition conditions and step actions are added during implementation phase and have to be written in one of the other standardised languages. It is important to include safety interlocks in the step actions where necessary, despite the coordination level above.

Similarly as the local control logic the supervisor is transformed into a sequential function chart. A solution is proposed which enables a separate implementation of the supervisor and the local control logic [9]. Special attention was paid to the situation when the supervisor and the local level control are implemented on different controllers and the possible communication delays must be taken into account. The solution is based on the assumption that every supervisory place can be interpreted as the mutual exclusion mechanism, such as in the example in Fig. 2a. Such a mechanism can be split into several nets, provided certain transitions are synchronised. Separated nets can be then transformed into separate SFCs, which can be implemented in distributed controllers. Synchronisation among transitions in these SFCs is accomplished through binary synchronisation flags that are shared over the controller network.

Because of the communication delays that could occur on the network it can not be guaranteed that transitions in different controllers fire simultaneously. This can be solved by defining the firing order of the transitions. In the cases when the two controllers share the same resource and the supervisor performs the re-

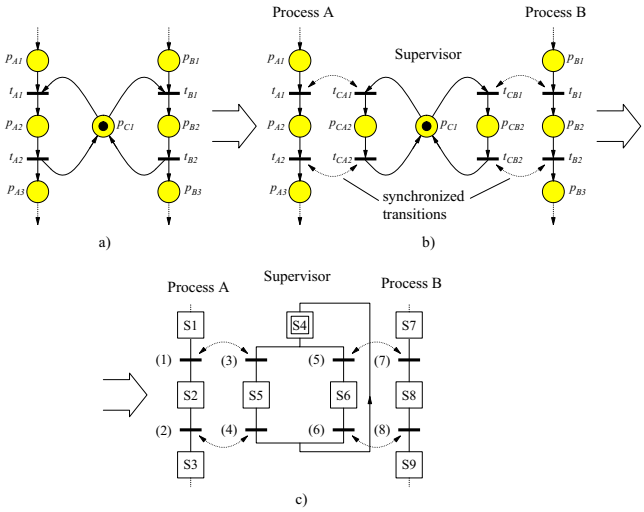

Fig. 2. Supervisor implementation in a distributed environment.

source allocation such as indicated in Fig. 2 the transition that books the resource must be fired in the supervisor first and only then in the local controller. In the opposite case the communication delay would allow a double booking of the shared resource by the two controllers.

This is solved as indicated in Fig. 3. Flags X_i are included in SFC transition conditions that signalise when step i in another SFC is active. In addition to these flags, the priority of the competing transitions in the supervisory part of the SFC has to be determined to ensure deterministic behaviour. This can be fixed through receptivities of the competing transitions, such as in Fig. 3 where transition (5) has higher priority than transition (3), or can be determined on-line by some higher level control subsystem, e.g. resource planning or scheduling system.

The proposed solution can be inadequate in certain situations, e.g. when the step whose activity is interpreted as a demand for a shared resource (such as step S1 in Fig. 3) is the only step in a chart which does not use a resource. The releasing of the resource performed through transitions (4) and (6) in the supervisory SFC in Fig. 3 does not work in this case. This can be solved by adding a step-transition pair in every branch of the supervisor as shown in Fig. 4.

The additional transition (3') fires when the process

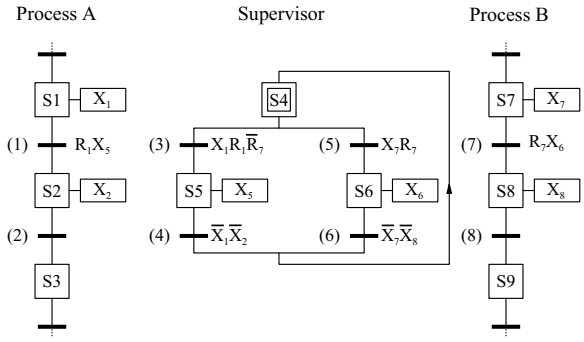

Fig. 3. Synchronisation between supervisor and local controllers.

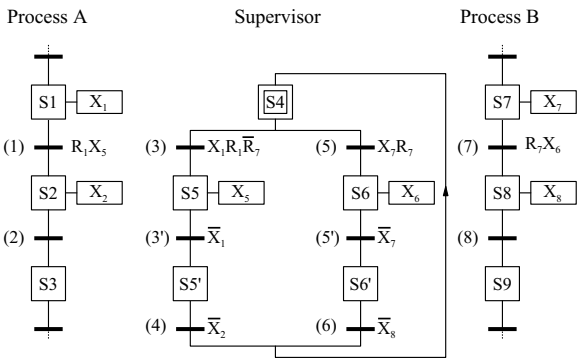

Fig. 4. Extended supervisor.

A starts using the resource, i.e. when the step S1 is no longer active. The inclusion of the condition $\overline{X}_1$ in the receptivity of transition (4) is no longer necessary and there are no problems with releasing the resource.

V. CONTROL OF A MODULAR PRODUCTION SYSTEM

Although simplified, the modular production system (Fig. 5) used in the case study incorporates several flexible automation concepts generally found in the area of manufacturing systems and used in modern production facilities. These include distributed design of the underlying control system and interaction of several subprocesses that require coordination among them.

System is divided into five partially independent working stations that are controlled by separate programmable logic controllers. For the purpose of coordination among stations the controllers are connected to a communication network. This enables them to share a set of register addresses. One of the controllers is dedicated as the network master others act as slaves. Although this makes no difference among controllers from the user program viewpoint, the network master also performs the second level supervisory function, i.e. coordination among working stations, which runs as an independent program task.

The desired behaviour of every station to be controlled is modelled by a Petri net. A top down synthesis procedure is applied which guarantees the de-

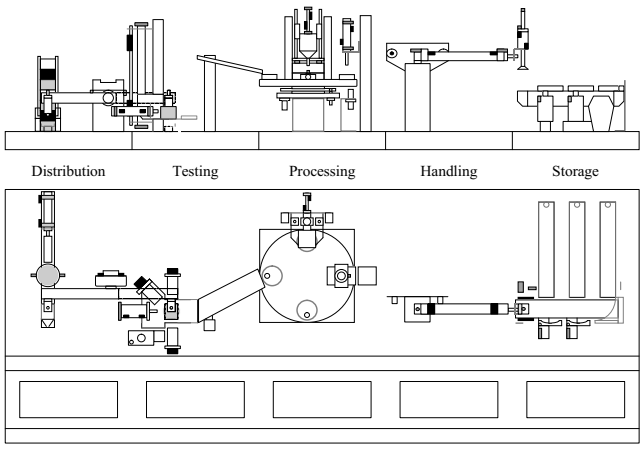

Fig. 5. Modular production system.

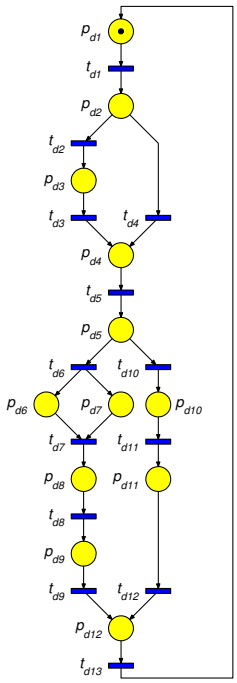

Fig. 6. Petri net model of the first working station.

TABLE I

CONTROL INTERPRETATION OF PLACES AND TRANSITIONS

Places	
p_{d1}	initial state
p_{d2}	delay (waiting for a workpiece)
p_{d3}	distribution piston moving forward
p_{d4}	manipulator arm moving toward input buffer
p_{d6}	holding the workpiece
p_{d7}	distribution piston moving backward
p_{d8}, p_{d10}	manip. arm moving toward lift
p_{d9}	releasing the workpiece
p_{d11}	buffer empty (waiting for a new workpiece)
p_{d5}, p_{d12}	no operation (linking places)
Transitions	
t_{d1}	controller in operation
t_{d2}	end of delay and input buffer not empty
t_{d3}	distribution piston forward (switch closed)
t_{d4}	end of delay and input buffer empty
t_{d5}	manip. arm at input buffer (switch closed)
t_{d6}	distribution piston forward (switch closed)
t_{d7}	vacuum on (vacuum switch closed) and distribution piston back (switch closed)
t_{d8}, t_{d11}	manip. arm at second station (switch closed)
t_{d9}	vacuum off (vacuum switch open)
t_{d10}	distribution piston not forward (switch open)
t_{d12}	new workpiece detected (optical sensor)
t_{d13}	no condition (fires immediately)

rived model has desired properties of liveness, boundedness and reversibility. The procedure starts with simple net and continues by stepwise refinement of places and/or transitions until the sufficiently detailed model is obtained. An example of such model is shown in Fig. 6. The control logic based on the model is then implemented as a SFC in an IEC 1131.3 compliant programming environment. The control implementation of places and transitions is listed in Table I.

Models of the desired behaviour of the locally controlled system are used as an open-loop model for the design of the supervisor. In the given case the described concept was used to synthesise a simple coordinator that prevents mechanical collisions among parts of the neighbouring working stations and maintains proper handling of workpieces travelling from one station to another. Computed supervisory mechanism is implemented as a set of SFCs in the same programming software as the local control logic.

The whole system was practically implemented and tested. Fig. 7 shows the SFC that maintains the main operational sequence of the first working station, the corresponding SFC for the second station is shown in Fig. 8. The two stations have to be coordinated because the manipulator arm periodically enters the area of the lift movement. The SFC of the supervisor that prevents the potential collisions is shown in Fig. 9. Related variables are listed in Table II.

VI. CONCLUSIONS

Presented approach can be used in solving various practical design problems in the area of manufacturing systems. The complexity of the Petri net models can be kept manageable due to the distributed design of the system. The proposed coordination approach minimally affects local control logic and can be designed separately. One of the problems of such a design is that used supervisory control method generally does not give a blocking-free supervisor. The supervised system may therefore include deadlocks. An analysis of a resulting closed loop Petri net model has to be performed and system eventually has to be modified to avoid such situations. A deadlock free coordination design is one

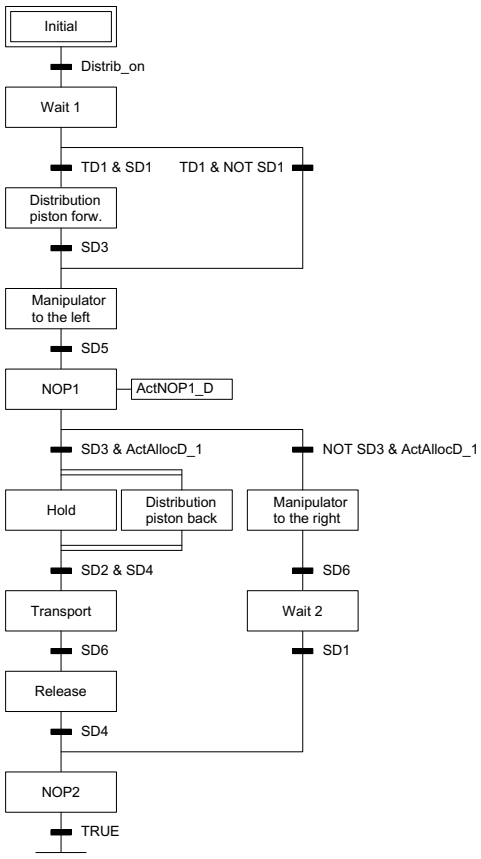

Fig. 7. Control implementation - distribution station.

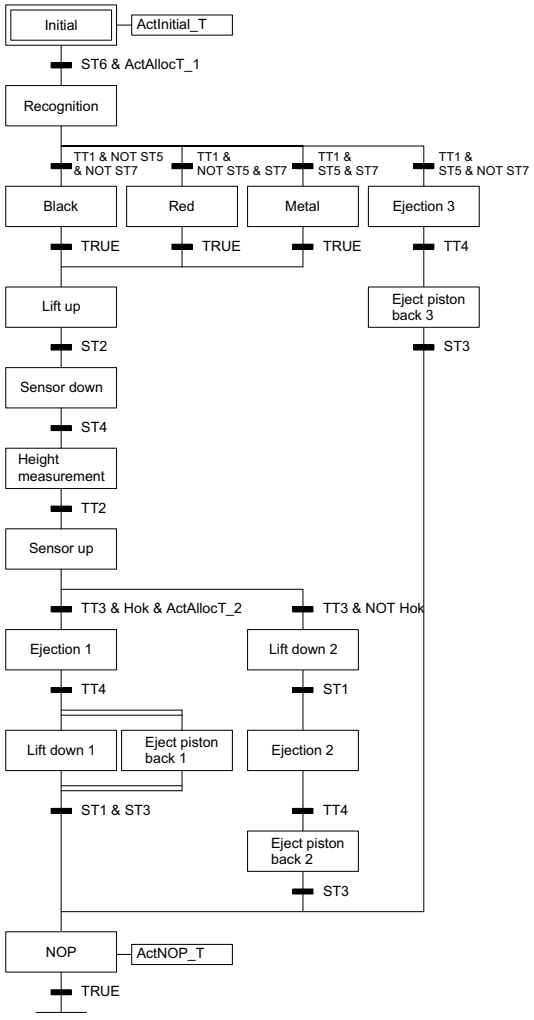

Fig. 8. Control implementation - testing station.

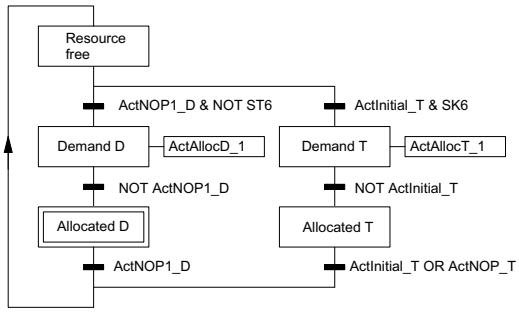

Fig. 9. Control implementation - supervisor 1.

of the issues for the further work. Another important drawback is the difficult translation of the coordination requirements into behavioural restrictions suitable for supervisory control synthesis. Only part of these requirements can be fulfilled by such a supervisor while some tasks still have to be programmed intuitively. A possible inclusion of these requirements in the scope of the presented approach is also one of the further research issues.

TABLE II

Variables used in sequential function charts

Distribution station:	
SD1	Sensor - workpiece present
SD2	Distribution piston position switch - back
SD3	Distribution piston posit. switch - front
SD4	Vacuum sensor
SD5	Manipulator position switch - left
SD6	Manipulator position switch - right
Distrib_on	Global variable - station is working
TD1	Time delay run out
ActNOP1_D	Network variable - step *NOP1* in the distribution control program active
ActAllocD_1	Network variable - allocation step in the supervisor 1 active
Testing station:	
ST1	Lift position switch - down
ST2	Lift position switch - up
ST3	Eject piston position switch - in
ST4	Height sensor position switch - down
ST5	Inductive sensor at lift platform
ST6	Capacitive sensor at lift platform
ST7	Optical sensor at lift platform
Hok	Result of the height measurement
TT1, TT2	Time delay run out
TT3, TT4	Time delay run out
ActInitial_T	Network variable - step *Initial* in the testing control program active
ActNOP_T	Network variable - step *NOP* in the testing control program active
ActAllocT_1	Network variable - allocation step in the supervisor 1 active
ActAllocT_2	Network variable - allocation step in the supervisor 2 active (coordinates testing and processing stations; not shown here)

References

[1] R. David and H. Alla, "Petri Nets for Modeling of Dynamic Systems - A Survey", *Automatica*, vol. 30, no. 2, 1994, pp. 175-202.

[2] R. David, "Grafcet: A Powerful Tool for Specification of Logic Controllers", *IEEE Trans. on Control Systems Technology*, vol. 3, no. 3, 1995, pp. 253-268.

[3] L. E. Holloway, B. H. Krogh and A. Giua, "A Survey of Petri Net Methods for Controlled Discrete Event Systems", *Discrete Event Dynamics Systems: Theory and Applications*, vol. 7, 1997, pp. 151-190.

[4] IEC, International Electrotechnical Commission, "Programmable Controllers - Part 3: Programming Languages", publication 1131.3, 1993.

[5] M. D. Jeng and F. DiCesare, "A Review of Synthesis Techniques for Petri Nets with Applications to Automated Manufacturing Systems", *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 23, no. 1, 1993, pp. 301-312.

[6] B. H. Krogh and L. E. Holloway, "Synthesis of Feedback Control Logic for Discrete Manufacturing Systems", *Automatica*, vol. 27, no. 4, 1991, pp. 641-651.

[7] J. O. Moody and P. J. Antsaklis, "Supervisory Control of Petri Nets with Uncontrollable/Unobservable Transitions", *Tech. Rep. ISIS-96-004*, University of Notre Dame, 1996.

[8] T. Murata, "Petri Nets: Properties, Analysis and Applications", *Proc. IEEE*, vol. 77, no. 4, 1989, pp. 541-580.

[9] G. Mušič and D. Matko, "Petri Net Based Supervisory Control of Flexible Batch Plants", in *Prepr. 8^{th} IFAC Symp. on Large Scale Systems: Theory & Application*, vol. II, Rio Patras, 1998, pp. 989-994.

[10] J. M. Proth and X. Xie, *Petri Nets, A Tool for Design and Management of Manufacturing Systems*, Wiley (UK), Chichester, 1996.

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

[12] K. Yamalidou, J. Moody, M. Lemmon and P. Antsaklis, "Feedback Control of Petri Nets Based on Place Invariants", *Automatica*, vol. 32, no. 1, 1996, pp. 15-28.

[13] M. C. Zhou and F. DiCesare, "Parallel and Sequential Mutual Exclusions for Petri Net Modeling of Manufacturing Systems with Shared Resources", *IEEE Trans. on Robotics and Automation*, vol. 7, no. 4, 1991, pp. 515-527.

[14] M. Zhou, F. DiCesare and A. A. Desrochers, "A Hybrid Methodology for Synthesis of Petri Net Models for Manufacturing Systems", *IEEE Trans. on Robotics and Automation*, vol. 8, no. 3, 1992, pp. 350-361.