21

Real-Time Petri Net Based Control System Design for Distributed Autonomous Robotic Manufacturing Systems

Gen'ichi Yasuda
Nagasaki Institute of Applied Science, Japan

1. Introduction

Generally speaking, flexible manufacturing systems are made up of some flexible production machines with some local storage facilities for tools and parts, some handling devices such as robots and a versatile transportation system. It is expected that more and more robots will be introduced into manufacturing systems to automate various operations in the near future. However, it is quite obvious that a single robot cannot perform effective tasks in an industrial environment, unless it is provided with some additional equipment that allows the machine to grasp, handle and dispose correctly workpieces or mechanical parts onto which technological operations are to be performed. Therefore, in order to avoid the need of loading and unloading of parts to the robot manually, it is usually required to integrate the robot into the production line that also includes machine tools, conveyors, and other special purpose machines. Mainly to provide flexibility to robots, a lot of researches have been done to develop an effective programming method for robots. But not much research has been done to integrate a system which includes various machines (robots and other devices) that cooperate in the same task (Holding & Sagoo, 1992). A common programming language for tasks that involve more than one robot or machine should be provided (Holt & Rodd, 1994).

Robot programs often must interact with people or machines, such as feeders, belt conveyors, machine tools, and other robots. These external processes are executing in parallel and asynchronously; therefore, it is not possible to predict exactly when events of interest to the robot program may occur. The programmable logic controllers (PLC) are widely used to the programming and control of flexible manufacturing systems. Implementation languages can be based on ladder diagrams or more recently state machines. However, when the local control is of greater complexity, the above kinds of languages may not be well adapted. It is important to have a formal tool powerful enough to develop validation procedures before implementation. Conventional specification languages such as ladder diagrams do not allow an analytical validation. Presently, the implementation of such control systems makes a large use of microcomputers. Real-time executives are available with complete sets of synchronization and communication primitives (Yasuda, 2000). However, coding the specifications is a hazardous work and debugging the implementation is particularly difficult when the concurrency is important.

The Petri net and its graphical representation is one of the effective means to describe control specifications for manufacturing systems. From the plant control perspective, the role and the presence of nets were considered in the scheduling, the coordination and the local control level (Silva, 1990). However, in the field of flexible manufacturing cells, the network model becomes complicated and it lacks the readability and comprehensibility. Therefore, the flexibility and expandability are not satisfactory in order to deal with the specification change of the control system. Despite the advantages offered by Petri nets, the synthesis, correction, updating, etc. of the system model and programming of the controllers are not simple tasks. The merging of Petri nets and knowledge based techniques seems to be very promising to deal with large complex discrete event dynamic systems such as flexible manufacturing systems (Gentina & Corbeel, 1987; Maletz, 1983; Wang & Sarides, 1990).

The aim of this chapter is to introduce manufacturing engineering specialists to the basic system level issues brought up by the development of computer-controlled robotic manufacturing systems and how Petri nets are applied to resolve the above mentioned problems of control system design. After some terminology concerning basic Petri nets, the extensions of Petri nets for manufacturing system control are briefly reviewed. Based on the hierarchical and distributed structure of the manufacturing system, the net model of the system is decomposed into a set of interacting local nets and a system coordinator net to perform distributed autonomous multitasking control based on Petri nets.

2. Modeling of discrete event manufacturing systems with Petri nets

The Petri net is one of the effective means to represent discrete event manufacturing systems. Considering not only the modeling of the systems but also the well-defined control, the guarantee of safeness and the capabilities to represent input and output functions are required. Therefore the Petri net has been modified and extended.

2.1 Modification of basic Petri nets

A Petri net is a directed graph whose nodes are places shown by circles and transitions shown by bars. Directed arcs connect places to transitions and transitions to places. Formally, a Petri net is a bipartite graph represented by the 4-tuple $G = \{P, T, I, O\}$ (Murata, 1989) such that:

$P = \{p_1, p_2, ..., p_n\}$ is a finite, not empty, set of places;

$T = \{t_1, t_2, ..., t_m\}$ is a finite, not empty, set of transitions;

$P \cap T = \phi$, i.e. the sets P and T are disjointed;

$I : T \to P^{\infty}$ is the input function, a mapping from transitions to bags of places;

$O : T \to P^{\infty}$ is the output function, a mapping from transitions to bags of places.

The input function I maps from a transition t_j to a collection of places $I(t_j)$, known as input places of a transition. The output function O maps from a transition t_j to a collection of places $O(t_j)$, known as output places of a transition.

Each place contains integer (positive or zero) marks or tokens. The number of tokens in each place is defined by the marked vector or marking $M = (m_1, m_2, ..., m_n)^T$. The number of

tokens in one place p_i is simply indicated by $M(p_i)$. The marking is shown by dots in the places. The marking at a certain moment defines the state of the net, or the state of the system described by the net. The evolution of the state therefore corresponds to an evolution of the marking, caused by the firing of transitions. The firing of an enabled transition will change the token distribution (marking) in a net according to the transition rule. In a basic Petri net, a transition t_j is enabled if $\forall p_i \in I(t_j)$, $M_k(p_i) \geq w(p_i, t_j)$, where the current marking is M_k and $w(p_i, t_j)$ is the weight of the arc from p_i to t_j.

Because discrete event manufacturing systems are characterized by the occurrence of events and changing conditions, the Petri net type considered is the condition-event net, in which conditions can be modeled by places whilst events can be modeled by transitions. Events are actions occurring in a system. The occurrence of these events is controlled by system states. Because the condition-event system is essentially asynchronous, events always occur when their conditions are satisfied. Consequently, bumping occurs when despite the holding of a condition, the preceding event occurs. This can result in the multiple holding of that condition. From the viewpoint of discrete event process control, bumping phenomena should be excluded. So, the firing rule of the basic Petri net should be modified so that the system is free of this phenomenon. Thus the axioms of the modified Petri net are as follows:

1. A transition t_j is enabled if for each place $p_k \in I(t_j)$, $m_k = 1$ and for each place $p_l \in O(t_j)$, $m_l = 0$;
2. When an enabled transition t_j is fired, the marking M is changed to M', where for each place $p_k \in I(t_j)$, $m'_k = 0$ and for each place $p_l \in O(t_j)$, $m'_l = 1$;
3. In any initial marking, there must not exist more than one token in each place.

The number of arcs terminated at or started from a place or a transition is unlimited, but at most one arc is allowed between a transition and a place. According to these axioms, the number of tokens in each place never exceeds one, thus, the modified Petri net is said to be a safe graph. The modified Petri net is a subclass of the Petri net, and it is transformed into the equivalent Petri net as shown in Fig. 1.

Fig. 1. (a) A place in the modified Petri net and (b) its equivalent Petri net

2.2 Extensions for real-time control

The extended Petri net adopts the following elements as input and output interfaces which connect the net to its environment: gate arc and output signal arc. A gate arc connects a transition with a signal source, and depending on the signal, it either permits or inhibits the occurrence of the event which corresponds to the connected transition. Gate arcs are classified as permissive or inhibitive, and internal or external. When the signal is 1 (true), a permissive arc permits the occurrence of the event. On the other hand, an inhibitive arc inhibits the occurrence of the event when the signal is 1. An internal arc deduces the signal

from a place, and the signal is 1 when a token exists in the place, otherwise 0 (false). An external arc deduces the signal from an external machine. An output signal arc sends the signal from a place to an external machine. In addition to the axiom 1, a transition is enabled if it does not have any internal permissive arc signaling 0 nor any internal inhibitive arc signaling 1. An enabled transition is fired if it does not have any external permissive arc signaling 0 nor any external inhibitive arc signaling 1. Thus the enabling condition and the external gate condition are formally expressed as follows.

$$t_j = \bigcap_{m=1}^{M} p_{j,m}^{I} \wedge \bigcap_{n=1}^{N} \overline{p_{j,n}^{O}} \wedge \bigcap_{q=1}^{Q} g_{j,q}^{IP} \wedge \bigcap_{r=1}^{R} \overline{g_{j,r}^{II}} \tag{1}$$

$$g_j^E = \bigcap_{u=1}^{U} g_{j,u}^{EP} \wedge \bigcap_{v=1}^{V} \overline{g_{j,v}^{EI}} \tag{2}$$

where

M : set of input places of transition j
$p_{j,m}^{I}$: state of input place m of transition j
N : set of output places of transition j
$p_{j,n}^{O}$: state of output place n of transition j
Q : set of internal permissive gate signals of transition j
$g_{j,q}^{IP}$: internal permissive gate signal variable q of transition j
R : set of internal inhibitive gate signals of transition j
$g_{j,r}^{II}$: internal inhibitive gate signal variable r of transition j
U : set of external permissive gate signals of transition j
$g_{j,u}^{EP}$: external permissive gate signal variable u of transition j
V : set of external inhibitive gate signals of transition j
$g_{j,v}^{EI}$: external inhibitive gate signal variable v of transition j

All the variables are logical binary variables, and $\wedge$, $\vee$ denote the logical product and the logical sum, respectively, and $\bigcap_{i=1}^{m} a_i = a_1 \wedge a_2 \wedge \ldots \wedge a_m$. The state (marking) change, that is, the addition or removal of a token of a place, is described as follows:

$$p_{j,m}^{I} = p_{j,m}^{I} \wedge \overline{(t_j \wedge g_j^E)} = RST(t_j \wedge g_j^E) \tag{3}$$

$$p_{j,n}^{O} = p_{j,n}^{O} \vee (t_j \wedge g_j^E) = SET(t_j \wedge g_j^E) \tag{4}$$

where $SET()$ and $RST()$ denote the set and the reset function, respectively.

Fig. 2 shows an example of extended Petri net model of robotic task control by transition firing with permissive and inhibitive gate arcs. The robot starts the loading operation based on signals from the switches, sends the commands through output signal arcs, and receive the status signals from the sensors through permissive gate arcs. Fig. 3 shows an example detailed net model of the lowest level local control of a machining center.

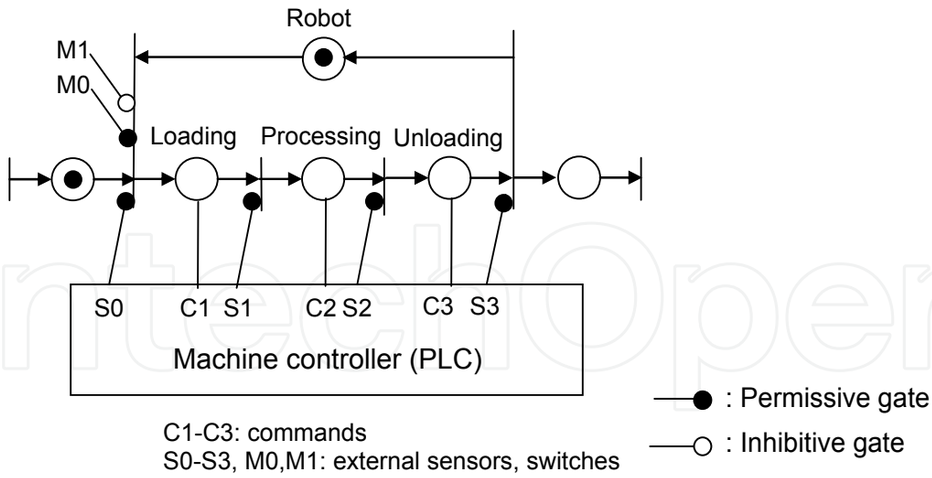


Fig. 2. Extended Petri net representation of robotic task with output signal arcs and gate arcs.

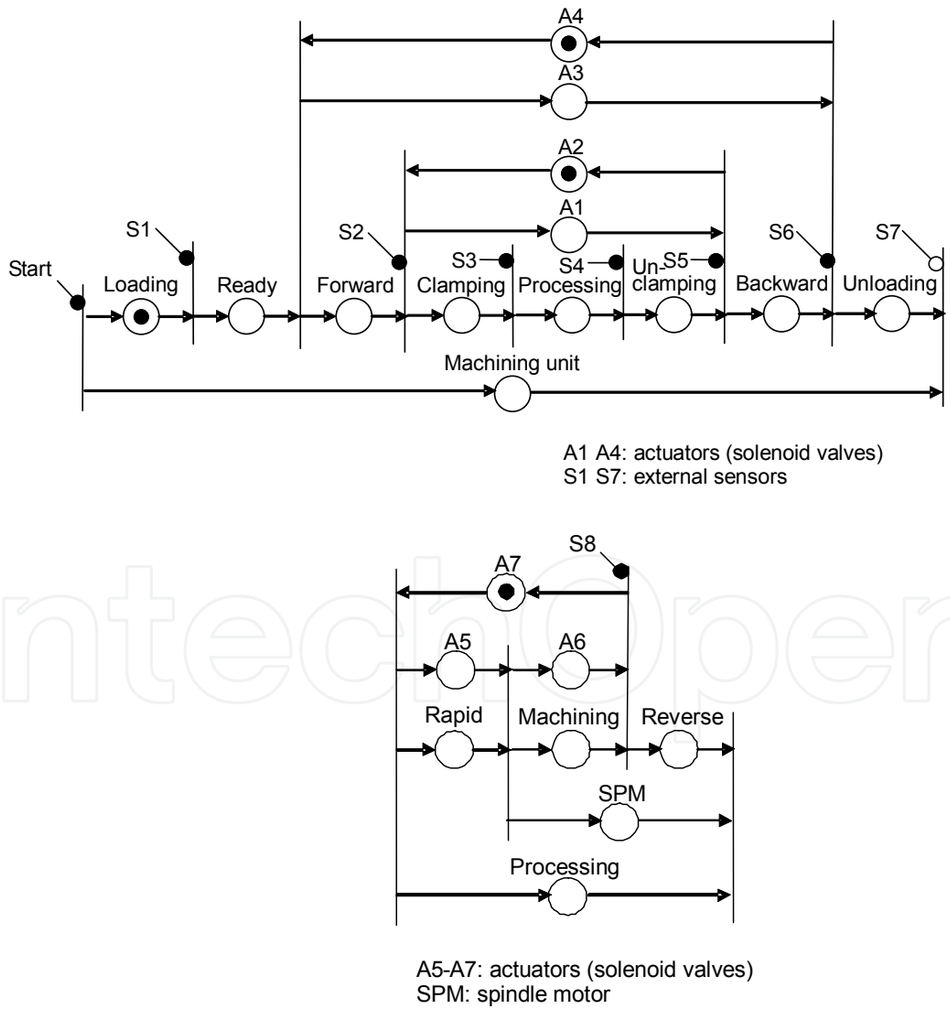


Fig. 3. Detailed net model of real-time control of manufacturing tool

3. Edition and simulation of net models

When programming a specific task, the task is broken down into subtasks. These subtasks are represented by a place. The internal states of machines are also represented by a place. The relations between these places are explicitly represented by interconnections of the transitions, arcs and gates. The whole task is edited with a net edition and simulation system. In parallel a graphic robot motion simulator system is used to edit a subtask program for a robot. The basic edition and simulation procedure is shown in Fig. 4.

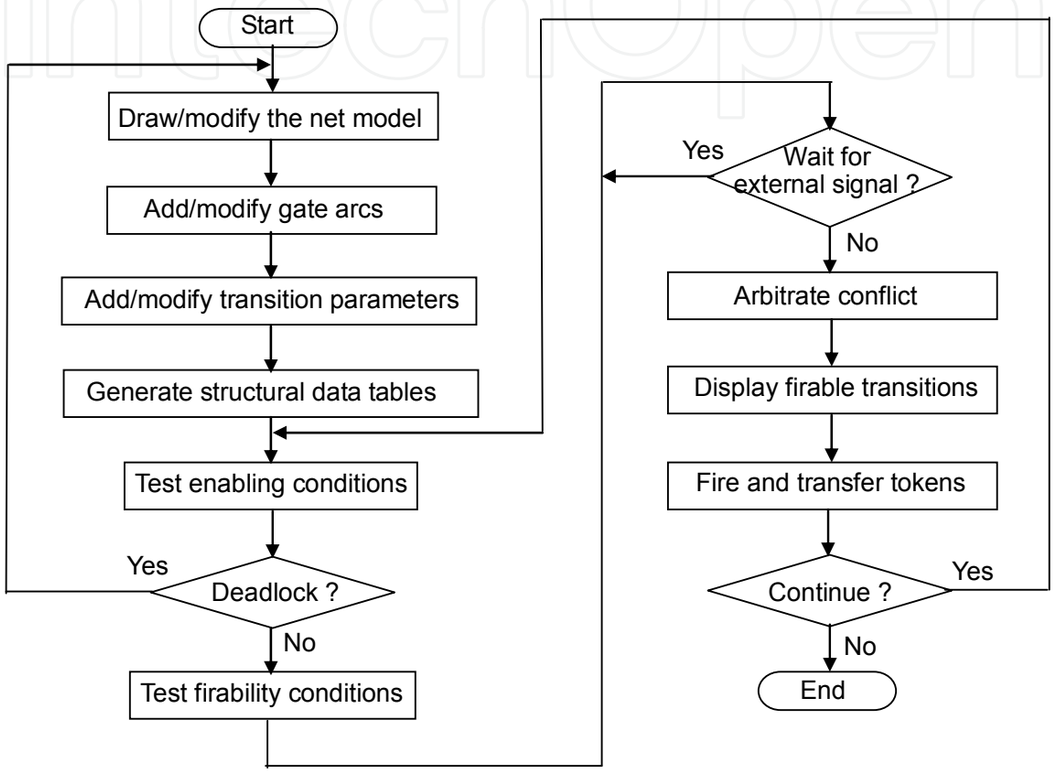


Fig. 4. Flow chart of net edition and simulation procedure

The net simulator is a tool for the study of condition-event systems and used to model condition-event systems through its graphical representation. When the net modeling is finished, the net is transformed into a tabular form and several data tables corresponding to the connection structure of the net are automatically generated (Yasuda, 2008). These tables are the following ones:

1. The table of the labels of the input and output places for each transition;
2. The table of the transitions which are likely to be arbitrated for each conflict place;
3. The table of the gate arcs which are internal or external, permissive or inhibitive, for each transition.

Although a variety of software implementations of Petri nets is possible using multitask processing (Taubner, 1988), a simple implementation method is adopted, where just one process is provided for the management of all places and tokens. Through the simulation steps, the transition vector table is efficiently used to extract enabled or fired transitions.

The table of marking indicates the current marking for each place. Using these data tables, the flow of the net simulation consists in the following steps:

1. Search enabled transitions using the axiom 1 or (1);
2. Test the enabled transitions considering gate conditions (2);
3. Arbitrate enabled transitions in conflict using some arbitration rule;
4. Execute transition firing and output corresponding signals to external machines;
5. Change the marking to the new marking using the axiom 2 or (3), (4) and update the system state.

The flow chart of the enabling condition test is shown in Fig. 5. The simulation algorithm is based on the execution rules of the net. The simulator tests each transition as to whether its input and output places and its internal gate arcs satisfy the enabling condition. If there is no enabled transition, it means that the net is in a deadlock condition. The simulator warns and requires the operator to change the initial marking or structure of the net. If there are some enabled transitions, it tests each of them as to whether its external gate arcs satisfy the firability condition, as shown in Fig. 6. If there is no firable transition, the simulator stops and shows which transitions are waiting for the gate signals.

For an example net as shown in Fig, 7, the enabling condition and the firability condition are written as (5), (6), respectively. The simulator tests each transition in the specified order of (5), (6). Fired transitions are memorized, and through their output places the output transitions of each place are searched. The enabling condition test is performed only for these transitions in order to shorten computation time. In Fig. 7, the enabling condition of only the transition t1 is evaluated, since the transition t5 is fired previously.

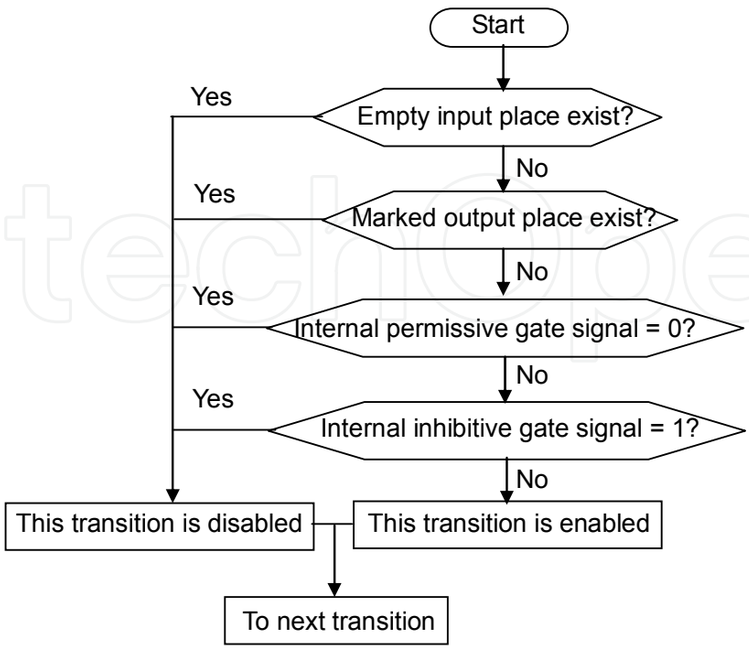


Fig. 5. Flow chart of enabling condition test

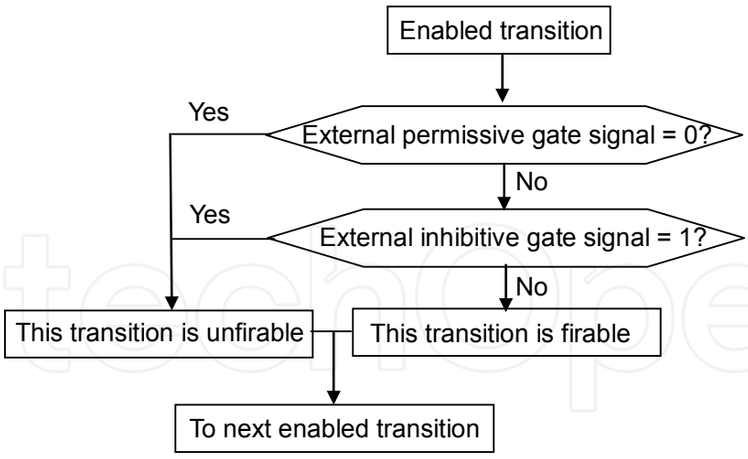


Fig. 6. Flow chart of firability condition test

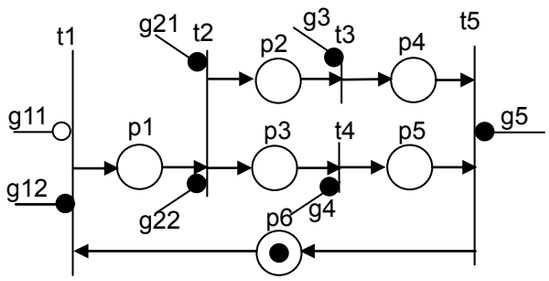


Fig. 7. Example of net representation with parallel activities.

$$
\begin{aligned}
t_1 &= p_6 \cdot \overline{p_1} \\
t_2 &= p_1 \cdot \overline{p_2} \cdot \overline{p_3} \\
t_3 &= p_2 \cdot \overline{p_4} \\
t_4 &= p_3 \cdot \overline{p_5} \\
t_5 &= p_4 \cdot p_5 \cdot \overline{p_6}
\end{aligned}
\tag{5}
$$

$$
\begin{aligned}
p_6 &= RST(t_1 \cdot \overline{g_{11}} \cdot g_{12}) \\
p_1 &= SET(t_1 \cdot \overline{g_{11}} \cdot g_{12}) \\
p_1 &= RST(t_2 \cdot g_{21} \cdot g_{22}) \\
p_2 &= SET(t_2 \cdot g_{21} \cdot g_{22}) \\
p_3 &= SET(t_2 \cdot g_{21} \cdot g_{22}) \\
p_2 &= RST(t_3 \cdot g_3) \\
p_4 &= SET(t_3 \cdot g_3) \\
p_3 &= RST(t_4 \cdot g_4) \\
p_5 &= SET(t_4 \cdot g_4) \\
p_4 &= RST(t_5 \cdot g_5) \\
p_5 &= RST(t_5 \cdot g_5) \\
p_6 &= SET(t_5 \cdot g_5)
\end{aligned}
\tag{6}
$$

If the transitions connected to a conflict place may happen to be in conflict, according to the rules of the net, only one of them is chosen to fire arbitrarily and the others become unfirable. The arbiter assigns the right of the order of firing among the transitions connected to a conflict place. But the right vanishes when the specified transition is not firable. The arbiter has a pointer to memorize the transition to be assigned the right next. The procedure of the arbitration is shown in Fig. 8. After the arbitration, all the firable transitions are displayed and fired. The simulator moves the tokens; it remove tokens in all the input places

of the fired transitions and put a token in each output place of the transitions. If some error is found or the simulation result does not satisfy the specification, it can be easily amended by reediting the net and by simulating it again. The edition and simulation are performed in an interactive form on a graphic display. The software written in Visual C# under OS Windows XP allows net models be modified on-line and simulation immediately restarted.

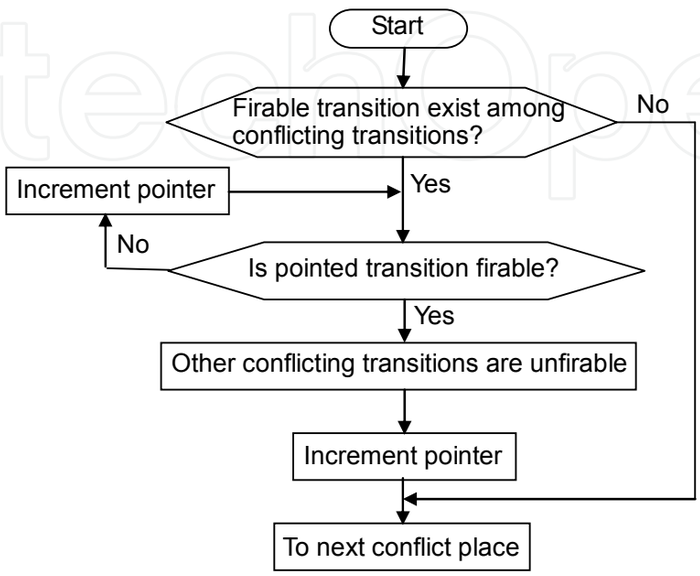


Fig. 8. Flow chart of arbitration procedure

In the basic Petri net, the firing of a transition is indivisible; the firing of a transition has duration of zero. The real-time performance of systems can be studied by adding time to the basic Petri net. An approach known as the timed Petri net associates a time parameter T with a transition, such that once the transition is enabled, it will fire after the period T. If the enabling condition is not satisfied before the schedule time comes, then the transition can not be fired and the passage of time is cancelled. Time values may be associated with places in order to maintain the instantaneous firing rule for transitions. A place with capacitance C_N, such as buffers in manufacturing systems, can be represented as a cascade connection of ordinary places with capacitance 1. The internal gate signal from the place is 1 when the number of tokens in the place is C_N, and 0 when the number is 0. These extensions are illustrated in Fig. 9(a) and (b).

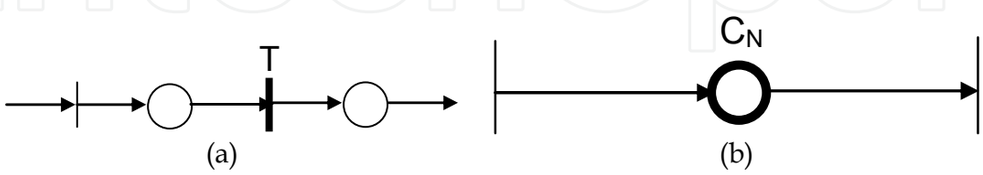


Fig. 9. Example of representation of (a) timed transition (b) place with capacitance N

4. Net models of multitasking control

Manufacturing tasks are a combination of several processes. These processes represent subtasks that are composed of task units. Tasks that include cooperative subtasks of different

machines are typical examples of concurrent processes. A system with one process is the degenerate case of a system of concurrent processes, which is obtained by combining nets representing several processes. Every sequential program can be represented by a flow chart. A flow chart is composed of nodes and arcs between them. It represents the flow of control in a program and can be represented by a Petri net, by replacing the nodes with places and the arcs with transitions as shown in Fig. 10. Each arc of the flow chart is represented by exactly one transition in the corresponding net. Petri net models of sequential constructs are shown in Fig. 11. A token residing in a place means that the program counter is positioned ready to execute the next instruction. Places for motion and computational actions have a unique output transition. Decision actions introduce conflict into the net. The choice can either be made nondeterministically or may be controlled by some external signal.

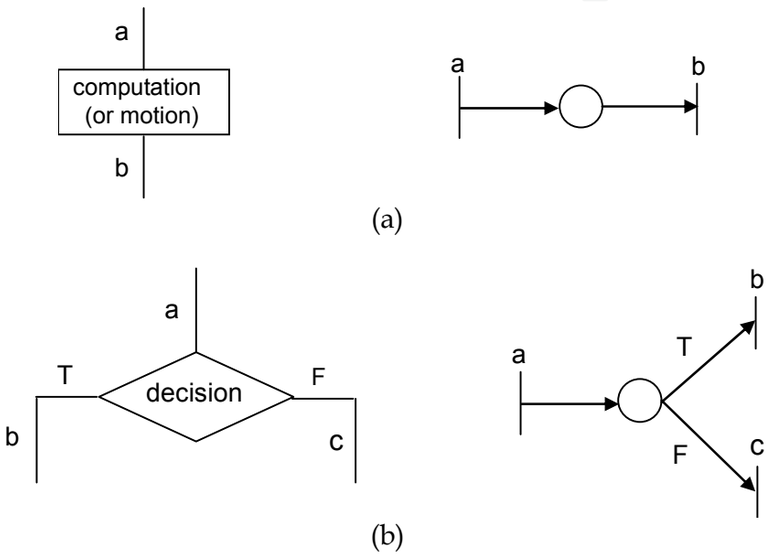


Fig. 10. Translation from nodes in a flow chart to places in a Petri net: (a) computation or motion, (b) decision

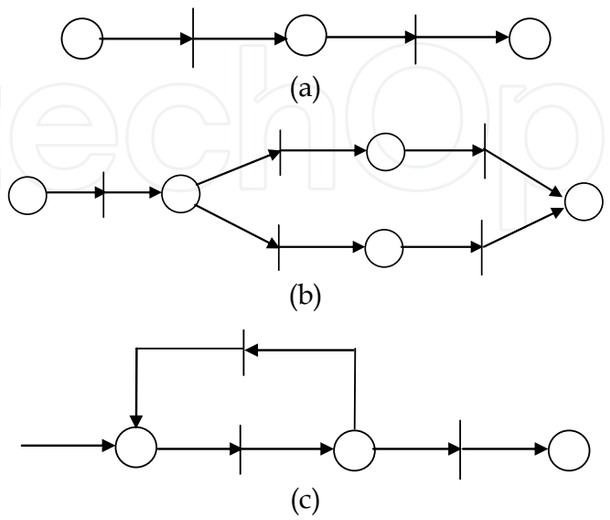


Fig. 11. Net representations of sequential constructs; (a) sequence, (b) decision, (c) iteration

In the case of two concurrent processes, where each process can be represented by a net model of a sequential process, the composite net which is simply the union of such nets can represent the concurrent execution of two processes. Parallelism is usefully introduced into a system only if the component processes can cooperate in the system. Such cooperation requires the sharing of information and resources between the processes. This sharing must be controlled to ensure correct operation of the overall system. One of the most popular synchronization mechanisms has been the P and V operations on semaphores. The WAIT and SIGNAL statements are used in a program written in a high level robot language and provides a variation of the P and V operations as a basic inter-process communication mechanism. Fig. 12 shows the net representation of an example of synchronization mechanism.

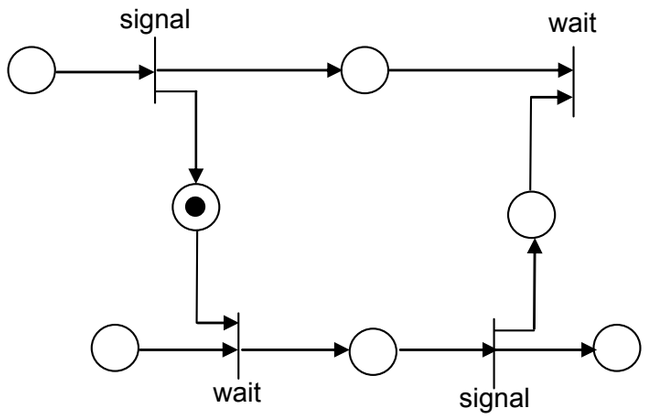


Fig. 12. Net representation of synchronization mechanism using asynchronous communication

Fig. 13 shows the net representation of cooperative operation using synchronization mechanism, where shared transitions require mutual synchronization between two robots. In contrast to decentralized implementation, synchronization can be also implemented by centralized coordination (Yasuda, 2010).

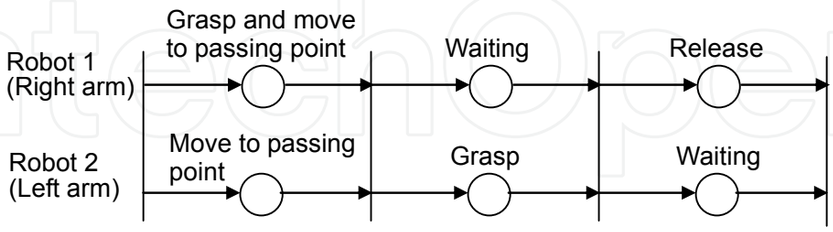


Fig. 13. Net representation of operation that passes a part from right arm to left arm

The main flow of execution control of robotic action using output signal arc and permissive gate arc is described as the following steps:

1. When a token is placed in a place which represents an action, the net based controller initiates the execution of the action (subtask) attached to the fired transition by sending the "start" signal through the output signal arc to the machine controller.

2. Then the machine controller interprets the request and runs the execution routine by sending the commands through serial interface to the robot or other external machine.
3. When the action is completed, the machine controller informs the system controller to proceed with the next activations by sending the "end" signal through external permissive gate arc.

When a token is placed in a place which is "ready" state in the net model, the controller sends the "ready" signal. If the machine receives the signal, it runs the processing routine which performs the initializations and other preliminary processing for the next execution routines. When the processing routine is completed, it sends the "ack" (acknowledgement) signal to the system controller. The "end" and "ack" signals work as gate signals for the system controller.

5. Implementation of real-time control system for robotic cells

To implement the Petri net based modeling and control method, the net based task editor and simulator, and the real-time controller based on tasks represented as net models were developed (Yasuda, 2008). The subtasks and sets of point data needed to execute the whole task are initially identified. Then they are edited and tested with the net based edition and simulation system. Initially, the proposed method is used to execute a simple example of pick-and-place task by a single robot. The experimental set up includes the following equipment: a small industrial robot with an arm (Mitsubishi Electric, Movemaster II RM501), two belt conveyors with their sequence control circuits, a NC machine tool and a general PC. All the software is written in Microsoft Visual C# on Windows XP. The task specification is represented as the flow of a workpiece and written as the following steps:

1. A workpiece arrives at point E1.
2. Conveyor CV1 carries the workpiece to point E2.
3. Robot R1 transfers the workpiece to point E3.
4. Machining operation M1 is done.
5. Robot R1 transfers the workpiece to point E4.
6. Conveyor CV2 carries the workpiece to point E5.

Synchronous cooperation is required to perform the loading and unloading operations between the robot and the conveyor or machining center. The cooperation can be implemented by a system coordinator which coordinates the machine controllers such that associated transitions of the local net models fire simultaneously. For high efficiency, it is desirable that the system accepts as many workpieces as possible, but it must not be in a deadlock condition. Generally, if there are some paths between two transitions, the largest number of tokens in each path is the smallest number of places of the paths. The task specification is shown as follows. Using the place of capacity control, the net representation of the task program written under these requirements is shown in Fig. 14.

Another example is a cooperative task by two arm robots which must synchronize their actions with each other. The task specification is summarized as the following steps:

1. A workpiece arrives at point E1.
2. Robot R1 transfers the workpiece to the exchange area, and at the same time Robot R2 moves to the exchange area.

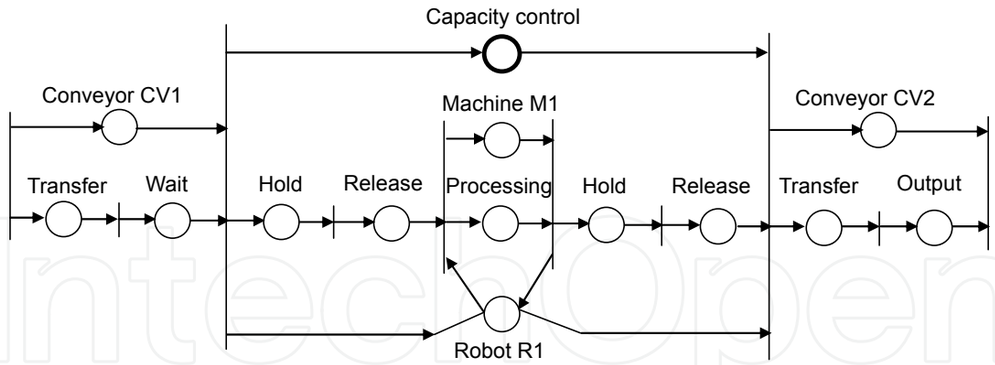


Fig. 14. Net representation of pick-and-place operation with a single robot

3. The workpiece is exchanged from robot R1 to robot R2.
4. Robot R2 changes the workpiece orientation.
5. Robot R2 transfers the workpiece to the exchange area. Robot R1 moves to the exchange area.
6. The workpiece is exchanged from robot R2 to robot R1.
7. Robot R1 transfers the workpiece to point E2.

Following the same procedure of the former example, the subtasks and sets of point data needed to execute the whole task are initially identified. Then they are edited and tested with the net based edition and simulation system. The net representation is written using shared transitions for system coordination as shown in Fig. 15. An experimental view of the cooperative task, passing and exchanging a workpiece, by two robots is shown in Fig. 16.

The detailed procedure of the implemented real-time control based on tasks represented as net models is described as follows. If there is a token in a place corresponding to subtasks, the net based controller sends a message to the respective hardware controllers such as arm, hand, sensor, etc. to execute the defined subtask with certain point data. These parameters (hardware controller code, subtask file code, point data file code) are defined during the net edition procedure. The net based controller was developed with all functions of the edition and simulation to permit correction or modification of the net model on-line. This characteristic is important to facilitate the debugging work. By executing the net model, the

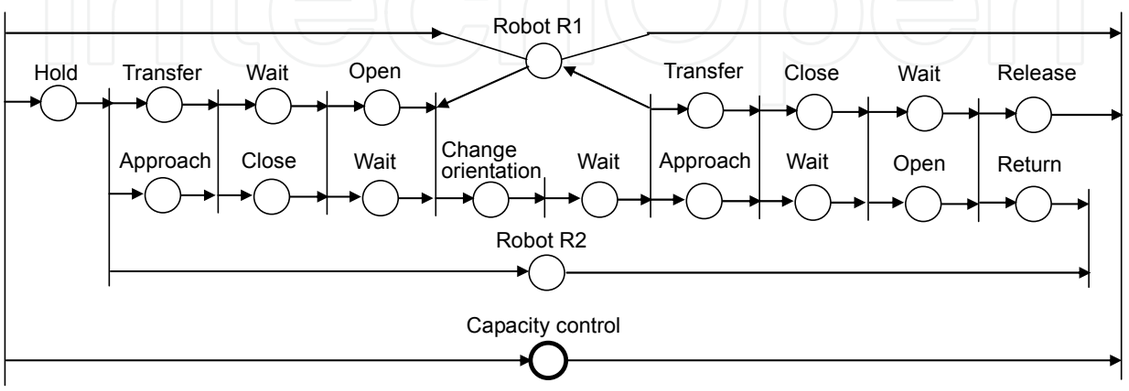


Fig. 15. Detailed net model of cooperative task by two arm robots

Fig. 16. Experiments of cooperative task by two arm robots

developed control system activates the arm, hand, and sensor, etc. and coordinates each individual controller. In making these experiments, it was verified that the implemented system can be used as an effective tool for introducing robots into the manufacturing system. The system can be used to verify and correct control algorithms including robot movements and to evaluate the effectiveness of a robot and other machines in the planning stage.

A multi-computer control architecture composed a system computer and several control computers has been adopted as shown in Fig. 17. The computer control architecture was

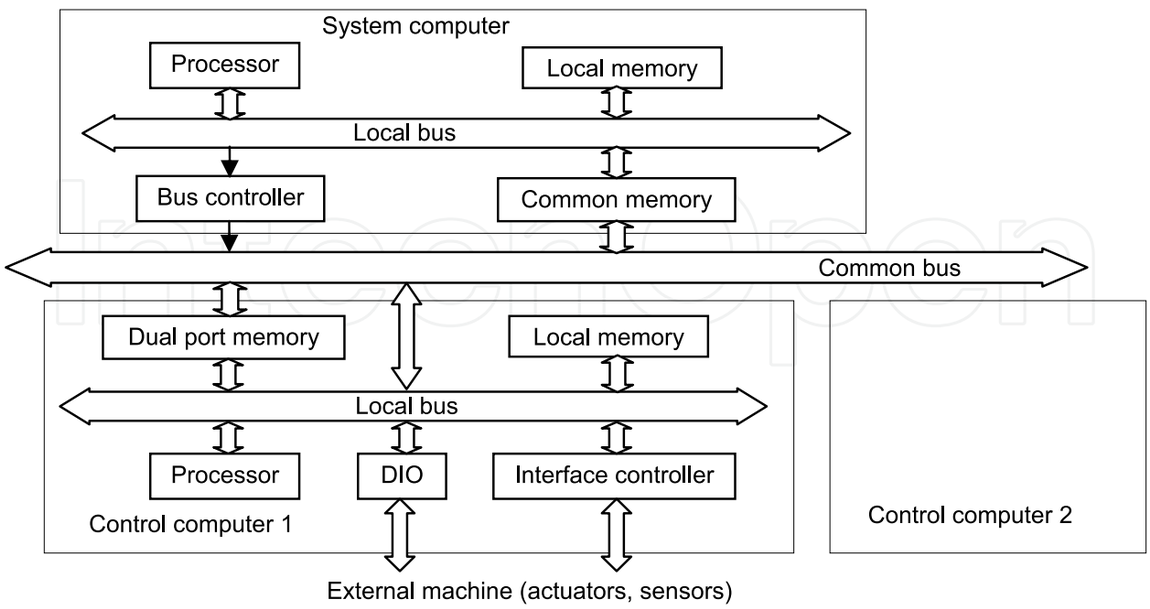


Fig. 17. Multi-computer control architecture composed a system computer and several control computers with dual port memory

developed for the use of distributed autonomous control of independent actuators or machines in compact factory automation systems (Yasuda & Tachibana, 1987). The system controller controls communication between the system controller and control computers through the bus controller based on the master-slave mode. The system computer installs the conceptual net model for system coordination and installs local net models in the control controllers through the common bus. The control computers are equipped with interface circuits to actuators and external sensors for direct machine control and monitoring. Then, in the real-time control, the system computer communicates with each control computer through dual port memory with respect to firing of shared transitions and gate arc signals (Yasuda, 2011). The presented control flow of the net model is successfully executed using output signal arc and permissive gate arc. The net model in the system controller is conceptual for system coordination and not so large. The computation speed of 50 MHz of the general microprocessor is satisfactorily high in comparison with those of controlled devices such as robotic arms, conveyors, machine tools and external sensors.

6. Conclusions

A Petri net based specification and real-time control method for large complex robotic manufacturing systems was introduced as an effective prototyping tool to realize distributed autonomous control systems corresponding to the hardware structure of robotic manufacturing systems. From the design point of view, the use of nets has many advantages in modeling, qualitative analysis, performance evaluation and code generation. The Petri net appears as a key formalism to describe, analyze and implement the distributed autonomous control system for manufacturing systems in future.

7. References

Gentina, J. C. & Corbeel, D. (1987). Coloured Adaptive Structured Petri-Net: A Tool for the Automatic Synthesis of Hierarchical Control of Flexible Manufacturing Systems, *Proceedings of 1987 IEEE International Conference on Robotics and Automation,* pp. 1166-1172

Holt, J. D. & Rodd, M. G. (1994). An Architecture for Real-Time Distributed AI-Based Control Systems. In: *IFAC Distributed Computer Control Systems 1994*, 47-52

Holding, D. J., & Sagoo, J. S. (1992). A Formal Approach to the Software Control of High-Speed Machinery. In: *Transputers in Real-Time Control,* G. W. Irwin & P. J. Fleming (Eds.), 239-282, Research Studies Press, Taunton, Somerset, U.K.

Maletz, M. C. (1983). An Introduction to Multi-robot Control Using Production Systems, *Proceedings of IEEE Workshop on Languages for Automation*, pp. 22-27

Murata, T. (1989). Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, Vol. 77, No. 4, pp. 541-580

Silva, M. (1990). Petri Nets and Flexible Manufacturing. In: *Advances in Petri Nets* G. Rozenberg (Ed.), LNCS 424, 374-417, Springer-Verlag, Berlin, Germany

Taubner, D. (1988). On the Implementation of Petri Nets. In: *Advances in Petri Nets* G. Rozenberg (Ed.), LNCS 340, 419-439, Springer-Verlag, Berlin, Germany

Wang, F. & Saridis, G. N. (1990). A Coordination Theory for Intelligent Machines, *Proceedings of the 11th IFAC World Congress*, pp. 235-240

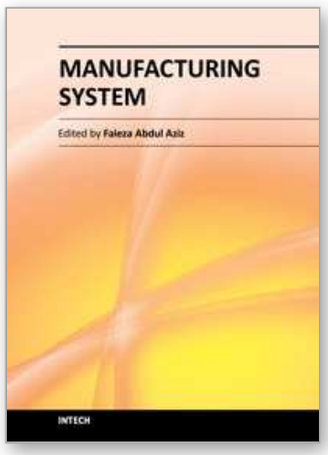
Yasuda, G. (2000). A Multiagent Control Architecture for Multiple, Cooperating Robot Systems, *Proceedings of the International Conference on Production Research Special ICPR-2000,* Paper ID 224, August, 2000

Yasuda, G. (2008). Implementation of Distributed Control Architecture for Industrial Robot Systems Using Petri Nets, *Proceedings of the 39th International Symposium of Robotics*, pp.533-538

Yasuda, G. (2010). Petri Net Based Implementation of Hierarchical and Distributed Control for Discrete Event Robotic Manufacturing Systems, *Proceedings of the 2010 IEEE International Conference on Control Applications, Part of 2010 IEEE Multi-Conference on Systems and Control*, pp.251-256

Yasuda, G. (2011). Design and Implementation of Distributed Control Architecture for Flexible Manufacturing Cells Based on Petri nets, *Proceedings of 12th Asia-Pacific Industrial Engineering & Management Systems Conference*, pp. 852-864

Yasuda, G. & Tachibana, K. (1987). A Multimicroprocessor-Based Distributed Processing System for Advanced Robot Control, *Proceedings of the IXth International Conference on Production Research*, pp.1926-1933



Manufacturing System

Edited by Dr. Faieza Abdul Aziz

ISBN 978-953-51-0530-5
Hard cover, 448 pages
Publisher InTech
Published online 16, May, 2012
Published in print edition May, 2012

This book attempts to bring together selected recent advances, tools, application and new ideas in manufacturing systems. Manufacturing system comprise of equipment, products, people, information, control and support functions for the competitive development to satisfy market needs. It provides a comprehensive collection of papers on the latest fundamental and applied industrial research. The book will be of great interest to those involved in manufacturing engineering, systems and management and those involved in manufacturing research.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen