



## A generic Petri net model for flexible manufacturing systems and its use for FMS control software testing

W. J. ZHANG<sup>†\*</sup>, Q. LI<sup>‡</sup>, Z. M. BI<sup>†</sup> and X. F. ZHA<sup>†</sup>

Developing a flexible manufacturing system (FMS) controller is a complicated task both in the hardware development and software development. To test the control software prior to putting it into use is crucial for the development of a FMS controller. An intelligent testing environment for FMS controllers is under development, the aim of which is to reduce manual work in the testing. To reach this objective the automatic building of a behavioural model of the FMS is demanded. This paper proposes a generic Petri net (GPN) model and approach for the development of control software for FMSs. The principle of this approach is based on checking the control parts of FMSs with the help of temporal relationships between physical operations, and the specification of the FMS controller with GPN. The strategy of GPN modelling is then incorporated with more general problem solving strategies in artificial intelligence. A template is first defined for a GPN model, and then the model for FMS individuals is established in the form of instances of the template. The transitions of the GPN are represented with the proposed general expressions with two syntaxes, 'verb+ noun' and 'verb+ noun+ where', from which the GPN can be formulated. GPN makes it simple to express FMS controls, and procedural language can also be used for information processing. A case study for the testing of FMS controller software is provided to show effectiveness and cost saving over development with conventional methods in which only ordinary Petri net and procedural language are used.

### 1. Introduction

A flexible manufacturing system (FMS) is a production system consisting of several workstations linked by a material handling system capable of enabling jobs to follow diverse routes through the system, monitored and controlled by a network of linked computers, microprocessors and data acquisition devices. Different jobs can be processed simultaneously and with one set-up of hardware system. Such an application flexibility has a great potential to enhance productivity (Archetti *et al.* 1989, Jeng 1995, Altioik 1996). Programmable logic controllers (PLC) have been used for simple control, with computers used in more complicated processes. In developing control software in these cases, programming is carried out using a ladder diagram or a decision table for PLCs or procedural languages such as C for computer control, in either case based on the control specification.

---

Revision received July 1999.

<sup>†</sup>Department of Mechanical Engineering, University of Saskatchewan, Saskatoon SK S7N 5A9, Canada.

<sup>‡</sup>Department of Mechanical Engineering, University of Adelaide, Adelaide, South Australia 5005, Australia.

\*To whom correspondence should be addressed. e-mail: chris\_zhang@engr.usask.ca

Much work has been done on the methods and the tools for specification and implementation of FMS control. These studies show that it is necessary to introduce a checking and testing function into a FMS control system. With a checking and testing function, the control system can react to different events which can occur, and then facilitate controlling and monitoring FMS. In Roux *et al.* (1992) and Nagao *et al.* (1992), studies were carried out to develop a methodology for fault detection and processing based on the utilization of Grafset (Ezpeleta and Martinez 1992) to control and to check the system considering the different states. The interest of this approach consists in using only one tool for the different stages of the system life cycle (design, formal validation, implementation, observation). The idea of checking a system from the model used for the control is very interesting. However, the methods have problems: frequent programming errors, difficulty in verifying whether or not the program meets the specifications, difficulty in understanding control flow and maintaining control software by anybody other than its programmer.

A descriptive model of sequence control by Petri net has become attractive due to its simplicity. Petri nets have a graphical feature which makes understanding the control flow easier. Furthermore, they offer the sequence control functions required to implement FMS. These include synchronization, interlock and concurrence (Jensen 1983, Valette *et al.* 1985, Murata *et al.* 1986). Wainwright and Thethi (1997) presented a review of different methodologies and tools to model a FMS, among which the Petri net was regarded as one of the most important methodologies and tools.

The theory of the Petri net, which has a sound mathematics foundation, was proposed by Petri in Germany in the 1960s (Petersen 1981). The Petri net enables the modelling of synchronic and uncertain behaviours of a system. The graphical representation of a Petri net can express system behaviours, resources and constraints more easily than other methods. Many merits of Petri nets have been identified, compared with other methods, referred to in the literature (Gentina and Corbeel 1987, Kastura *et al.* 1988, Jeng 1993, Lee 1994). Basic Petri nets used in manufacturing systems are defined with two state types (operation places and resource state places) and two classes of transitions (input and output transitions). The place-transition net is represented as a bipartite directed graph. Places are marked by tokens. The maximum number of tokens a place can hold defines its capacity. Places and transitions are connected via directed edges. The number of tokens an edge can transfer defines its weight. A place-transition net is characterized by an initial marking of places and a firing rule. This type of Petri net can be used to control a FMS with only few manufacturing devices and machining parts (Beck 1986, Krogh *et al.* 1988, Chaar 1993). To extend the applicability of Petri net in the FMS with a greater number of devices and jobs, many studies have proposed various Petri net models, such as augmented Petri nets presented by Coolahan and Roussopoulos (1983), and generalized stochastic Petri net by Al-Jaar and Desrochers (1990), time Petri nets by Leveson and Stolzy (1987), predicate transition Petri nets by Giordana and Saitta (1985) and Murata and Zhang (1988), coloured Petri nets by Kamath and Viswanadham (1986) and Martinez *et al.* (1987), structured adaptive coloured Petri nets by Gentina and Corbeel (1987), the modular Petri net synthesis method by Jeng (1995), and the extended Petri net by Crockett *et al.* (1987), Zhou (1989) and Sodhi *et al.* (1994).

While there have been some programming tools developed for FMS control software in Petri nets, they are not sufficiently capable of FMS control involving complicated information processing. In this paper, a generic Petri net model is proposed for the development of FMS control software, and an approach is developed to implement operations checking and control software testing in a FMS. It consists of two phases: analysis of the system and its functioning, and the design of the checking and testing system using data from the analysis. A generic Petri net of FMS is first described as a kind of ‘template’, and the model of a particular FMS is then established as instances of the template. By doing so, the complexity of conventional Petri nets can be greatly reduced. It is then possible to make a computer program generate a Petri net for a given FMS, which is envisaged for the FMS control software testing. The key ideas leading to our objective include: (i) the resources in a FMS are abstracted according to their activity characteristics in the FMS; (ii) the elements in sets of the places and the transitions are described via the general problem solving strategy in AI; (iii) the topologies among the places and transitions are obtained based on two proposed general expressions with the syntaxes ‘Verb + Noun’ and ‘Verb + Noun + Where’ for describing material- and tool-flows in a FMS.

## 2. Description of a FMS

A FMS typically is composed of: (i) several manufacturing machines, such as CNC machining centres, CNC, measuring machines, and washing machines; (ii) material transport and handling equipment, such as an automatic guided vehicle together with a material loading and unloading station, central material buffer, and local material buffer dedicated to an individual manufacturing machine to carry out efficient material-flow tasks within the system; and (iii) tool transport and exchange equipment such as a movable robot together with a tool loading and unloading station, central tool bases, and local tool magazines for each manufacturing machine to carry out efficient tool-flow tasks within the system.

The development of a GPN calls for a more comprehensive classification of a FMS. Service resources in a FMS are classified according to dynamic characteristics in the running procedure; the objects to be served include the materials and the machining tools, such as the workstation (WS), buffer (BF), conveying device (CD), and auxiliary tool (AT). The definitions and examples of these service resources are shown in table 1. For example, WSs are defined as all the devices which can change the process information of a material in the material-flow, or which can change the number of the tools in the tool flow. They may include a loading/unloading station, a cleaner, a measuring machine, an inspection machine in the material flow; or may include a tool loading/unloading station, and a tool inspection station in the tool flow. Some other resources are the local resources related to WSs, which are shown in table 2. As a result, WSs can be further classified into eight types according to the inclusion of local resources such as local buffer, loading tool, and machine tool.

Figure 1 shows a typical structure configuration of a FMS, which will be used as a testing case in this research. The service resources of this case FMS and their functions are described in table 3. All eight sets of the fixtures and pallets used are interchangeable, and they are available at reach of the loading robots at the start. The operation information of the jobs and resources will be discussed in §5 below.

Abbreviation	Definition	Examples
WS	All the devices which can change the process information of a material in the material-flow, or which can change the number of the tools in the tool flow	<p><i>In the material flow:</i> Loading/unloading station, cleaner, measuring machine, inspection machine</p> <p><i>In the tool flow:</i> Tool loading/unloading station, tool inspection station</p>
BF	All the devices whose positions can be occupied by materials (pallets) but can't change the process information, or whose position can be occupied by tools but can't change the number of the tools	<p><i>In the material flow:</i> Local buffer, centre buffer</p>
CD	All the devices which can change the physical position of a serviced object	<p><i>In the tool flow:</i> Centre tool base, local tool magazine AGV, tool exchange, transport</p>
CT	All the resources accompanying an object serviced in the whole system	Fixture tool, pallets and tool shelf
AT	All the auxiliary tools needed in the service process of a material	<p>Loading manipulator, man inspecting or cleaning tools</p> <p>Machining tools in the material flow</p>

Table 1. Classification of service resources in a FMS.

Type	1	2	3	4	5	6	7	8
Local buffer	×	×	×	×	✓	✓	✓	✓
Loading tool	×	×	✓	✓	×	×	✓	✓
Machine tool	×	✓	×	✓	×	✓	×	✓

Table 2. Type classification of WSs.

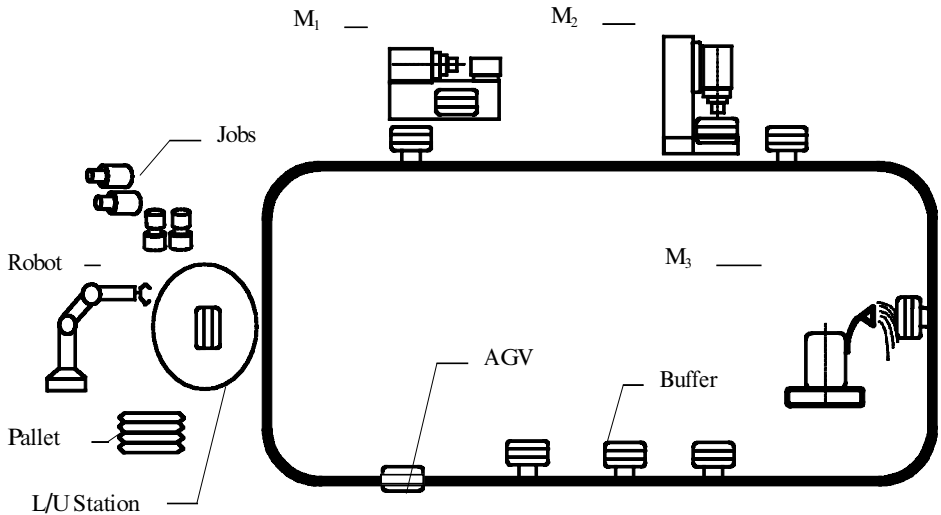


Figure 1. Configuration of the FMS tested.

Device type	Device name	Resource type	Position No.	Function
1	L/U	WS	1	Place for loading/unloading a job
2	M <sub>1</sub>	WS	1	Cutting
			2	Buffer position
3	M <sub>2</sub>	WS	1	Boring
			2	Buffer position
4	M <sub>3</sub>	WS	1	Cleaning
5	AGV	CD	1	Conveying job
6	Buffer	BF	1	Buffer position
			2	Buffer position
7	Robot	CT	—	Loading or unloading a job

Table 3. The configuration of the FMS tested.

In practice, events in a FMS are the transitions happening on the system resources. An operation of a FMS may be composed of a series of such transitions. Consequently, a set of transitions corresponding to various resources should be known first so as to control and monitor a FMS.

### 3. Generic Petri net model

Petri nets generally consist of places ( $P$ ) and transitions ( $T$ ), which are linked to each other by arcs. They can be described as bipartite directed graphs whose nodes are a set of places associated with a set of transitions. Their activities are based on a

vision of tokens moving around an abstract network. Tokens are conceptual entities which appear as small solid dots and model the objects that move in a real network. Pictorially, Petri nets consist of four primitive elements: tokens, places, transitions, and arcs (Murata 1989).

From a formal view point, an ordinary Petri net is defined as a 6-tuple  $PN = (P, T, I, O, K, M_0)$ , where,  $P$  is a set of places,  $T$  is a set of transitions,  $P \cap T = \emptyset$ ,  $P \cup T \neq \emptyset$ ,  $I$  is the input functions of the transitions,  $O$  is the output functions of the transitions,  $K$  is the capacities of points, and  $M_0$  is the initial marks of the places.

The major difficulty with ordinary Petri nets is that industrial applications are likely to result in large systems consisting of many places and transitions. Other shortcomings include structural inflexibility, and the inability to identify individual tokens. In order to use Petri nets for modelling complex systems, several extensions to standard or ordinary Petri nets have been proposed with consideration of time, uncertainty, and knowledge information involved (Jensen 198, Giordana and Saitta, 1985, Leveson and Stolzy 1987, Zha *et al.* 1998). In this paper, a new generic Petri net model is proposed, which incorporates the ordinary place/transition Petri net into general problem description scheme in artificial intelligence.

From artificial intelligence theory, a problem can generally be expressed by a 3-element  $(X, f, \mathfrak{R})$  (Zhang 1990), where,  $X$  is a set of the variables;  $f(\cdot)$  are the attributes of variables, expressed by the functions  $f: X \rightarrow Y$ , where  $Y$  is a multi-dimensional space;  $\mathfrak{R}$  denotes the topologies on the set of the variables. From the Petri net notation above, the correspondences between the Petri net and the general description of a problem can be observed and described as follows:

$$X \Leftrightarrow \{P, T\}, \mathfrak{R} \Leftrightarrow \{I, O\}, f \Leftrightarrow \{K, M_0\} \quad (1)$$

Therefore, Petri net modelling can be considered to be equivalent to general problem solving strategies in AI. By a generic Petri net modelling scheme, we mean that a Petri net model of a problem is first described as a kind of 'template', and the model of the particular sub-problems is then established as instances of the template. Since the proposed generic Petri net is generated by incorporating a Petri net into a general problem description scheme in AI, the existing AI-based problem solving strategies such as search, reasoning and expert systems (Zha *et al.* 1998), are potentially applicable to generic Petri net modelling and analysis.

#### 4. FMS modelling and analysis with generic Petri nets

It is known that the dynamic behaviours of FMS are modelled by a Petri net. In what follows of this section, the variables, attributes and topologies in the context of Petri nets of FMSs will be elaborated. As such, the so-called generic Petri net model of a FMS is developed. The analysis of the generic Petri net model of a FMS will also be discussed.

##### 4.1. Variables in controlling and monitoring a FMS

The variables in a FMS include the elements both in a set of places and a set of transitions. For simplicity, in the following the elements in the material-flow only will be discussed; however, the term job-flow will be used interchangeably with material-flow as long as no confusion is present.



$$\begin{aligned} \text{job}'_{i,j} &= \langle \text{job}_{i,j}, f_1^i, p_1^i \rangle (f_1^i \in F_1^i, p_1^i \in P_1^i) \\ J'_i &= \{ \text{job}'_{i,j} | j' \in [0, g_i] \} \\ J' &= J'_1 \cup J'_2 \dots \cup J'_n \end{aligned}$$

2-dimensional place: the state that a job stays on a service resource

$$\begin{aligned} \wp_{21} &= \{ \langle \text{job}'_{i,j+1}, b_{i,j+1} \rangle | \text{job}'_{i,j+1} \in J', b \in (B_1 \cup B_2) \} \\ \wp_{221} &= \{ \langle \text{job}'_{i,j+1}, mb \rangle | \text{job}'_{i,j+1} \in J'_i, mb \in (MB_1^{i,j+1} \cup MB_2^{i,j+1}) \} \\ \wp_{222} &= \{ \langle \text{job}'_{i,j+1}, m \rangle | \text{job}'_{i,j+1} \in J'_i, m \in (M_1^{i,j+1} \cup M_2^{i,j+1}) \} \\ \wp_{223} &= \{ \langle \text{job}'_{i,j+2}, m \rangle | \text{job}'_{i,j+2} \in J'_i, m \in (M_1^{i,j+1} \cup M_2^{i,j+1}) \} \\ \wp_{224} &= \{ \langle \text{job}'_{i,j+2}, mb \rangle | \text{job}'_{i,j+2} \in J'_i, mb \in (MB_1^{i,j+1} \cup MB_2^{i,j+1}) \} \\ \wp_{23} &= \{ \langle \text{job}'_{i,j+1}, t \rangle | \text{job}'_{i,j+1} \in J', t \in (T_1 \cup T_2) \} \\ \wp_{24} &= \{ \langle \text{job}'_{i,j+1}, b \rangle | \text{job}'_{i,j+1} \in J', b \in (B_1 \cup B_2) \}. \end{aligned}$$

3-dimensional place: the state that a job is being transported from one service to another

$$\begin{aligned} \wp_{31} &= \{ \langle \text{job}'_{i,j+1}, m, t \rangle | \text{job}'_{i,j+1} \in J', m \in (M_1^{i,j} \cup M_2^{i,j}), t \in (T_1 \cup T_2) \} \\ \wp_{32} &= \{ \langle \text{job}'_{i,j+1}, t, b \rangle | \text{job}'_{i,j+1} \in J', t \in (T_1 \cup T_2), b \in (B_1 \cup B_2) \} \\ \wp_{33} &= \{ \langle \text{job}'_{i,j+1}, b, t \rangle | \text{job}'_{i,j+1} \in J', b \in (B_1 \cup B_2), t \in (T_1 \cup T_2) \} \\ \wp_{34} &= \{ \langle \text{job}'_{i,j+1}, t, m \rangle | \text{job}'_{i,j+1} \in J', t \in (T_1 \cup T_2), m \in (M_1^{i,j} \cup M_2^{i,j+1}) \}. \end{aligned}$$

In addition, there is another kind of 3-dimensional state—the state on a WS, which only involves local resources of the WS and can be represented as:

$$\begin{aligned} \wp_{351} &= \left\{ \langle \text{job}'_{i,j+1}, mb, mlt \rangle | \text{job}'_{i,j+1} \in J', mb \in (MB_1^{i,j+1} \cup M_2^{i,j+}), \right. \\ &\quad \left. mlt \in (MLT_1^{mb} \cup MLT_2^{mb}) \right\} \\ \wp_{352} &= \left\{ \langle \text{job}'_{i,j+1}, m, mmt \rangle | \text{job}'_{i,j+1} \in J', m \in (M_1^{i,j+1} \cup M_2^{i,j+1}), \right. \\ &\quad \left. mmt \in (T_1^{i,j+1,m} \cup T_2^{i,j+1,mb}) \right\} \\ \wp_{353} &= \left\{ \langle \text{job}'_{i,j+2}, m, mlt \rangle | \text{job}'_{i,j+2} \in J', m \in (M_1^{i,j+1} \cup M_2^{i,j+1}), \right. \\ &\quad \left. mlt \in (MLT_1^{mb} \cup MLT_2^{mb}) \right\}. \end{aligned}$$

The places over 3-dimensional state are used only when some high-level beyond the shop floor level strategies are considered. They are thus not discussed here.

#### 4.1.2. Transitions in the job-flow of a FMS

As discussed before, a transition is an event in a FMS where one state of the system is changed to another. As far as resources are concerned, in the client aspect a job is transformed from one position to another; in the server aspect, the states of service resources must be changed as well. If the case that a service resource is broken down within its idle state is neglected, the state transitions of a service resource may include the following cases: (i) a service resource becomes busy after it is occupied,

(ii) a service resource becomes idle after it is released, and (iii) a service resource is broken down when it is in use.

In the operation of a FMS, when a control command is issued from the control software, reaction of the command results in an ‘active’ transition of the corresponding resources. After the command is carried out, a ‘passive’ transition happens. The passive transition becomes new prerequisites for a further controlling action. Active transitions are the aggregation of the places with lower dimensions into higher dimensions, whereas the passive transition are the decomposition of the place with higher dimensions. In what follows, these transitions will be presented.

*1-dimensional place*  $\oplus$  *2-dimensional place*  $\Leftrightarrow$  *3-dimensional place*. Where ‘ $\rightarrow$ ’ stands for an active transition, and ‘ $\leftarrow$ ’ a passive transition. These interpretations are also applicable to later discussions.

The above statement indicates the following fact. When a job is assigned to a FMS, some service resources are occupied. The following transitions can happen in the system.

*Active transitions:*

$$\begin{aligned}\mathfrak{T}_{11} &= \{t_{11} | \langle \text{job}'_{i,j+1}, m_1 \rangle \oplus t_0 \rightarrow \langle \text{job}'_{i,j+1}, m_1, t_1 \rangle \quad (t_0 \in T_0, \quad t_1 \in T_1, \quad m_1 \in M_1^{i,1})\} \\ \mathfrak{T}_{12} &= \{t_{12} | \langle \text{job}'_{i,j+1}, t_1 \rangle \oplus m_0 \rightarrow \langle \text{job}'_{i,j+1}, t_1, m_1 \rangle \quad (t_1 \in T_1, \quad m_0 \in M_0^{i,j}, \quad m_1 \in M_1^{i,j})\} \\ \mathfrak{T}_{13} &= \{t_{13} | \langle \text{job}'_{i,j+1}, t_1 \rangle \oplus b_0 \rightarrow \langle \text{job}'_{i,j+1}, t_1, b_1 \rangle \quad (t_1 \in T_1, \quad b_0 \in B_0, \quad b_1 \in B_1)\} \\ \mathfrak{T}_{14} &= \{t_{14} | \langle \text{job}'_{i,j+1}, b_1 \rangle \oplus t_0 \rightarrow \langle \text{job}'_{i,j+1}, b_1, t_1 \rangle \quad (t_0 \in T_0, \quad t_1 \in T_1, \quad b_1 \in B_1)\}.\end{aligned}$$

*Passive transitions:*

$$\begin{aligned}\mathfrak{T}'_{11} &= \{t'_{11} | \langle \text{job}'_{i,j+1}, m_1, t_1 \rangle \rightarrow \langle \text{job}'_{i,j+1}, m_1 \rangle \oplus t_0 \quad (t_0 \in T_0, \quad t_1 \in T_1, \quad m_1 \in M_1^{i,j})\} \\ \mathfrak{T}'_{12} &= \{t'_{12} | \langle \text{job}'_{i,j+1}, t_1, m_1 \rangle \rightarrow \langle \text{job}'_{i,j+1}, t_1 \rangle \oplus m_0 \quad (t_1 \in T_1, \quad m_0 \in M_0^{i,j}, \quad m_1 \in M_1^{i,j})\} \\ \mathfrak{T}'_{13} &= \{t'_{13} | \langle \text{job}'_{i,j+1}, t_1, b_1 \rangle \rightarrow \langle \text{job}'_{i,j+1}, t_1 \rangle \oplus b_0 \quad (t_1 \in T_1, \quad b_0 \in B_0, \quad b_1 \in B_1)\} \\ \mathfrak{T}'_{14} &= \{t'_{14} | \langle \text{job}'_{i,j+1}, b_1, t_1 \rangle \rightarrow \langle \text{job}'_{i,j+1}, b_1 \rangle \oplus t_0 \quad (t_0 \in T_0, \quad t_1 \in T_1, \quad b_1 \in B_1)\}\end{aligned}$$

*1-dimensional place*  $\oplus$  *1-dimensional place*  $\Leftrightarrow$  *2-dimensional place*. In addition, when a job is assigned to or released from the system, the following transitions take place.

*Active transitions* (a job is assigned to the system):

$$\mathfrak{T}_0 = \{t_0 | \text{job}_{i,0} \oplus f_0 \oplus p_0 \oplus m_0 \rightarrow \langle \text{job}'_{i,0}, m_1 \rangle\}$$

where

$$\begin{aligned}\text{job}'_{i,0} &= \langle \text{job}_{1,0}, f_1, p_1 \rangle, \\ f_1 &\in F_1^i, p_1 \in P_1^i, f_0 \in F_0^i, p_0 \in P_0^i, m_0 \in M_0^{i,0}, m_1 \in M_1^{i,0}.\end{aligned}$$

*Passive transitions* (a job is released from the system):

$$\mathfrak{T}'_0 = \{t'_0 | \langle \text{job}'_{i,g_i}, m_1 \rangle \rightarrow \text{job}_{i,g_i} \oplus f_0 \oplus p_0 \oplus m_0\}$$

where

$$\begin{aligned}\text{job}'_{i,g_i} &= \langle \text{job}_{i,g_i}, f_1, p_1 \rangle, \\ f_1 &\in F_1^i, p_1 \in P_1^i, f_0 \in F_0^i, p_0 \in P_0^i, m_0 \in M_0^{i,g_i}, m_1 \in M_1^{i,g_i}.\end{aligned}$$

Especially, when a job is receiving an operation the following transitions may happen:

$$\begin{aligned}
\mathfrak{T}_{21} &= \left\{ t_{21} \left| \begin{array}{l} \langle \text{job}'_{i,j+1}, mb_1 \rangle \oplus mlt_0 \rightarrow \langle \text{job}'_{i,j+1}, mb_1, mlt_1 \rangle, \\ (\text{job}'_{i,j+1} \in J', mb_1 \in MB_1^{i,j+1}, mlt \in MLT^{mb}) \end{array} \right. \right\} \\
\mathfrak{T}'_{21} &= \left\{ t'_{21} \left| \begin{array}{l} \langle \text{job}'_{i,j+1}, mb_1, mlt_1 \rangle \otimes m_0 \rightarrow \langle \text{job}'_{i,j+1}, m_1 \rangle \oplus mlt_0 \oplus mb_0, \\ (\text{job}'_{i,j+1} \in J', m_1 \in M_1^{i,j+1}, mlt \in MLT^{mb}) \end{array} \right. \right\} \\
\mathfrak{T}_{22} &= \left\{ t_{21} \left| \begin{array}{l} \langle \text{job}'_{i,j+1}, m_1 \rangle \oplus mmt_0 \rightarrow \langle \text{job}'_{i,j+1}, m_1, mmt_1 \rangle, \\ (\text{job}'_{i,j+1} \in J', m_1 \in M_1^{i,j+1}, mmt \in T^{i,j+1,m}) \end{array} \right. \right\} \\
\mathfrak{T}'_{22} &= \left\{ t'_{22} \left| \begin{array}{l} \langle \text{job}'_{i,j+1}, m_1, mmt_1 \rangle \rightarrow \langle \text{job}'_{i,j+1}, m_1 \rangle \oplus mmt_0, \\ (\text{job}'_{i,j+1} \in J', m_1 \in M_1^{i,j+1}, mmt \in T^{i,j,m}) \end{array} \right. \right\} \\
\mathfrak{T}_{23} &= \left\{ t_{23} \left| \begin{array}{l} \langle \text{job}'_{i,j+1}, m_1 \rangle \oplus mlt_0 \rightarrow \langle \text{job}'_{i,j+1}, m_1, mlt_1 \rangle, \\ (\text{job}'_{i,j+1} \in J', m_1 \in M_1^{i,j+1}, mlt \in MLT^{mb}) \end{array} \right. \right\} \\
\mathfrak{T}'_{23} &= \left\{ t'_{23} \left| \begin{array}{l} \langle \text{job}'_{i,j+1}, m_1, mlt_1 \rangle mb_0 \rightarrow \langle \text{job}'_{i,j+1}, mb_1 \rangle \oplus mlt_0 \oplus m_0, \\ (\text{job}'_{i,j+1} \in J', m_1 \in M_1^{i,j+1}, mlt \in MLT^{mb}) \end{array} \right. \right\}.
\end{aligned}$$

*2-dimensional place*  $\Leftrightarrow$  *2-dimensional place*. If a service resource is broken down while it is in use, the transition with unchangeable dimensions takes place. In particular, a breakdown corresponds to an active transition, whereas the recovery of the breakdown a passive transition.

$$\begin{aligned}
\mathfrak{T}_{311} &= \{t_{311} | \langle \text{job}'_{i,j+1}, mb_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+1}, mb_1 \rangle (\text{job}'_{i,j+1} \in J', mb \in MB^{i,j+1})\} \\
\mathfrak{T}_{312} &= \{t_{312} | \langle \text{job}'_{i,j+2}, mb_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+2}, mb_2 \rangle (\text{job}'_{i,j+2} \in J', mb \in MB^{i,j+1})\} \\
\mathfrak{T}_{32} &= \{t_{32} | \langle \text{job}'_{i,j+1}, b_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+1}, b_2 \rangle (\text{job}'_{i,j} \in J', b \in B)\} \\
\mathfrak{T}_{311} &= \{t_{33} | \langle \text{job}'_{i,j+1}, t_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+1}, t_2 \rangle (\text{job}'_{i,j} \in J', t \in T)\}.
\end{aligned}$$

*3-dimensional place*  $\Leftrightarrow$  *3-dimensional place*. If a job occupies two service resources, and one of them broken down, the corresponding transitions take place and can be expressed as:

$$\begin{aligned}
\mathfrak{T}_{41} &= \{t_{41} | \langle \text{job}'_{i,j+1}, m_1, t_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+1}, m_2, t_1 \rangle (\text{job}'_{i,j+1} \in J', m \in M^{i,j}, t_1 \in T_1)\} \\
\mathfrak{T}_{42} &= \{t_{42} | \langle \text{job}'_{i,j+1}, m_1, t_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+1}, m_1, t_2 \rangle (\text{job}'_{i,j+1} \in J', m_1 \in M_1^{i,j}, t \in T)\} \\
\mathfrak{T}_{43} &= \{t_{43} | \langle \text{job}'_{i,j+1}, t_1, m_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+1}, t_1, m_2 \rangle (\text{job}'_{i,j+1} \in J', t_1 \in T_1, m \in M^{i,j+1})\} \\
\mathfrak{T}_{44} &= \{t_{44} | \langle \text{job}'_{i,j+1}, t_1, m_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+1}, t_2, m_1 \rangle (\text{job}'_{i,j+1} \in J', t \in T, m_1 \in M_1^{i,j})\} \\
\mathfrak{T}_{45} &= \{t_{45} | \langle \text{job}'_{i,j+1}, b_1, t_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+1}, b_2, t_1 \rangle (\text{job}'_{i,j+1} \in J', b \in B, t_1 \in T_1)\} \\
\mathfrak{T}_{46} &= \{t_{46} | \langle \text{job}'_{i,j+1}, b_1, t_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+1}, b_1, t_2 \rangle (\text{job}'_{i,j+1} \in J', b_1 \in B_1, t \in T)\} \\
\mathfrak{T}_{47} &= \{t_{47} | \langle \text{job}'_{i,j+1}, t_1, b_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+1}, t_2, b_1 \rangle (\text{job}'_{i,j+1} \in J', t \in T, b_1 \in B_1)\} \\
\mathfrak{T}_{48} &= \{t_{48} | \langle \text{job}'_{i,j+1}, t_1, b_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+1}, t_1, b_2 \rangle (\text{job}'_{i,j+1} \in J', t_1 \in T_1, b \in B)\}.
\end{aligned}$$

In addition, when a breakdown happens on a WS, the corresponding transitions are

$$\begin{aligned} \mathfrak{T}_{491} &= \left\{ t_{491} | \langle \text{job}'_{i,j+1}, mb_1, mlt_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+1}, mb_1, mlt_2 \rangle \right. \\ &\quad \left. (\text{job}'_{i,j+1} \in J', mb \in MB^{i,j}, mlt_1 \in MLT^{mb}) \right\} \\ \mathfrak{T}_{492} &= \left\{ t_{492} | \langle \text{job}'_{i,j+1}, m_1, mmt_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+1}, m_2, mmt_1 \rangle \right\} \\ &\quad \left. (\text{job}'_{i,j+1} \in J', m \in M_1^{i,j+1}, mmt \in T^{i,j+1,m}) \right\} \\ \mathfrak{T}_{493} &= \left\{ t_{493} | \langle \text{job}'_{i,j+1}, m_1, mmt_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+1}, m_1, mmt_2 \rangle \right\} \\ &\quad \left. (\text{job}'_{i,j+1} \in J', m \in M_1^{i,j+1}, mmt \in T^{i,j+1,m}) \right\} \\ \mathfrak{T}_{494} &= \left\{ t_{494} | \langle \text{job}'_{i,j+1}, mb_1, mlt_1 \rangle \Leftrightarrow \langle \text{job}'_{i,j+1}, mb_1, mlt_2 \rangle \right\} \\ &\quad \left. (\text{job}'_{i,j+1} \in J', mb \in MB_1^{i,j+1}, mlt_1 \in MLT_1^{mb}) \right\}. \end{aligned}$$

It is noted that the transitions described above within the scope of the job-flow cover all possible transitions in a FMS.

#### 4.2. Attributes of the variables

In the Petri net model of a FMS, useful attributes of the variables include marks of the places and transitions, capacities of the places, input and output functions of the transitions, which are further clarified below:

$K(p), p \in P$	Capacities of the place;
$M(p), p \in P$	Number of the tokens of the places;
$M(t), t \in T$	Number of times when the transition is fired;
$I(p, t), p \in P, t \in T$	Input function of a transition;
$O(p, t), p \in P, t \in T$	Output function of a transition.

#### 4.3. Topology of the variables in the job-flow

The topology of the variables is a set of the relations among the variables. In the Petri net model of a FMS, the topological relations are described by the general expressions with the syntaxes ‘Verb + Noun’ (VN for short), ‘Noun + Verb + Where’ (VNW). The term Noun will be replaced by things like materials, tools, etc., and the term Verb by things like loading, unloading, transporting, etc. Mathematically, they can be expressed by  $I = P \times T$ ,  $O = T \times P$ . There are a number of basic events which happen in a FMS; in particular in the job-flow they can be shown below:

- a job is loaded to or unloaded from the system (VNW);
- a job is transported from one workstation to another (VNW);
- a job is transported from a buffer to a workstation (VNW);
- a job is transported from a workstation to a buffer (VNW);
- a machining process is started or finished (VN).

The NVW relations among the basic events are shown in figure 2. The figure shows the following generic behaviour of a FMS. The transformation of a raw job in a FMS begins with loading the raw material at a L/U station. It then receives machining operations step by step at the workstations determined. Suppose the next workstation is occupied by another job. A FMS controller may transform it to a temporal position—available at the centre buffer, waiting until the next workstation is available. Such a loop of the transformation will continue until the job has

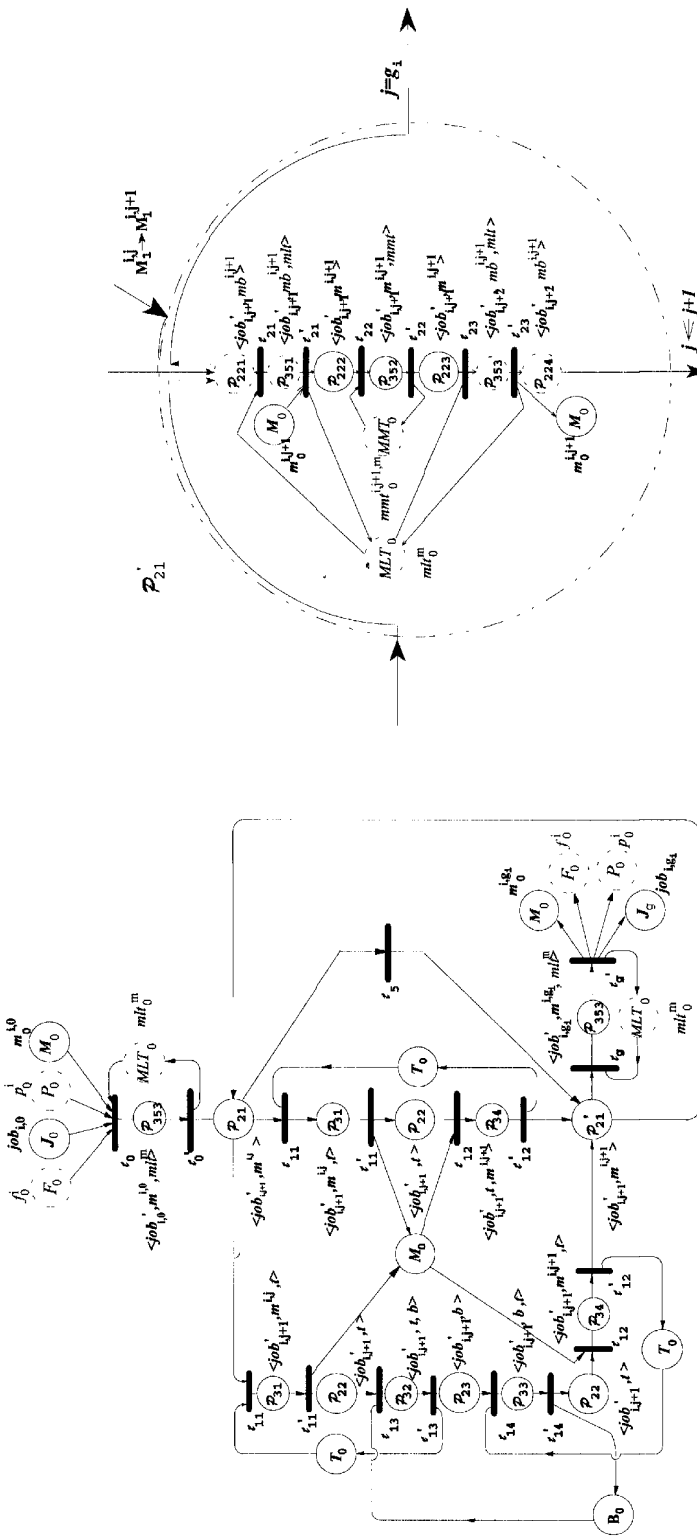


Figure 2. Topological relations in the job-flow.

gone through all predefined machining operations. The job will be ended when the machined raw materials (parts or products) are unloaded at a L/U station.

In practice, besides the normal topological relations, the abnormal case (i.e. breakdown of a service resource) should be considered as well. Suppose that a breakdown happens when the resource is busy and it only involves one of the resources. The corresponding topological relations are shown in figure 3, where the breakdown resource is recovered or replaced. Other cases can be derived in the same way.

## 5. Generic Petri net system for a FMS

As illustrated above, the generic Petri net can be used for modelling a FMS. However, this method needs much more reasoning experience or intervention and has such an inconvenience that the user has to verify many feasible states. It is too complex and time-consuming to be coped with manually, especially for more complex system (with more resources). To design and verify FMS control software automatically, a computer programming system has been developed for generic Petri net modelling of the FMS described above. The key implementation issues will be presented in this section.

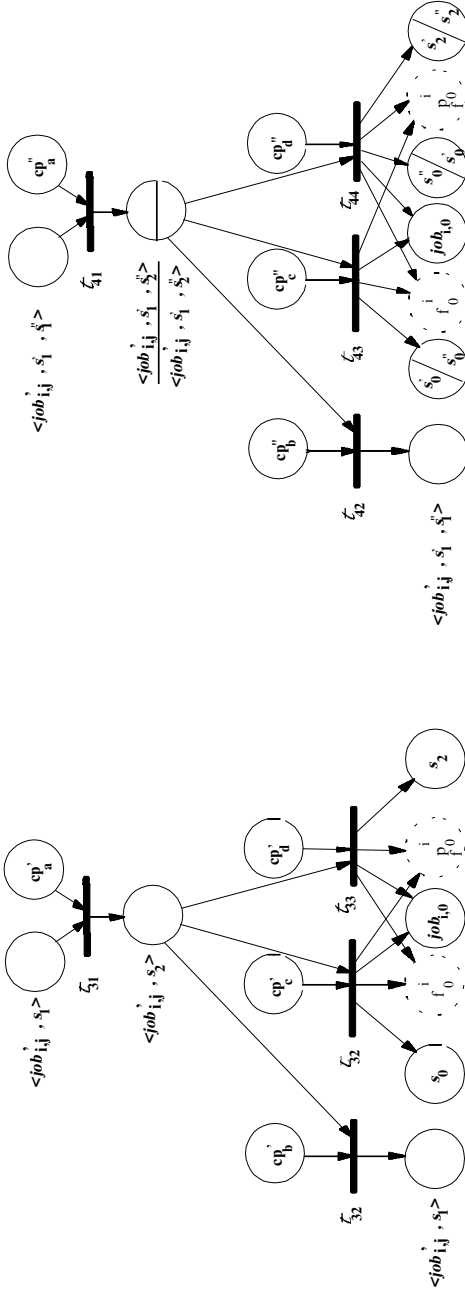
### 5.1. System overview

The GPN FMS system automatically generates control programs given the necessary control specifications described with Petri nets. It has an editor to input and edit control specifications expressed by the Petri net, a simulator to verify edited specifications, a generator to convert the net to C source programs for a controller, and a reporter to print control specifications. The basic events are predefined, followed by the predefinition of the places and transitions. When the structure of a particular FMS is known to the program system, the templates of places and transitions (i.e. the generic Petri net model) are instantiated. In this environment, the configuration of a FMS can be built easily through a user-friendly interface. The FMS controller is installed on a host computer; both the command information (issued from the FMS controller) and the feedback information (produced by the computer in the simulating environment) are received by the system. By connecting the GPN system to the FMS controller, its status can be monitored in real-time. The GPN FMS system has the following features:

- (i) graphical representation of complicated control specifications using a high-level Petri net;
- (ii) expression of transition firing and token flow conditions using VN, VNW, and If-Then rules;
- (iii) combination of Petri net with a procedural language; and
- (iv) full support of all development stages including general design, detailed design, programming and testing.

### 5.2. System implementation

The GPN FMS system has been implemented with the computer language C and is defined as several types of data structure. The instances of these data structures represent the model of a particular FMS. Suppose that a job is introduced to a system (i.e. a basic transition) and the FMS controller is requested to control the



(a) A Job Occupying One Resource

Where  $s \in M^{i,j} \sqcup MB^{i,j} \sqcup B \sqcup T$

$cp_s, cp'_s, cp_s, cp'_s$  are the places of control information

(b) A Job Occupying Two Resources

Where  $s, s'' \in M^{i,j} \cup MB^{i,j} \cup B \cup T, s \neq s''$

$cp_s, cp'_s, cp_s, cp'_s$  are the places of control information

Figure 3. Topological relations of the variables in the case of breakdown.

FMS to select a job from a set of candidate jobs in a queue based on a defined priority rule. The types of data structures as well as instances can be listed below.

### 5.2.1. 1-dimensional place: the state of an idle resource

The states of the idle fixtures and pallets are 1-dimensional places, and their structures are defined as:

```

Struct
{
Int      F_T;           /* the fixture type */
Float    Life_Time;    /* the life time of the fixture */
Float    Used_Time;    /* the used time of the fixture */
Int      Status;       /* the status of the fixture;
                       (0, empty; 1, busy; 2, broken down)*/
Float    Start_Time;   /* the beginning time of present status */
}
F0[];
Struct
{
Int      P_T;           /* the pallet type */
Float    Life_Time;    /* the life time of the pallet */
Float    Used_Time;    /* the used time of the pallet */
Int      Status;       /* the status of the pallet;
                       (0, empty; 1, busy; 2, broken down)*/
Float    Start_Time;   /* the beginning time of present status */
}
P0[];

```

The states of the idle devices and auxiliaries are also 1-dimensional places, and their structures are defined as:

```

Struct
{
Int      Device_Class; /* the device class which is coded based on table 2.
                       */
Int      Device_Type;  /* the device type by which a job is processed */
char     Device_Name[]; /* the device name */
int      Position_Number; /* the number of position where a job is processed */
float    Used_Time;    /* the used time of the device */
int      Status;       /* the status of the device;
                       (0, empty; 1, busy; 2, broken down)*/
float    Start_Time;   /* the beginning time of present status */
}
M0[];
struct
{
int      Number_Device; /* the number of devices served */
int      Device_Type[]; /* the set of devices served */
char     Tool_Name[];   /* the name of the loading tool */
int      Life_Time;     /* the life-time of the loading tool */
}

```

```

float  Used_Time;      /* the used time of the loading tool */
int    Status;        /* the status of the loading tool;
                      (0, empty; 1, busy; 2, broken down)*/
float  Start_Time;    /* the beginning time of present status */
}
MLT0
[];

```

In addition, for the raw jobs in the system, the corresponding structure is defined as:

```

struct
{
struct  Job_Type;      /* contains the process information of a job */
int     Remained_Proc; /* the number of remained processes */
}
J0[];
  struct
  {
char    Due_Date[];   /* the delivery date of the job */
int     Priority;      /* the priority of the job */
int     Fixture_Type; /* the fixture type used */
int     Pallet_Type;  /* the pallet type used */
int     T_N_P;        /* total number of processes */
int     N_M[];        /* the number of the machine alternatives */
int     Machine[][];  /* the alternative machine types used in the i-th
                      process */
int     NC_No[][];    /* the corresponding NC No. used in the i-th
                      process */
float   Process_Time[][]; /* the corresponding machining time of the i-th
                      process */
int     T_N[][];      /* the number of tools used in the i-th process */
int     M_Tool[][][]; /* the set of corresponding machining tools
                      used in the i-th process */
}
Jtype[];

```

### 5.2.2. 2-dimensional place: the state that a job stays on a service resource

To consider basic transitions, only one 2-dimensional place is concerned, and its structure is defined as:

```

struct
{
int     Job_No;        /* the job no. which is located on the device */
int     Device_No;    /* the device no. which a job is on */
int     Device_L_No;  /* the local position no. which is placed the job */
float   Start_Time;   /* the beginning time of present place */
}
p2[];

```

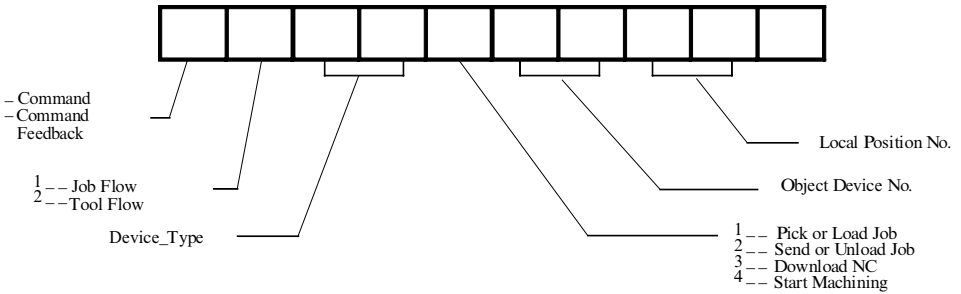


Figure 4. Description of the control command information.

### 5.2.3. 3-dimensional place: the state on a WS and involving a local resource

The basic transition only involves one 3-dimensional place involved, its structure is defined as:

```

struct
{
int    Job_No;           /* the job no. which is located on the device */
int    Device_No;       /* the device no. on which a job is placed */
int    Device_L_No;     /* the local position no. of the device */
int    Auxiliary_Tool_No; /* the used time of the auxiliary tool */
float  Start_Time;     /* the beginning time of present place */
}
p353[];

```

### 5.2.4. The example of basic transitions

The structure of the example is described as:

```

struct
{
Int    Job_No;           /* the job no. */
int    Device_No;       /* the loading/unloading device no. */
int    Device_L_No;     /* the number of position of the device */
int    Auxiliary_Tool_No; /* the auxiliary tool no. used */
float  Fire_Time;      /* the fire time of the transition */
}
t0[];
t0'[];

```

There is a conversion program which interprets the control command issued from a FMS controller. The control command takes the format as shown in figure 4. For example, the first digit can take the number 1 or 2 which represents the command issued from a FMS controller or the feed back command from the environment, respectively.

## 6. A case study: FMS control software testing

Suppose that the control software of the FMS shown in figure 1 needs to be tested. In the case tested, all eight sets of the fixtures and pallets used are interchangeable, and they are available at reach of the loading robot at the beginning. The operation information of the job is listed in table 4. In addition, the loading

Job No.	Type	First process			Second process			Third Process			Due-data	Priority
		Device	NC No.	Time	Device	NC No.	Time	Device	NC No.	Time		
00	00	M <sub>1</sub>	11	17	M <sub>2</sub>	21	10	M <sub>3</sub>	30	5	01/12	01
01												
02	01	M <sub>2</sub>	22	10	M <sub>3</sub>	30	05				02/12	02
03												

Table 4. Process information of the jobs.

robot takes 15 minutes to load a job, and 5 minutes to unload a job, and both loading and unloading operations are regarded as necessary operations.

The initial states of the FMS are described by initial marking. Therefore, initial marks should be assigned before the testing system of GPN starts. The initial marks of 1-dimensional points for the states of the idle fixtures and pallets, i.e.  $F_0$  and  $P_0$ , are expressed as:  $M(F_0) = M(P_0) = 8$ ; therefore, instances are:

$$F0[0] = F0[1] = \dots = F0[7] = \{1, 120.0, 0.0, 0, 0.0\}$$

$$P0[0] = P0[1] = \dots = P0[7] = \{1, 240000.0, 0.0, 0, 0.0\}$$

Similarly, the initial marks of 1-dimensional points for the states of the idle devices and auxiliaries,  $M_0$  and  $MLT_0$ , are expressed as:

$$M(M_0) = 4, M(MLT_0) = 1;$$

$$M0[0] = \{3, 1, \text{'L/U\_station'}, 1, 0.0, 0, 0, \dots, 0\};$$

$$M0[1] = \{6, 2, \text{'M1'}, 2, 0.0, 0, 0.0\};$$

$$M0[2] = \{6, 3, \text{'M2'}, 2, 0.0, 0, 0.0\};$$

$$M0[3] = \{1, 4, \text{'M3'}, 1, 0.0, 0, 0.0\};$$

$$MLT0[0] = 1, \{1\}, \text{'Robot'}, 200000.0, 0.0, 0, 0.0\};$$

According to the description of the case tested, there are two batches of the jobs:

$$\begin{aligned} JT_{type}[0] = & \{\text{'01/12'}, 1, 0, 0, 5, \{1, 1, 1, 1, 1\}, \{\{1\}, \{2\}, \{3\}, \{4\}, \{1\}\}, \\ & \{\{\}, \{11\}, \{21\}, \{30\}, \{\}\}, \{\{18.0\}, \{17.0\}, \{10.0\}, \{5.0\}, \{5.0\}\}, \\ & \{\{0\}, \{0\}, \{0\}, \{0\}, \{0\}\}, \{\{\{\}\}\}\}; \end{aligned}$$

$$\begin{aligned} JT_{type}[1] = & \{\text{'02/12'}, 2, 0, 0, 4, \{1, 1, 1, 1, 1\}, \{\{1\}, \{3\}, \{4\}, \{1\}\}, \\ & \{\{\}, \{22\}, \{30\}, \{\}\}, \{\{18.0\}, \{22.0\}, \{5.0\}, \{5.0\}\}, \\ & \{\{0\}, \{0\}, \{0\}, \{0\}\}, \{\{\{\}\}\}\}; \end{aligned}$$

Then, the initial mark of  $J_0$  is expressed as:

$$M(J_0) = 4;$$

$$J0[0] = J0[1] = \{JT_{type}[0], 5\};$$

$$J0[2] = J0[3] = \{JT_{type}[1], 4\};$$

For the basic transitions, the initial markings for the 2-dimensional point concerned and the 3-dimensional point involved are all defined as  $M(P_0) = 0$ ,  $M(P353) = 0$ , respectively.

When the testing system of GPN has received the first control message, the conversion program interprets that a  $t_0$  transition is fired. The testing system then verifies whether or not the control command is correct according to some criteria predefined such as priority rules, and renews the marks of the above points in the generic Petri net model as such:

Time	Received message	Explanation	Diagnose message
00:00:01	1107100011	Robot is commanded to load Job No. 00	Pass
00:15:03	2107100011	Job No. 00 is loaded at L/U	Pass
00:15:04	1105100011	AGV is commanded to pick Job No. 00	Pass
00:16:30	2105100011	Job No. 00 is picked by AGV <sub>Pass</sub>	Pass
00:16:44	1105200021	AGV is commanded to send Job No. 00	Pass
00:17:01	1107102011	Robot is commanded to load Job No. 02	Warning Error !! L/U_Chose_Job Rule is Priority Rule The set of alternative jobs is: No. 01 The chosen job is: No. 02 L/U_Chose_Job Rule is disobeyed
00:17:05	2105200021	Job No. 00 is moved to local buffer of M <sub>1</sub>	Pass
00:17:23	1102311000	M <sub>1</sub> is commanded to download NC No. 11	Pass
00:17:24	2102311000	NC No. 11 is downloaded	Pass
00:17:25	1102400000	M <sub>1</sub> is commanded to start machining	Pass
00:32:01	2107102011	Job No. 02 is loaded at L/U	Pass
00:32:01	1105102011	AGV is commanded to pick Job No. 02	Pass
00:32:02	1107101011	Robot is commanded to load Job No. 01	Pass
00:33:03	2105102011	Job No. 02 is picked by AGV	Pass
00:33:04	1105202031	AGV is commanded to send Job No. 02	Pass
00:34:18	2105202031	Job No. 02 is moved to local buffer of M <sub>2</sub>	Pass
00:34:19	1103322000	M <sub>2</sub> is commanded to download NC No. 22	Pass
00:34:20	2103322000	NC No. 22 is downloaded	Pass
00:34:25	1102400000	M <sub>2</sub> is commanded to start machining	Pass
00:36:21	2102400000	M <sub>1</sub> has machined Job No. 00	Pass
00:36:22	1105100021	AGV is commanded to pick Job No. 00	Pass
00:37:13	2105100021	Job No. 00 is picked by AGV	Pass
00:37:13	1105200061	AGV is commanded to send Job No. 00	Pass
00:38:20	2105200061	Job No. 01 is moved to buffer No. 00	Pass
00:40:22	1105101011	AGV is commanded to pick Job No. 01	Fatal Error !! AGV is commanded to pick a job: No. 01 The former operation has not finished. Fatal error is produced.

Table 5. A sample of testing process.

$$M(t0) = 1;$$

$$t0[0] = \{00,0,0,0,0.01\};$$

$$M(P353) = 1;$$

$$P353[0] = \{00,0,0,0,0.01\};$$

$$M(J0) = 3;$$

$$J0[0] = \{JType[0], 5\};$$

$$J0[1] = J0[2] = \{JType[1], 4\};$$

$$M(M0) = 3, M(MLT0) = 0;$$

$$M0[0] = \{6, 2, 'M1', 2, 0.0, 0, 0.0\};$$

$$M0[1] = \{6, 3, 'M2', 2, 0.0, 0, 0.0\};$$

$$M0[2] = \{1, 4, 'M3', 1, 0.0, 0, 0.0\};$$

$$M(F0) = M(P0) = 7;$$

$$F0[0] = F0[1] = \dots = F0[6] = \{1, 120.0, 0.0, 0, 0.0\}$$

$$P0[0] = P0[1] = \dots = P0[6] = \{1, 240000.0, 0.0, 0, 0.0\}$$

All the messages received can be diagnosed in the same way. A sample of testing procedure is shown in table 5.

## 7. Discussions and conclusions

To carry out the design and verification of a control software for a FMS effectively, a model for the behaviours of FMSs needs to be developed. This can be fulfilled by applying a Petri net. However, the problem of using an ordinary Petri net to model the FMSs behaviour is that unmanageable complexity could occur both in the size of the net and the way of creating a model. This complexity can be greatly alleviated with the observation as well as idea behind the work of generic Petri net reported here.

In modelling the real world's structures or behaviours, two general strategies are generally used which could be called 'instance pattern' and 'schema/instance pattern'. It is particularly noted that the schema/instance pattern first starts with developing a so-called 'model template' (i.e. generic Petri net model in this paper) and a model of a particular real world problem is then represented as instances of conformity to that template, whereas the instance pattern does not follow that. Obviously, the schema/instance pattern requires a classification of elements of an underlying application. The general idea described in this paper follows the schema/instance pattern. To this connection, the following salient points are summarized.

- (1) Places and transitions are generalized (into a set of templates) and individual jobs and transition events are regarded as instances of the place's and transition's templates.
- (2) Since the model is built as a kind of templates which are only associated with a set of basic events, the number of jobs, devices, etc. the complexity of the model is controllable.

- (3) Since these templates are relatively 'fixed', and can be coded in a predefined manner, the model of a particular application (i.e. instances of the template) can then be easily created. The complexity of the Petri net model of FMS is then reduced.
- (4) Since the proposed generic Petri net is generated by incorporating a Petri net into a general problem description scheme in AI, the existing AI-based problem solving strategies are potentially applicable to generic Petri net modelling and analysis.

The program system developed is based on a generic Petri net description model combined with conventional procedural language. It has potentials that support all development stages including general design, detailed design, programming and testing in a consistent manner. A case study for the testing of FMS controller software show effectiveness and cost saving over development with conventional methods in which only ordinary Petri net and procedural language are used.

### Acknowledgement

The authors want to thank a partial financial support for this research from Natural Science and Engineering Research Council of Canada (NSERC) and Atomic Energy Canada Limited (AECL).

### References

- ARCHETTI, F., LUCERTINI, M. and SERAFINI, P., 1989, *Operations Research Models: in Flexible Manufacturing Systems* (New York: Springer Verlag).
- AL-JAAR, R. Y. and DESROCHERS, A. A., 1990, Performance evaluation of automated manufacturing systems using generalised stochastic Petri nets. *IEEE Transactions on Robotics and Automation*, **6**, 621–639.
- ALTIOK, T., 1996, *Performance Analysis of Manufacturing Systems* (New York: Springer Verlag).
- BECK, C. L., 1986, Models for simulation and discrete control of manufacturing systems. *Proceedings of the IEEE Conference On Robotics and Automation*, San Francisco, CA, pp. 305–310.
- BI, Z. M., 1995, Study on testing mechanism of FMS controllers (in Chinese). *China Mechanical Engineering*, **6**(2), 34–36.
- BI, Z. M., 1996, Debugging in software testing environment for FMS controllers (in Chinese). *Journal of Nanjing University of Science and Technology*, **20**(2), 30–36.
- CHAAR, J. K., 1993, Developing manufacturing control software: a survey and critique. *International Journal of Flexible Manufacturing Systems*, **5**, 53–88.
- COOLAHAN, J. J. E. and ROUSSOPOULOS, N., 1983, Timing requirements for time-driven systems using augmented Petri nets. *IEEE Transactions on Software Engineering*, **9**, 603–616.
- CROCKETT, D., DESROCHERS, A., DICESARE, F. and WARD, T., 1987, Implementation of a Petri net controller for a machining workstation. *Proceedings of the IEEE International Conference on Robotics and Automation*, Raleigh, NC, USA, pp. 1861–1867.
- EZPELETA J. and MARTINEZ, J., 1992, Petri net as a specification language for manufacturing systems. In J. C. Gentina and S. G. Tzafestas (eds), *Robotics and Flexible Manufacturing Systems* (Amsterdam: Elsevier Science), pp. 427–436.
- GENTINA, J. C. and CORBELL, D., 1987, Coloured adaptive structured Petri net: a tool for automatic synthesis of hierarchical control of flexible manufacturing system. *Proceedings of the IEEE Conference on Robotics and Automation*, Raleigh, NC, USA, pp. 1166–1173.
- GIORDANA, A. and SAITTA, L., 1985, Modelling production rules by means of predicate transition networks. *Information Sciences*, **35**, 1–41.

- JENG, M. D., 1993, A review of synthesis technology for Petri nets with applications to automation manufacturing systems. *IEEE Transactions on Man and Cybernetics*, **23**, 301–312.
- JENG, M. D., 1995, Modular synthesis of Petri nets for modelling flexible manufacturing system. *International Journal of Flexible Manufacturing Systems*, **7**, 287–310.
- JENSEN, ?, 1983, Au: please supply.
- JENSEN, K. and ROZENBERG, G. (eds), 1991, *High Level Petri Nets: Theory and Application* (New York: Springer-Verlag).
- KAMATH, M. and VISWANADHAM, N., 1986, Applications of Petri net based models in the modelling and analysis of flexible manufacturing systems. *Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA*, pp. 312–317.
- KASTURA, E., DICESAR, F. and DESROCHERS, A., 1988, Real time of multilevel manufacturing systems using Petri nets. *Proceedings of the IEEE Conf. on Robotics and Automation, Philadelphia, USA*, pp. 1114–1119.
- KROGH, B. H., WILLSON, R. and PATHAK, D., 1988, Automated generation and evaluation of control programs for discrete manufacturing processes. *Proceedings of the International Conference on Computer Integrated Manufacturing, New York*, pp. 92–99.
- LEE, D. Y., 1994, Scheduling flexible manufacturing system using Petri nets and heuristic search. *IEEE Transactions on Robotics and Automation*, **10**, 123–132.
- LEVESON, N. G. and STOLZY, J. L., 1987, Software analysis using Petri nets. *IEEE Transactions on Software Engineering*, **13**, 386–397.
- MARTINEZ, J., MURO, P. and SILVA, M., 1987, Modelling, validation and software implementation of production system high level Petri nets. *Proceedings of the IEEE International Conference on Robotics and Automation, Raleigh, NC, USA*, pp. 1180–1185.
- MURATA, T., 1989, Petri nets: properties, analysis and applications. *Proceedings of the IEEE*, **77**, 541–580.
- MURATA, T., KOMODA, N., MATSUMOTO, K. and HARUNA, K., 1986, Petri net-based controller for flexible and maintainable sequence control and its applications in factory automation. *IEEE Transactions on Industrial Electronics*, **33**, 1–8.
- MURATA, T. and ZHANG, D., 1988, Predicate-transition net model for parallel interpretation of logic programs. *IEEE Transactions on Software Engineering*, **14**, 481–497.
- NAGAO, Y., OHTA, H., URABE, H., AND MATSUBRA, S., 1992, Petri net based programming system for FMS. In J. C. Gentina and S. G. Tzafestas (eds), *Robotics and Flexible Manufacturing Systems* (Amsterdam: Elsevier Science), pp. 295–304.
- PETERSON, J. L., 1981, *Petri Net Theory and the Modelling of Systems* (Englewood Cliffs, NJ: Prentice Hall).
- ROUX, G., DESCOTES GENON, B., AND LADET, P., 1992, Operation checking in flexible manufacturing systems. In J. C. Gentina and S. G. Tzafestas (eds), *Robotics and Flexible Manufacturing Systems* (Amsterdam: Elsevier Science), pp. 305–314.
- SODHI, R. S., ZHOU, M. C. and DAS, S., 1994, *Advances in Manufacturing Systems: Design, Modelling and Analysis*, (Amsterdam: Elsevier Science).
- VALETTE, R., COURVOISIER, M., DEMMOU, H., BIGOU, J. M. and DESCLAUX, C., 1985, Putting Petri nets to work for controlling flexible manufacturing systems. *Proceedings of the IEEE International Symposium on Circuits and Systems, Kyoto, Japan*, pp. 929–932.
- WAINWRIGHT, C. E. R. and THETHI, A. J. S., 1997, The development of an improved model of manufacturing system simulation. *Proceedings of the 13th International Conference on Computer-Aided Production Engineering, Warsaw*, pp. 401–408.
- WANG, H., 1993, Modelling and analysis of flexible manufacturing system using Petri net (in Chinese). Doctoral Thesis, Tsinghua University, P.R. China.
- ZHA, X. F., LIM, S. Y. E., AND FOK, S. C., 1998, Integrated knowledge-based Petri net intelligent flexible assembly planning. *Journal of Intelligent Manufacturing*, **9**, 235–250.
- ZHANG, Q., 1990, *Theory of Solving Problem and its Application* [in Chinese], (P.R. China: QingHua University Publisher).
- ZHOU, M. C., 1989, A top-down approach to systematic synthesis of Petri nets models for manufacturing systems. *Proceedings of the IEEE Conference on Robotics and Automation, Scottsdale*, pp. 534–539.