Int j simul model 15 (2016) 3, 566-576
ISSN 1726-4529 Original scientific paper

SIMULATION STUDY OF FLEXIBLE MANUFACTURING CELL BASED ON TOKEN-ORIENTED PETRI NET MODEL

Nie, X. D.[*,**]; Chen, X. D.[**] & Chen, X.[**]

[*] School of Economics and Commerce, Guangdong University of Technology, Guangzhou, Guangdong, P. R. China

[**] School of Electromechanical Engineering, Guangdong University of Technology, Guangzhou, Guangdong, P. R. China

E-Mail: niexiaodong@gdut.edu.cn, chenxindu@gdut.edu.cn, chenx@gdut.edu.cn

Abstract

An initial manufacturing plan may not consider the particular implementation of flexible manufacturing cells (FMCs). Therefore, the FMC is subject to modelling and simulation to evaluate and correct the production plan using feedback. In the case of highly shared resource contention, deadlock and blocking present an inevitable, unavoidable problem, and corrections may be required for production plans. However, by using existing Petri net model theories and certain simulation software for model establishment, the structure and scale of the model may vary with changes in the parts, machines, and robots. This results in a cumbersome and complicated model building process. To address this practical problem, a token-oriented Petri net model theory is therefore proposed. The movable directions of a token are detected to determine if the firing conditions are satisfied. To avoid deadlock, the token first predicts the status of resource utilization prior to decision-making when entering the transporting-transition state. Accordingly, during the model run, events are triggered against the expected time, and transitions are thereby enabled and fired. An improved machine processing plan and robotic transporting plan may be obtained via path scenario deduction. In this study, classic case data were processed for simulation analysis of the manufacturing job, which verified the validity of the model algorithm.

(Received, processed and accepted by the Chinese Representative Office.)

Key Words: Flexible Manufacturing Cell (FMC), Petri Net, Transporting Robots, Simulation

1. INTRODUCTION

Owing to the organizational hierarchy of an enterprise, as well as cooperation with the delegation of responsibilities, upper management may not fully consider the particular implementation of flexible manufacturing cells. Thus, only an initial production plan involving parts and machines will be formulated. However, in the case of highly shared resource contention, deadlock and blocking present an inevitable, unavoidable problem. Furthermore, corrections may be required for production plans based on the status of the actual implementation. To ensure the implementation of production plans, modelling and simulation are most often required to evaluate and correct the production plan by using feedback and ensuring mutual coordination until the requirements are satisfied.

Petri nets have been used extensively in automated manufacturing [1]. Many research achievements in this area have become available. Li and Zhou and Li et al. employed the process-oriented Petri net (POPN) [2, 3], whereby the flow of system parts can be clearly demonstrated by the model structure. Alla et al. and Viswanadham and Narahari proposed the coloured Petri net (CPN) [4, 5], which enables operation places to accommodate parts of different types, while enabling resource places to contain the various resources. Other scholars have proposed the object-oriented Petri net (OOPN) [6], which features modularity and the coordination of various operating modules via messages and gate connections. In addition, Wu, Wu and Zhou, Wu et al. proposed the resource-oriented Petri net (ROPN) [7-10]. In ROPN, the parts manufacturing process is based on sequential access to resources

according to the parts along predefined processing paths. In terms of dynamic changes, the intelligent token Petri net was established [11]. Hence, once changes, such as machine failures and machine reconfigurations, are detected, a new path flow of parts is constructed to form a new Petri net.

Representative simulation software systems for flexible manufacturing systems developed by China for job-shop scheduling include the following: job-shop scheduling simulation software, factory simulation scheduling environment, intelligent rule-based scheduling system [12], and flexible manufacturing systems simulators [13]. Other reputable flexible manufacturing system simulation software includes AutoMod, Quest, AutoSched, JobTimePlus, Factor, Factor/Aim and SIMNETD. The above simulation software focuses on the layout modelling and operating of specific equipment. Consequently, specific well-targeted statistical analyses and scheduling rules may become available. However, the simulation models vary with the changes in parts and resources.

To address the above issue, a token-oriented Petri net model theory is proposed. Additionally, core algorithms were developed, including an intelligent function of transitions, a deadlock control policy, and scenario deduction. A simulation system was furthermore implemented with functions in deadlock control and plan correction, thereby addressing existing problems in modelling and simulation of manufacturing cells. Moreover, a calculation was performed based on classic case data, which verified the validity of the simulation model algorithm.

2. PROBLEM STATEMENT

In the flexible manufacturing cell, a robot handles the transporting of parts between the equipment and rack. The in-cell equipment is represented by $machine_m$ ($m = 1, 2, \ldots, MC$), the set of rack locations is denoted by *Store*, the input station is represented by *In*, the output station is *Out*, and the transporting robots are denoted by $robot_r$ ($r = 1, 2, \ldots, R$). This paper is based on the following assumptions:

(1) Each machine processes only one part at a time;
(2) Each part is processed on only one machine at the very same moment;
(3) Once a part occupies one specific machine, other part cannot pre-empt it;
(4) One operation may be processed by multiple machines;
(5) The expected processing time of a specific operation may be computed based on an NC program;
(6) Rack, transporting robots, and input and output stations are not subject to faults;
(7) Once the operation of a part is finished, the robot may immediately move to transporting. If the machine required for the next operation is not available, then the part is placed on the rack, which is directly situated on the machine for processing if the target machine is idle;
(8) If the subsequent operation of the part is processed by the same machine, the robot does not proceed with transporting after the current operation is completed;
(9) The part being transported may pre-empt in advance the targeted machine for processing.

Subsequent to the initial production plan developed by upper management, a computational simulation is required to obtain machine processing plans and robotic transportation planning. These are fed back to upper management for evaluation and analysis.

3. TOKEN-ORIENTED PETRI NET

3.1 Definition

The formal definition of a token-oriented Petri net (TOPN) is denoted as:

$$TOPN = (P, IT, F, M, K, E, \Theta) \tag{1}$$

where P represents a set of places, IT is a set of intelligent transitions, F represents a set of arcs, M represents markings, K denotes the capacity function, E represents a real-time enabling function of tokens, and Θ is a set of tokens in the net, of which P, F, M, and K employ definitions that are identical to those of existing Petri nets [11]. Extensions of notations are primarily made to Θ, E, and IT in this paper.

Firstly, Θ is a set of tokens in the Petri net. Information of a token includes basic properties of parts, machines, or robots. Token θ is graphically denoted by a coloured small solid oval. Next, E represents the real-time enabling function of tokens, $\theta.enableNext(\tau)$ denoting the movable direction of a token at a particular time, τ. Lastly, IT represents a set of intelligent transitions, graphically denoted by $|O|$, a circle braced with two vertical lines. Input transition function $it.fireIn(\tau)$ moves appropriate tokens in the pre-place into a transition; output transition function $it.fireOut(\tau)$ moves the in-transition tokens into suitable subsequent places.

In the token-oriented Petri net, places and transitions are defined in alignment with potential path scenarios for tokens. In the manufacturing cell, machines, robots, and parts may be defined as tokens, which undergo clear and bounded path scenarios. For example, machines encounter idle and processing scenarios. Potential scenarios for robots include ‘stationary, fetching a part, carrying a part, placing a part, and no-load motion’. Potential scenarios for parts normally consist of ‘waiting at the external input buffer zone, part input, waiting at the input station, being fetched by the robot form input station, remaining over the robotic arm, being placed onto the machine, being ready for processing, being processed, waiting on the machine, being fetched by robots from the machine, remaining over the robotic arm, being placed onto the output station, being fetched externally, and waiting at the external output buffer zone.

In the case in which no machines are available, the part must be placed on the rack. Possible related scenarios may include ‘remaining over the robotic arm, being robotically placed on the rack location, waiting in the rack, and being selected from the rack by the robots’. Places and transitions are established based on these scenarios and evolution paths. Then, places or transitions of an identical nature are respectively merged. Thus, tokens reside in and move through these transitions and places aligned with their own properties.

Despite the complicated and diversified layouts of manufacturing cells in real-world manufacturing, modelling based on potential path scenarios of machines, robots, and parts, as well as the structure and scale of the established model, may be fixed. Furthermore, tokens must only reside in and move through a constant number of places and transitions, thereby changing in machines, robots and parts do not require repeated modelling.

3.2 Identification of Petri nets

Tokens are significant in the token-oriented Petri net. Therefore, the traditional approach to identifying Petri nets using the number of tokens in the place, or token colouring, has become unsuitable. Hence, the marking method adopted in this paper is denoted as:

$$M(Net) = \sum P_p\{\theta_1, \theta_2, \ldots, \theta_u\} + \sum IT_{it}\{\theta`_1, \theta`_2, \ldots, \theta`_v\} (u, v = 1, 2, \ldots) \tag{2}$$

where Net is TOPN, $M(Net)$ represents the marking of this net, $P_p\{\theta_1, \theta_2, \ldots, \theta_u\}$ means that tokens $\theta_1, \theta_2, \ldots, \theta_u$ reside in place P_p, and $IT_{it}\{\theta`_1, \theta`_2, \ldots, \theta`_v\}$ means that tokens $\theta`_1, \theta`_2, \ldots,$

$\theta`_v$ reside in transition IT_{it}. For simplicity in markings, places or transitions with no residing tokens may be omitted.

3.3 Formal statement of scenario deduction

Assume that the marking of the Petri net is M at particular time τ. After a certain duration of time, a firing event occurs at time $\tau`$, which changes the marking of the Petri net from $M$ to $M`$ In the case in which transition (it) input is enabled to fire, the corresponding appropriate tokens in the pre-place are moved into transition (it) to create new tokens.

$$M`(p) = M(p) - \{\theta_p\},\ p \in {}^{\bullet}it \tag{3}$$

$$M`(it) = M(it) + \{\theta_{it}\} \tag{4}$$

where θ_p represents tokens in place p that are selected to move, $p \in {}^{\bullet}it$ denotes the pre-place of transition (it), and θ_{it} represents newly created tokens in transition (it).

In the case in which transition (it) output is enabled to fire, the corresponding appropriate tokens in (it) are moved into the corresponding subsequent place to create new tokens.

$$M`(it) = M(it) - \{\theta_{it}\} \tag{5}$$

$$M`(p) = M(p) + \{\theta_p\},\ p \in it^{\bullet} \cap \theta_{it}{}^{\bullet} \tag{6}$$

where θ_{it} represents tokens in the transition (it) selects to move, θ_p represents newly created tokens in place p, $it^{\bullet}$ denotes the subsequent place of transition (it), and $\theta_{it}{}^{\bullet}$ represents a set of movable directions of tokens. To note, after the enabled firing of the transition output, new creation of tokens will not occur in all subsequent places of this transition, which will only occur in the subsequent places directed by $\theta_{it}{}^{\bullet}$.

After the fired transition, the new marking of the Petri net is denoted by $M`= M`_P + M`_{IT}$, which corresponds to time $\tau`$. By combining the marking and corresponding time as a system state, $S(M, \tau) \rightarrow S(M`, \tau`)$ denotes that the system state changes from $S(M, \tau)$ to $S(M`, \tau`)$ after the duration of time ($\tau`$-τ).

4. MAJOR ALGORITHMS

4.1 Intelligent function of transition

When token θ in place p at particular time τ fires an event, $it.fireIn(\tau)$ ($it \in p^{\bullet}$) executes the algorithm outlined below.

Algorithm of transition input function

Step 1: Check if ${}^{\bullet}it$ are all marked; if not, then proceed to Step 10;

Step 2: Scan ${}^{\bullet}it$, obtain the total number of previous places, m, and then initialize $i = 1$;

Step 3: Obtain previous place: $P_{pre}[i] \in {}^{\bullet}it$;

Step 4: Obtain the total number of tokens n for $P_{pre}[i]$, and initialize the set of movable tokens $TokensCarry_i$, $j = 1$;

Step 5: Obtain token θ_j. If $it \in \theta_j.enableNext(\tau)$, then add token θ_j to $TokensCarry_i$;

Step 6: $j = j + 1$. If $j \leq n$, then proceed to Step 5; otherwise, perform Step 7;

Step 7: If $TokensCarry_i$ is null, then proceed to Step 10;

Step 8: $i = i + 1$. If $i \leq m$, then proceed to Step 3; otherwise, perform Step 9;

Step 9: Move tokens sorted first in the production plan from $TokensCarry_i$ to transition (it);

Step 10: End the algorithm.

When token θ in transition (it) at particular time τ fires an event, it implements the algorithm outlined below.

Algorithm of transition output function

Step 1: Execute $\theta.enableNext(\tau)$. If it is null, then proceed to Step 7; otherwise, execute the capacity function K, where $\theta.enableNext(\tau)$ points. In the case of the upper bound of the capacity, proceed to Step 7;

Step 2: Obtain the total number of subsequent places m that are accessible to tokens, and initialize $i = 1$;

Step 3: Obtain subsequent place, $P_{post}[i] \in \theta.enableNext(\tau) \subseteq it^{\bullet}$;

Step 4: Separate elements from token, and create new tokens to deposit in place $P_{post}[i]$;

Step 5: $i = i + 1$. If $i \leq m$, then proceed to Step 3; otherwise, perform Step 6;

Step 6: Remove token θ from transition (it);

Step 7: End the algorithm.

4.2 Deadlock and blocking control policy

In flexible manufacturing cells furnished with transporting robots, deadlock and blocking result from incorrect robotic transporting instructions. To avoid this problem, token θ representing the part first predicts the status of resource utilization prior to decision-making on entering the transporting transition state. In this case, tokens in rack places are employed. The algorithm below is an example.

Algorithm of real-time enabling function of tokens in rack places

Step 1: If the processing of the part represented by token θ is finished, then return the transporting transition and proceed to Step 6;

Step 2: Obtain set *machineSet* from the place of idle machines. If it is null, then return null and proceed to Step 6;

Step 3: Identify from the system the part being robotically transported. In the case in which this part will soon occupy a machine, then add that machine to set *machineSetOccupy*. In addition, remove the element contained in *machineSetOccupy* from *machineSet*;

Step 4: Obtain the set of machines, *machineRequire*, that are available to perform the next operation for the part represented by token θ;

Step 5: If an element of *machineRequire* is also contained in *machineSet*, then return the transporting transition; otherwise, return null;

Step 6: End the algorithm.

4.3 Scenario deduction algorithm

Machines selected in alignment with operations, including operation sequences identified in the job-shop plan, adopt the coding method defined in the literature [14]. In real-world manufacturing, the job-shop production plan is typically first identified. Then, performance indicators are computed. Accordingly, events are fired during the model run based on the planning and scheduling prioritization and expected processing time.

However, the initial job-shop production plan may not account for the particular implementation, which is subject to corrections in the actual manufacturing. Tokens in the Petri net model continue firing events to enable transitions for scenario deductions. This improves job-shop production plans; moreover, robotic transporting instructions are enabled via path scenario deduction based on the Petri net model theory. The scenario deduction algorithm is described below.

Scenario deduction algorithm

Step 1: Place *currentState* and *currentTime* in *statePath*;

Step 2: Obtain the initial job-shop production plan, identify the machines selected in alignment with operations and operation sequences, and obtain the list of operations (*operationList*). Additionally, obtain the total number of operations, *m*, initialize $i = 1$, and set the status change flag, *isChange*, to false;

Step 3: Scan the list of operations (*operationList*), and find operation *i*;

Step 4: Find the part aligned with this operation, and identify the token containing this part under *currentState*;

Step 5: Let this token fire an event at *currentTime* and report its property;

Step 6: Combine related tokens to determine whether some transitions can be enabled;

Step 6.1: If no transitions can be enabled, then $i = i + 1$. If $i \leq m$, then proceed to Step 3;

Step 6.2: If a transition can be enabled, then perform an intelligent transition to obtain the new state, and update the status change flag, *isChange*, to true;

Step 7: If this token fires an end event, and this operation is not the last one of a certain part, then add this operation to *finishedOperationList*. If this operation is the last one of a certain part, then wait until the robotic transporting is completed before adding it to *finishedOperationList*;

Step 8: If $i < m$, then $i = i + 1$, and proceed to Step 3. Otherwise, perform Step 9;

Step 9: Remove the operations contained in *finishedOperationList* from *operationList*;

Step 10: Determine whether the state changes. If it changes, then is*Change = true*. If there still exists an operation in *operationList*, then $i = 1$, and proceed to Step 3. Otherwise, perform Step 11;

Step 11: Based on *currentTime* and the current Petri net marking, create a new state and place it in *statePath*;

Step 12: Based on the properties of tokens in the current Petri net marking, as well as on the expected time of firing events, calculate the time required for the next fireable event triggered by tokens;

Step 13: If the time required for the next fireable event triggered by tokens is infinite, no fireable events will be triggered by tokens. Proceed to Step 16;

Step 14: If the time required for the next fireable event triggered by at least one token is not infinite, i.e. there exists at least one token to trigger a fireable event, then find the soonest time to trigger the event as the new *currentTime*;

Step 15: If the new *currentTime* does not exceed the defined time duration, *T*, and operations still exist in *operationList*, then initialize $i = 1$, set the status change flag *isChange* to false, and proceed to Step 3. Otherwise, perform Step 16;

Step 16: End the algorithm; and *statePath* is attained.

Algorithm to obtain robotic transporting instructions from path scenario deduction

Step 1: Obtain the total number of state from state path, *n*, and initialize $i = 1$;

Step 2: Scan state path, find state *i*, and produce the Petri net marking;

Step 3: Obtain the total number of tokens, *m*, in the Petri net marking, and initialize $j = 1$;

Step 4: Produce token *j;*

Step 5: If the token belongs to the one representing an idle robot, and $j < m$, then $j = j + 1$, and proceed to Step 4;

Step 6: If the token does not contain a robot and $j < m$, then $j = j + 1$, and proceed to Step 4;

Step 7: Obtain the robot from the token;

Step 8: Add the token to the robotic transporting instruction list;

Step 9: If $j < m$, then $j = j + 1$, and proceed to Step 4;

Step 10: If $i < n$, then $i = i + 1$, and proceed to Step 2;

Step 11: Obtain the robotic transporting instructions, and graphically denote with Gantt charts the planning of these instructions.

Machines are selected in alignment with operation sequences defined in the job-shop production plan. Moreover, the processing status of various operation sequences can be learned from the parts. Therefore, the status of operation sequences processed by all machines, as well as the improved job-shop production plan, can be obtained simply by acquiring the Petri net marking of the last state contained in state path.

Algorithm to obtain operation sequences of machines from path scenario deduction

Step 1: Obtain the state path;

Step 2: Scan state path, find the last state, and produce the Petri net marking;

Step 3: Obtain the total number of tokens, m, and initialize $i = 1$;

Step 4: Produce token i;

Step 5: If the token belongs to the input buffer and $i < m$, then $i = i + 1$, and proceed to Step 4;

Step 6: If the token does not contain the part and $i < m$, then $i = i + 1$, and proceed to Step 4;

Step 7: Obtain the data of the part from the token. In addition, obtain the total number of operations, n, to be processed for this part, and initialize $j = 1$;

Step 8: Read operation j to be processed for this part;

Step 9: If the current operation is not assigned to a machine and $j < n$, then $j = j + 1$, and proceed to Step 8;

Step 10: If the current operation is not being processed and $j < n$, then $j = j + 1$, and proceed to Step 8;

Step 11: If the current operation is not completed, then the completion time will be computed based on the expected processing time;

Step 12: Obtain the machine assigned to the operation;

Step 13: Add the given operation to the machine operation list;

Step 14: If $j < n$ then $j = j + 1$, and proceed to Step 8;

Step 15: If $i < m$, then $i = i + 1$, and proceed to Step 4;

Step 16: Obtain the operation lists of all machines, and graphically denote with Gantt charts the jobs of all machines.

5. MODEL ALGORITHM APPLICATIONS

Java was adopted to implement the simulation model algorithm proposed in this paper. Some of the jobs for the flexible job-shop were sourced from the literature [14], as presented in Table I. The time for robotic motion between adjacent equipment was assumed to be one.

The initial production planning and scheduling given by upper management are shown in Table II. The fireable events were triggered based on machines selected in alignment with the operations and the operation sequences defined in the initial production plan. A portion of the

Gantt chart screen capture, as shown in Fig. 1, was obtained following a simulation model run.

The production plan for the actual manufacturing operation is presented in Table III. Obtained after the simulation computation, it can be adopted for communicating with upper management or for guiding the preparation of materials. Nevertheless, it is best targeted to real-life production scenarios.

Table I: Flexible job-shop problem of six machines and three parts.

Parts	**Operation**	**Candidate Processing Machines**					
		M1	M2	M3	M4	M5	M6
J1	O11	2	3	4	---	---	---
	O12	---	3	---	2	4	---
	O13	1	4	5	---	---	---
J2	O21	3	---	5	---	2	---
	O22	4	3	---	---	6	---
	O23	---	---	4	---	7	11
J3	O31	5	6	---	---	---	---
	O32	---	4	---	3	5	---
	O33	---	---	13	---	9	12

Table II: Initial production planning and scheduling.

SN	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Operation	O11	O31	O21	O12	O22	O13	O32	O33	O23
Machine	M2	M1	M5	M4	M1	M1	M4	M6	M5
Time	3	5	2	2	4	1	3	12	7

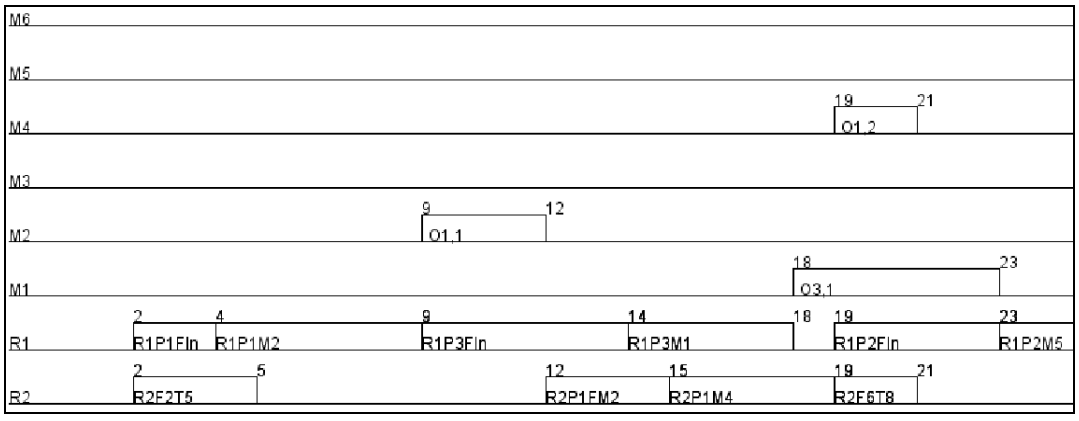

Figure 1: Gantt chart of in-cell processing and transporting.

Table III: Actual production planning and scheduling.

SN	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Operation	O11	O31	O12	O21	O13	O32	O33	O22	O23
Machine	M2	M1	M4	M5	M1	M4	M6	M1	M5
Time	3	5	2	2	1	3	12	4	7

In the Gantt chart in Fig. 1, the rectangle denotes the start or completion time of the operation. The description of the transporting instructions for robots consists of the following four portions. The first portion refers to robots (R1), the second portion indicates parts (P1, P2), the third portion denotes whether to fetch a part (F) or to place a part (omitted here), and the last portion represents the equipment. For example, R1 P1 Fin means that robot R1 fetches part P1 from input station In. R1 P1 M2 means that robot R1 places part P1 on machine M2. In the case of a robotic in-motion dodge, F represents 'From' and T denotes 'To' in terms of the position movement. For example, R2 F2 T5 means that robot R2 moves from Position 2 to Position 5.

An additional test of the simulation algorithm proposed in this paper was conducted. The data on various scales and different degrees of flexibility sourced from the flexible job-shop classic case [15] were processed for the manufacturing job computation. Owing to space constraints herein, and without loss of generality, an mt06 case with three degrees of flexibility sourced from *vdata* of Hurink data is presented. Figs. 2 and 3 show Gantt charts with two and three robots, respectively, which were obtained following a model run of the simulation algorithm.

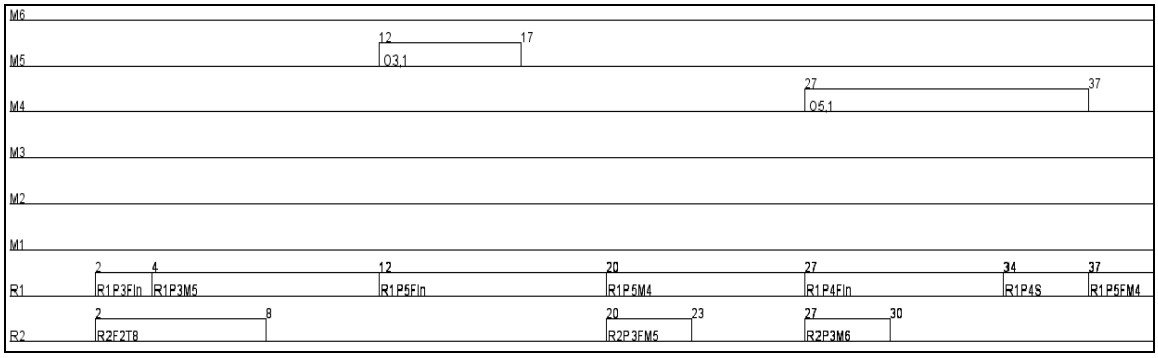

Figure 2: Gantt chart screen capture of an mt06 case with three degrees of flexibility and two robots.

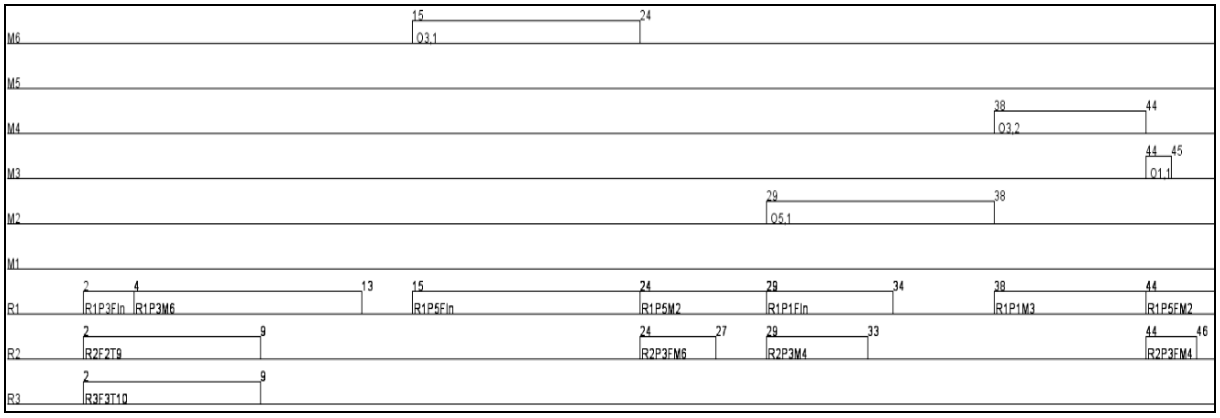

Figure 3: Gantt chart screen capture of an mt06 case with three degrees of flexibility and three robots.

Based on the computational experiment, and by using the simulation system developed according to the token-oriented Petri net model theory, the flexible manufacturing cell had to be modelled only once. From that point, the input of information on parts, machines, and robots enabled the simulation model to be run for operating results. Changes in the above information did not require repeated modelling. Moreover, the production plan derived from the simulation remained deadlock-free for a smooth implementation. The Gantt charts for the evaluation and feedback were thereby produced for machine processing and robotic transporting plans. This result thus verifies the validity of the proposed simulation model algorithm.

6. CONCLUSION

The initial master manufacturing plan typically considers only parts and machines. Therefore, the flexible manufacturing cell is subject to the modelling and simulation for evaluating and correcting the production plan with feedback. Nevertheless, by using the traditional Petri net model theory for model establishment, the structure and scale of the model can vary with changes in parts, machines, or robots, which results in a cumbersome and complicated model building process.

In current simulation software for flexible manufacturing systems, layout modelling and the operating of specific equipment are emphasized. Therefore, the simulation model varies in accordance with the changes in parts and resources. The token-oriented Petri net model theory was therefore proposed. Additionally, core algorithms were developed, including the intelligent function of transitions, a deadlock control policy, and scenario deduction, to address existing problems in the modelling and simulation of manufacturing cells.

In this paper, the classic case data were processed for computational experiments of the manufacturing job. Accordingly, the flexible manufacturing cells were not subject to repeated modelling. The production plan derived from the simulation remained deadlock-free for smooth implementation, and Gantt charts for evaluation and feedback for machine processing and robotic transporting plans were effectively produced. This result verifies the validity of the proposed simulation model algorithm.

ACKNOWLEDGEMENT

This paper was funded by the National Natural Science Foundation of China (Grant No. 51105082). We would like to thank Editage [www.editage.cn] for English language editing.

REFERENCES

[1] Holloway, L. E.; Krogh, B. H.; Giua, A.(1997). A survey of Petri net methods for controlled discrete event systems, *Discrete Event Dynamic Systems: Theory and Applications*, Vol. 7, No. 2, 151-190, doi:10.1023/A:1008271916548

[2] Li, Z.-W.; Zhou, M.-C. (2004). Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems, *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, Vol. 34, No. 1, 38-51, doi:10.1109/TSMCA.2003.820576

[3] Li, Z.-W.; Hu, H.-S.; Wang, A.-R. (2007). Design of liveness-enforcing supervisors for flexible manufacturing systems using Petri nets, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Vol. 37, No. 4, 517-526, doi:10.1109/TSMCC.2007.897333

[4] Alla, H.; Ladet, P.; Martinez, J.; Silva-Suarez, M. (1984). Modelling and validation of complex systems by coloured Petri nets application to a flexible manufacturing system, Rozenberg, G. (Ed.), *Advances in Petri Nets 1984*, Springer-Verlag, Berlin, Vol. 188, 15-31, doi:10.1007/3-540-15204-0_2

[5] Viswanadham, N.; Narahari, Y. (1987). Coloured Petri net models for automated manufacturing systems, *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, 1985-1990

[6] Miyamoto, T.; Kumagai, S. (2005). A survey of object-oriented Petri nets and analysis methods, *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E88-A, No.11, 2964-2971, doi:10.1093/ietfec/e88-a.11.2964

[7] Wu, N.-Q. (1997). Avoiding deadlocks in automated manufacturing systems with shared material handling system, *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, 2427-2433

[8] Wu, N.-Q. (1999). Necessary and sufficient conditions for deadlock-free operation in flexible manufacturing systems using a coloured Petri net model, *IEEE Transactions on Systems, Man,*

and Cybernetics, Part C (Applications and Reviews), Vol. 29, No. 2, 192-204, doi:10.1109/5326.760564

[9] Wu, N.-Q.; Zhou, M.-C. (2001). Avoiding deadlock and reducing starvation and blocking in automated manufacturing systems, *IEEE Transactions on Robotics and Automation*, Vol. 17, No. 5, 658-669, doi:10.1109/70.964666

[10] Wu, N.-Q.; Zhou, M.-C.; Li, Z.-W. (2008). Resource-oriented Petri net for deadlock avoidance in flexible assembly systems, *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, Vol. 38, No. 1, 56-69, doi:10.1109/TSMCA.2007.909542

[11] Wu, N.-Q.; Zhou, M.-C. (2011). Intelligent token Petri nets for modelling and control of reconfigurable automated manufacturing systems with dynamical changes, *Transactions of the Institute of Measurement and Control*, Vol. 33, No. 1, 9-29, doi:10.1177/0142331208095622

[12] Xiong, G.-L.; Wang, X. (1999). Application and development of simulation technology in manufacturing, *Acta Simulata Systematica Sinica – Journal of System Simulation*, Vol. 11, No. 3, 145-151, doi:10.16182/j.cnki.joss.1999.03.001 (in Chinese)

[13] Li, X. N.; Yuan, H. B.; Cheung, E. H. M. (1998). A new FMS simulator with object-oriented-programming techniques, *Journal of Materials Processing Technology*, Vol. 76, No. 1-3, 238-245, doi:10.1016/S0924-0136(97)00354-3

[14] Gao, L.; Zhang, G.-H.; Wang, X.-J. (2012). *Intelligent Algorithm of Flexible Job Shop Scheduling and its Application*, Huazhong University of Science and Technology Press, Wuhan, 31-32 (in Chinese)

[15] Mastrolilli, M. Flexible Job Shop Problem, from *http://people.idsia.ch/~monaldo/fjsp.html*, accessed on 25-08-2016