Modeling, Implementation and Simulation of Virtual Factory Based on Colored Timed Petri Net

Yang Jianhua
Dept. of Electrical & Computer Eng.
Yokohama National University
Tokiwadai 79-5, Hodogayaku, Yokohama 240-8501
Japan

Yasutaka Fujimoto
Dept. of Electrical & Computer Eng.
Yokohama National University
Tokiwadai 79-5, Hodogayaku, Yokohama 240-8501
Japan

Abstract − **The Real-time Colored Petri Net (RTCPN), a modification of traditional Colored Timed Petri net, is proposed to describe virtual factory. In RTCPN, firing discipline differs from traditional way in that the concept of time-enabled transition is introduced to implement real-time simulation. In this paper, we address how to describe virtual factory using RTCPN model. Combined with block diagram, the concept of virtual place is also proposed to model large-scale factory rapidly and conveniently. At last, a Java-built-in implementation of RTCPN tool is developed and a Printed-Circuit-Board factory is given to investigate time cost of developed tool. Concurrent feature of RTCPN is also discussed to decrease simulation time and improved results shows its effectiveness.**

I INTRODUCTION

Virtual factory [1][2] can be regarded as a mapping in computer world of an actual factory. A three-level virtual model, composed of virtual enterprise, virtual factory and virtual device, is introduced hereby to limit the range of our study. Virtual factory is assumed that all jobs, hereafter only discrete manufacturing considered, are given where predicted jobs are also included. And optimization, if needed, is concerned in sense of factory level. No detailed inner behavior should be known for an operation on a device, which will be included in virtual device level [3][4]. Simply, virtual factory is concerned with the job flow, processed on machines, transferred by conveyors and stored in buffers.

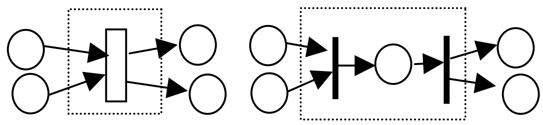
Many methodologies have been introduced to illustrate the mechanism of virtual factory. Among them, Petri net technique [5][6][7] is widely applied due to not only its powerful graphic capacity but also its generalized applications on almost all levels in a manufacturing enterprise. In this work we propose a modification of Colored Timed Petri net, Real-time Colored Petri Net (RTCPN) to describe the manufacturing process, where its firing discipline differs from traditional manner in order to let all transitions work according to the same mode.

In this work, we address how to simulate a virtual factory using RTCPN model. A java-built-in implementation is developed to investigate simulation time cost on different computing environments. An algorithm is developed to reduce simulation time based on concurrence features existing in our proposed RTCPN model.

II REAL-TIME COLORED PETRI NET

High-Level net, Colored Petri Net [8][9] is a modification of low-level Petri net proposed by Dr C. A. Petri in early 1960s [10]. It had been widely applied to solve various practical problems by the use of tokens that can distinguish from each other. Simply a colored Petri net can defined as a 5-tuple set $CPN = \{P, T, A, M_0, C\}$, where P, T, A, M_0 stand for place, transition, arc, and initial marking as described in low-level Petri net, and C for token's color. Furthermore, Colored Timed Petri Net appears while the element "Time" is introduce to Colored Petri Net. In fact, the element "Time" can be also added into low-level Petri Net, generating so-called Timed Petri Net. Thus Colored Time Petri Net is a combination of Colored Petri Net and Timed Petri Net.

Traditionally two policies are adopted to introduce the element "Time" [11][12]. The first, the transition is timed, denoted by a box as shown in Fig.1 (a). It takes specified time to complete the firing of a transition, moving specified tokens into corresponding output places. The second, the place is timed. A token will be kept in the input place until its delay time is over. In practice, timed transition based model can also be transformed into timed place based model as shown in Fig.1 (b). Each timed transition is converted to two instant transitions, denoted by bars, and one timed place. Two instant transitions stand for the start and the end of the firing of timed transition respectively. The timed place stands for the firing status of timed transition.



(a) Timed Transition (b) Timed Place
Fig.1 Time Petri Net

Usually the timed transition based model is more widely applied than the timed place based model because of its simplicity and easy understanding. But both of them ignore a case that new tokens may be added in the future. To solve this problem, firing rule of some transitions should be modified to receive new tokens available in the future. However it will result in destroying the consistence of transition firing rule because some transitions deal with new tokens but others needn't. Therefore we modify above policies by attaching a time point to tokens and changing its firing rule. Fig.2 illustrates its mechanism.

In Fig.2, the tokens are classified into black token and colored token. In definition of Colored Petri Net, it is unnecessary to distinguish them because all of tokens are colored. The black token introduced hereby is to conveniently model many practical problem. The black token is the same as the one in ordinary Petri net thus it can easily describe conditions such as logical switch. On the other hand, each colored token in our proposed Colored Timed Petri Net features a time point while the black token is attached nothing. The availability of colored token is related to its time point and current time while the black token is always available. If the time point is earlier than current time, the colored token will be available. If not, the colored token will be unavailable. Moreover, each place will specify which kind of tokens can be received. There are three cases: (1) black token can be received; (2) colored token can be received; and (3) a combination of black token and colored token can be received. A place only receives tokens whose color is specified by the place. In Fig.2, it is supposed that there are one black token in place p_1, and two colored tokens, a, b, in place p_2. Place p_3 is specified to receive black tokens and place p_4 the colored tokens. Let current time point be χ_{cur}, time point of colored tokens a, b be χ_a, χ_b. Let $\mu(p)$ represent the number of tokens in place p, including black tokens and colored tokens. We specify that transition t_1 be fired if $\mu(p_1) > 0$ && $\mu(p_2) > 0$ && ($\chi_a \le \chi_{cur}$ || $\chi_b \le \chi_{cur}$). For an example, if we have $\mu(p_1) > 0$ && $\mu(p_2) > 0$ && $\chi_a \le \chi_{cur}$, the place p_4 will be added a new colored token a whose time point becomes $\chi_a = \chi_{cur} + \tau_a$, where τ_a stands for its time cost on transition t_1. Meanwhile, the place p_3 will be added a black token.

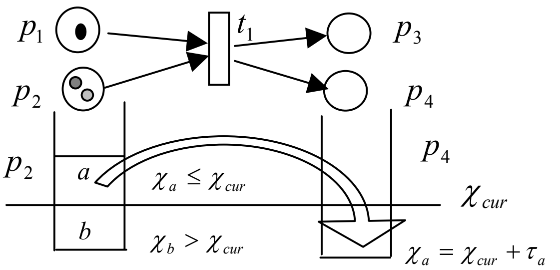


Fig.2 Real-time Colored Petri net

Next we give the definition of Colored Timed Petri net proposed in this paper. To emphasize its real-time characteristic, we name it as Real-time Colored Petri net.

[Definition 1: Real-time Colored Petri Net] A Real-time Colored Petri net is defined as a 7-tuple set $RTCPN = \{P, T, A, M_0, C, \Gamma, \chi_{cur}\}$ where Γ stands for time cost of colored tokens C on transitions T and χ_{cur} for the current time point. The current time point χ_{cur} will vary when RTCPN works. Each colored token is attached a time point which is also changeable with the running of RTCPN.

[Definition 2: Enabled Transition] A transition t_j is enabled if $\mu(p_i) \ge 1$ for all $p_i \in P_j$, where P_j stands for input places of transition t_j, $\mu(p_i)$ for tokens in place p_i including black tokens and colored tokens.

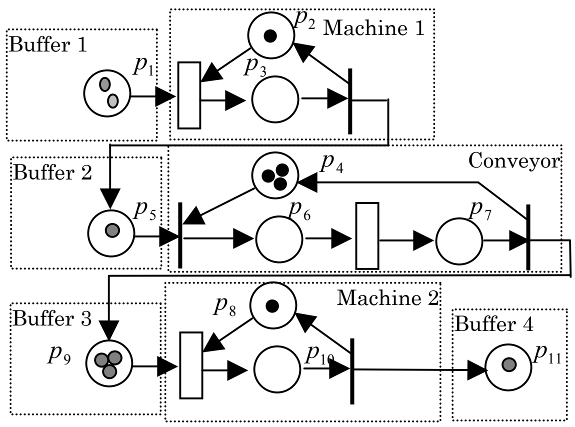
In our proposed Real-time Colored Petri net, the concept of "Enable" of transition keeps the same as that in low-level Petri net. Nevertheless, an enabled transition may not be fired unless it is a time-enabled one defined as follows.

[Definition 3: Time-enabled Transition] A transition t_j is time-enabled if t_j is enabled and $\exists \chi_c \le \chi_{cur}$ for all $p_i \in P_j^c \subset P_j$ where P_j^c, a subset of P_j, stands for places with colored tokens, χ_c for time point of certain colored token, χ_{cur} for current time point.

Based on above definition, the discipline of transition firing is introduced as follows.

[Firing discipline] A transition will be fired at once if only if it is time-enabled.

Fig.3 shows a RTCPN model of a mini-factory composed of two machines, one conveyor and four buffers. The black tokens in Fig.3 express available resources in the mini-factory. That no black token is set to buffers means that the capacity for all buffers is unlimited. It is possible to ignore the limitation of buffers if we know that no overflow occurs. In Fig.3, colored tokens will appear in places without black token pre-distributed. And some of them will be thrown into those places combined with black token.



p_1: Buffer 1 stores jobs, receiving colored token.

p_2: Machine 1 is idle, receiving black token.

p_3: Machine 1 is busy, receiving combination of black token and colored token.

p_4: Conveyor is idle, receiving black token.

p_5: Buffer 2 stores jobs, receiving colored token.

p_6: Conveyor mounts parts, receiving combination of black token and colored token.

p_7: Conveyor transfers parts, receiving combination of black token and colored token.

p_8: Machine 2 is idle, receiving black token.

p_9: Buffer 3 stores jobs, receiving colored token.

p_{10}: Machine 2 is busy, receiving combination of black token and colored token.

p_{11}: Buffer 4 stores jobs, receiving colored token.

Fig.3 A RTCPN model of a mini-factory

Considering the problem of predicting complete time for ordered jobs, we know that the process of simulation is to move all colored tokens in buffers 1, 2, and 3 to buffer 4. It means that no enabled transition exists when simulation is end. Fig.4 gives the entire flow chart of simulation for a RTCPN, where all input places of enabled transitions are investigated to get next time point when no enabled transition is available at current time.

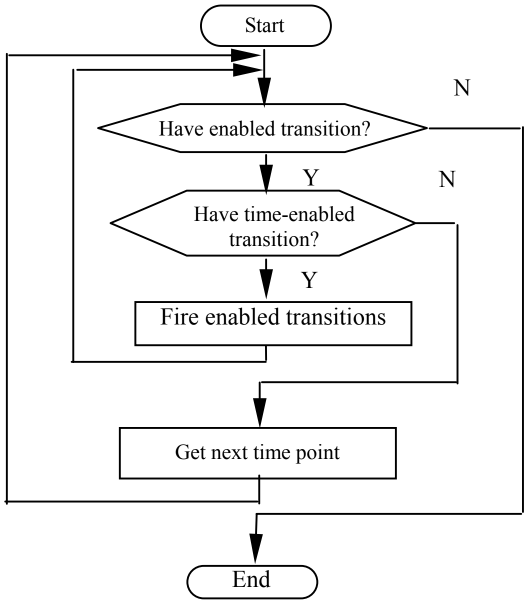


Fig.4 The simulation process of RTCPN

III CONTRUCTION OF VIRTUAL FACTORY

More factors should be included when RTCPN is employed to model a virtual factory. Fig.3 depicts the outline structure of virtual factory. A man-machine interface is always needed to build RTCPN model according to structure of actual factory. The Simulator is an engine of the virtual factory, which generates various results according to practical use purpose, such as job scheduling, performance evaluation, completion time predication etc.

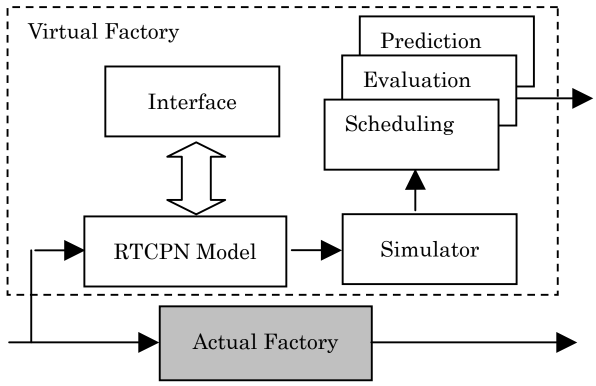


Fig.5 Structure of Virtual Factory

For an actual factory consisting of quantity of machines, buffers and conveyors, it may take time to construct its RTCPN model if we try to model it directly. Block diagram is a convenient way to simplify modeling, especially for blocks with the similar manner. Generally, a block diagram can be equivalent to a manufacturing cell if the detailed information in the manufacturing cell is ignored. In practice, many manufacturing cells work in similar way except that their processing parameters differ. Therefore a sub-RTCPN model of a block can be easily copied to build another one and decrease the time cost. Between two sub-RTCPN models, we introduce a particular place, referred to as virtual place, to connect them. Fig.6 is an example of virtual place.

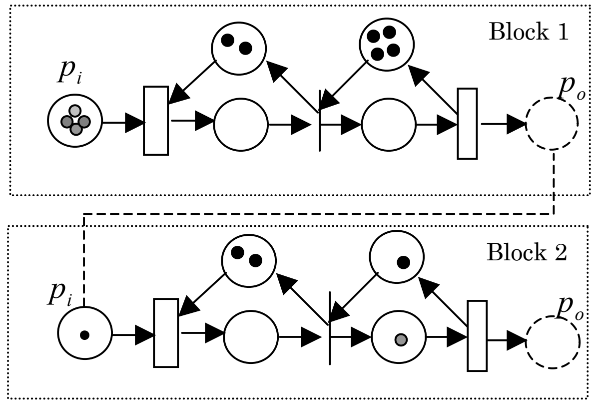


Fig.6 Virtual place and connection of two blocks

In Fig.6, the place p_o in Block 1 is completely the same as the place p_i in Block 2. They own the same tokens and corresponding colors. The Block 1 in Fig.6 has the same structure as the Block 2 thus it can be copied to Block 2 once it has been built. As pointed out previously, the tokens in corresponding place and the fire time of corresponding transition differ.

In some studies [13][14], hierarchical Petri net model is developed to describe large-scale manufacturing system, whose drawback is that the simulator will become complex and hard to implement. In fact, all simulation operations should be performed at the lowest level for a hierarchic Petri net model. Therefore in this work all elements of RTCPN model are atomic. Combined with block diagram, it might rapidly construct a RTCPN model of large-scale manufacturing system, making simulation simple and easy as well.

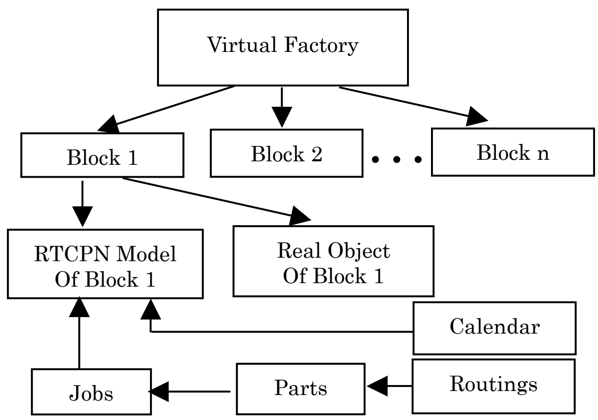


Fig.7 Implementation structure of Virtual Factory

Fig.7 shows the inner structure of virtual factory implementation. Hereby exchanging information with external environment is also provided, including future jobs and work-in-process jobs, whose parts and routings are given by CAPP (computer-aided-process-plan).

In Fig.7, the calendar, describing workday and their work time, appears. For most real enterprises, it is necessary because few factories work continuously without having a rest. Usually the calendar is attached to transitions to determine when it is available. The firing process of a transition should be modified as follows due to calendar.

[Firing discipline including calendar] Let τ_c be firing time of a colored token c on transition t_j. Let current time point be χ_{cur}. The colored token c's entering time χ_c into its corresponding output place can be calculated by $\chi_c = \chi_{cur} + \tau_c + \delta$, where δ is an offset of rest time.

In our work, the workflow of virtual factory is designed as follows.

Step 1: Determine how many manufacturing cells should be modeled and how they are divided into corresponding blocks.

Step 2: Draw and describe blocks including their location, shape, identifier, characteristics etc.

Step 3: Focus on each block, drawing and describe its elements such as machines, buffers and/or conveyors.

Step 4: Focus on each block, building its corresponding RTCPN model. It can be shared or be copied from another block.

Step 5: Determine which places are related to final results, to be used to show GANTT and LOAD graph.

Step 6: Determine which transitions should be attached by a calendar.

Step 7: Given black tokens, which are obtained from elements in the block, and colored tokens, which are read from external file or database. For virtual factory, colored tokens will be applied to represent jobs including corresponding information of parts and routings.

Step 8: Simulate.

Step 9: Show GANTT and LOAD graph, including complete time for each job.

IV EXAMPLE AND TIME COST

A PCB (printed circuit board) production factory, composed of 29 manufacturing cells each of which includes several machines, is used to demonstrate our proposed methodology. Basically it is a flow-shop, shown in Fig.8, except that three cells can receive jobs and some machines are shared at two processing stages.

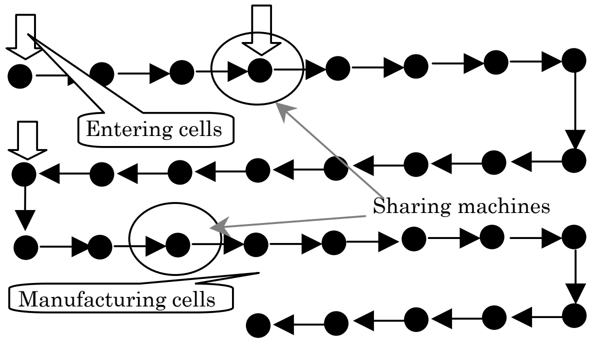


Fig.8 The routing view of the PCB production

Using virtual place introduced in previous section, it is very easy to deal with shared machines. Fig.9 models two manufacturing cells sharing three common machines.

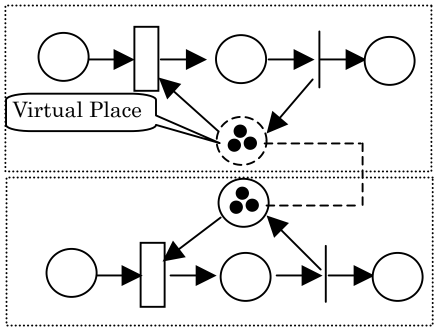


Fig.9 Virtual place and machine share

The RTCPN model of the above example includes 115 places, 29 times transitions, 29 instant transitions (time delay=0), and 174 arcs. 358 future jobs and 930 work-in-process jobs during two weeks are applied to demonstrate the performance of simulator. An original simulator, a specialized tool developed by C language, takes about 5~6 seconds to get final results on computing environment as show in Table 1. Because the original simulator cannot be expanded to deal with other factory and execute on other computing environment, a Java-built-in tool based on RTCPN is developed. The Java-built-in application takes about 34~35 seconds on the same computing environment.

Table 1 Time cost of simulation

Computing Environment	Language	Time Cost
Pentium 1.4G Linux	Standard C	5~6 s
Pentium 1.4G Linux	Java Jdk 1.4	34~35s
Pentium 1G Windows 2K	Java Jdk 1.4	50~52s

We hope to decrease the time cost of simulation to near that of the special simulator using C language in order to make Java-built-in application feasible to actual factory. The results in Table 1 are based on the algorithm given in Fig.4 without considering the block division. In fact, our proposed RTCPN model features that a transition may be fired several times at current time point because other firings may make it time-enabled. The most possible case is that the inner firings in a block may make transitions in the block time-enabled thus the time cost can be decreased if firing loop is firstly done within each block. This idea has been studied by G..Chiola [15], and the like works [16][17]. However, the results, given in Table 2, show that it just speeds up by about 10~14%.

Table 2 Time cost of simulation with block loop

Computing Environment	Language	Time Cost
Pentium 1.4G Linux	Standard C	5~6 s
Pentium 1.4G Linux	Java Jdk 1.4	30~31s
Pentium 1G Windows 2K	Java Jdk 1.4	45~46s

Upon investigation to simulation process, the time cost of simulation can be evaluated by

$$TimeCost = f(m, w, d, s, l) + a$$

where

f – A function related to simulator implementation,

m – Total firing times of transitions,

w – A coefficient related to scheduling method,

d – Average time for a firing process,

s – Average search time for determining next time-enabled transition,

l – Average time of processing calendar,

a – Pre-processing time for reading external information.

It is clear that all factors except for average search time are basically fixed unless we change the program structure. From viewpoint of algorithm, it is the average search time that make is possible to improve simulation performance. Return to algorithm shown in Fig.4, we should get next time point χ_{cur} for whole RTCPN model after no time-enabled transition exists. The inner loop in block can limit the search range thus decrease search time. The question is that the number of time-enabled transitions in inner loop is not so many enough to improve simulator performance greatly.

The mode in Fig.4 works in series along time axis that a transition cannot fire until it is time-enabled at current time point and only one current time point exists for entire simulation. It can be modified under some conditions so that an independent current time point is attached to each block, making simulating concurrent and ignoring other blocks' influence. It is easy to implement if the original system is completely flow-line type. From the first manufacturing cell to last one, simulating can be quickly carried out one by one because all jobs in current block become known once previous block has been simulated. Unfortunately, it is infeasible for the example in Fig.7 due to shared machines shown in Fig.9. Shared machines result in that a block may receive jobs from other blocks following current one. The jobs are not completely known before current block starts to runs. Thus for a generalized job-shop completely concurrent algorithm is almost impossible.

However, existence of buffers makes simulation concurrent a little possible. Fig.10 shows the maximum of blocking jobs in each buffer.

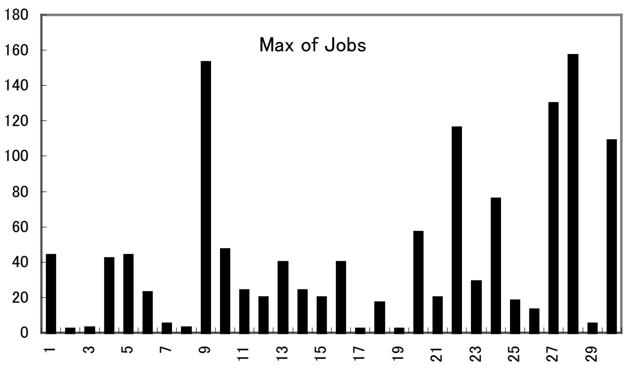


Fig.10 Maximum of blocking jobs in buffers

If too many jobs are blocked, it seems that the most of current jobs should be processed according to current jobs' sequence, ignoring jobs coming in the future. In generally obtained results in this way are approximate. Let the upper limitation be 50 jobs. About 6% of GANTT blocks are out of order compared to accurate result. But a problem is that about 74% of wrong results appear due to emergency jobs. Let the upper limitation be 100, the results almost don't change. Analysis of GANTT graph illustrates that emergency jobs have a great influence on old sequence thus cannot be ignored.

One of reasons for setting jobs emergency is to pass some manufacturing cells, referred to as bottlenecks, quickly. Obviously at a bottleneck cell, if emergency jobs exist, the processing sequence generally keeps until the last emergency job in current queue. The sequence of jobs following the last emergency job may become changeable due to possible coming of emergency jobs. A method to determine whether concurrent simulation at current block continues is predicting time duration for the next appearance of emergency job. For all future emergency jobs, we calculate the minimum duration θ by

$$\theta = Min\sum_{j=0}^{k}\tau_j \qquad (1)$$

where k stands for remaining operations until entering current block. Let χ' be the time point of the last emergency job. Then the time depth of concurrent process can be determined by following formula.

$$L = Max(\theta, \chi') \qquad (2)$$

Fig.11 gives a simulation algorithm based on concurrent process for each block. And improved results are given in Table 3.

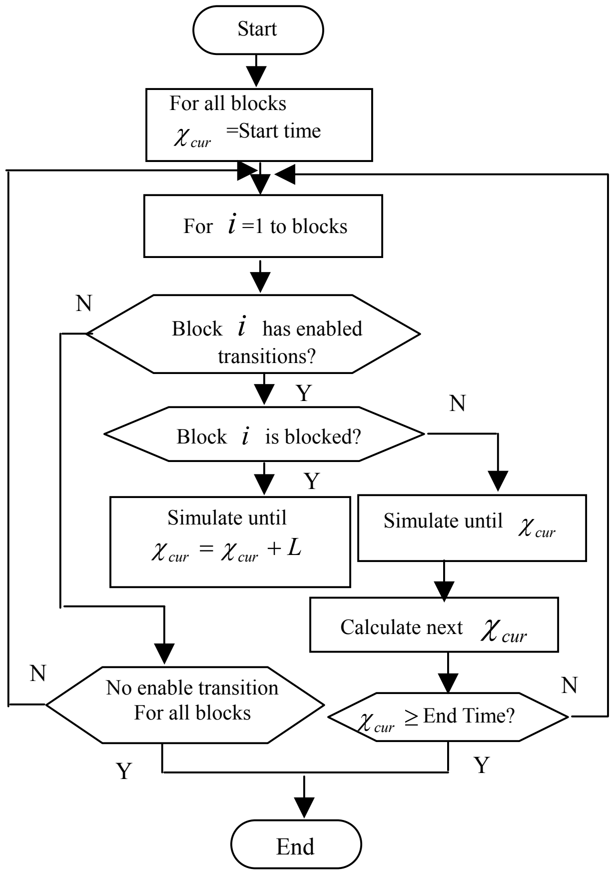


Fig.11 Simulation flow chart concerning concurrency

Table 3 Time cost of improved simulation

Computing Environment	Language	Time Cost
Pentium 1.4G Linux	Standard C	5~6 s
Pentium 1.4G Linux	Java Jdk 1.4	7~8s
Pentium 1G Windows 2K	Java Jdk 1.4	9~11s

The GANTT graph generated by algorithm depicted in Fig.11 has less wrong GANTT blocks than that without considering prediction of emergency jobs. Only about 1% of GANTT blocks lose their location and the errors of completed time of jobs are less than 2 hours compared to accurate results. Because of uncertainty in the future, generally the accurate simulation results are of approximation. The errors within reasonable range are allowed.

On the other hand, an independent application is often useless. An implementation of virtual factory should exchange information with other applications by network and database. Java application can make full use of compression technology to reduce data transferring time. The jobs and their related information given in above example need about 4.5 Mega bytes in text content while compressed file in ZIP format is just about 156k bytes. It will be of great benefit when a real-time simulator of virtual factory is applied over an intranet where other applications may access the same data resource simultaneously.

V CONCLUSION

Real-time Colored Petri net (RTCPN) based model, introducing concept of time-enabled transition, is applied to describe virtual factory. The architecture, implementation structure and simulation algorithm of virtual factory are given. A PCB production factory is used to demonstrate the feasibility of our Java-built-in application of virtual factory. The time cost of simulation is discussed and a concurrent algorithm is proposed to obtain approximate results with higher simulating performance and less errors.

VI REFERENCES

[1] Bodner, D. A. and S. Reveliotis. "Virtual Factories: An Object-Oriented, Simulation-Based Framework for Real-Time FMS Control," *Proceedings of the 1997 IEEE International Conference on Emerging Technologies and Factory Automation*, 1997, pp.208-213.

[2] Han-Pang Huang, C.F.Yeh, "Development of a virtual factory emulator based on three-tier architecture", *IEEE Intl. Conf. On Robotics and Automation*, Detroit, USA, MAY 10-15, 1999, pp.2434-2439.

[3] Bodner, D. A., M. Damrau, P. M. Griffin, L. F. McGinnis, A. McLaughlin, M. L. Spearman and C. Zhou. "Virtual Machine Models for Electronics Assembly," *Proceedings of the 1997 Deneb International Simulation Conference and Technology Showcase*, Troy, MI, pp. 61-66, 1997.

[4] Wolfgang Mueller-Wittig; Reginald Jegathese,etc, "Virtual Factory - Highly Interactive Visualisation for Manufacturing", *Winter Simulation Conference, WSC 2002,* San Diego, USA, 08-11 December, 2002

[5] Zhou, M. C., K. McDermott, and P. A. Patel**,** "Petri Net Synthesis and Analysis of a Flexible Manufacturing System Cell," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol.23 No.2, March/April 1993, pp.523-531**.**

[6] F. Balduzzi, A. Giua, C. Seatzu, "Modelling manufacturing systems with First-Order Hybrid Petri Nets, " *International Journal of Production Research, Special Issue on Modeling, Specification and Analysis of Manufacturing Systems,* Vol. 39, No. 2, January 2001, pp. 255-282

[7] Feldmann, K.; Colombo, A. W. "Monitoring of Flexible Production Systems Using High-Level Petri Nets", *Control Engineering Practice (CEP), Int. Journal of IFAC,* vol. 7, No.12, Pergamon Press Int. Dec.1999, pp.1449-1466.

[8] Jensen, K., *Colored Petri Nets: Basic Concepts, Analysis Methods, and Practical Use*, Berlin, Germany, Springer-Verlag, 1997.

[9] J. Peterson, "Petri net", *Computing surveys*, Vol.9, No.3, 1977, pp223-252.

[10] C. Petri, *Kommunikation mit Automation*, Ph.D. dissertation, University of Bonn, Boon, Germany, 1962.

[11] Murata, T., "Petri nets: properties, analysis and application", Proceedings of IEEE Vol.77, 1989, pp. 541-580.

[12] J.Wang, *Timed Petri Nets, Theory and Application*, Kluwer Academic Publishers 1998.

[13] J.E. Hong and D.H. Bae, "HOONets: Hierarchical object-oriented Petri nets for system modeling and analysis", KAIST Technical Report CS/TR-98-132, November 1998.

[14] Yoshiaki Shimizu and Keisuke Hiraide, "Timed Colored Petri Net Model for Analyzing Operating Procedures for Batch Processes", *Proc. of Int. Symposium on Process Systems Engineering* (PSE Asia 2000), PS-36, 2000,pp.279-284

[15] G. Chiola and A. Ferscha, "Distributed simulation of petri nets", *IEEE Parallel & Distributed Technology*, Vol.1, No.3, August 1993, pp. 33--50.

[16] D. M. Nicol and W. Mao, "Automated Parallelization of Timed Petri-net simulations", *Journal of Parallel and Distributed Computing*, Vol.29, No.1, Aug 1995.

[17] A. Ferscha, "Concurrent Execution of Timed Petri Nets", *Proceedings of the 199d Winter Simulation Conference*, 1994, pp. 229 - 236.