Proceedings of the 1999 IEEE
International Conference on Robotics & Automation
Detroit, Michigan • May 1999

Development of a Virtual Factory Emulator Based on Three-Tier Architecture

Han-Pang Huang♦, Chien-Fa Yeh*
Robotics Laboratory, Department of Mechanical Engineering
National Taiwan University, Taipei, TAIWAN 10674, R.O.C.
TEL/FAX: (886) 2-23633875
e-mail: hanpang@ccms.ntu.edu.tw
♦Professor and correspondence addressee *Graduate student

Abstract

Virtual factory (VF) can provide a total solution for shortening manufacturing cycle time and achieving customer satisfactions. Hence, many world-class companies dedicate themselves to establish their own virtual factories. However, it is difficult to implement a virtual factory because of the lack of assistant tools. A three-tier architecture based colored timed Petri net emulator is developed in this paper to provide such kind of tool. The three-tier architecture based emulator is easy to maintain and extend. This paper adopts UML and Microsoft COM/DCOM techniques to design and implement the emulator, respectively. The emulator is suitable to model complex systems, especially those with concurrent and asynchronous behaviors. In addition, this emulator provides distributed simulation ability. Each emulator can communicate with each other to achieve the concurrence among subnets. In particular, the proposed emulator can be integrated with WWW and voice system. Finally, the emulator is used to construct a flexible manufacturing system, and demonstrate its feasibility for virtual factory applications.

1. Introduction

Virtual factory has various definitions for different viewpoints and perspectives. One of the definitions is described as follows: a virtual factory is an environment that provides transparent descriptions and simulations of a real factory to users (internal or external) who are separated from the real entity in space and/or time via open and easy access and real time response to users' specific needs [1,7]. Inside a company a virtual factory integrates manufacturing resources with information, and in the outside world it provides satisfactory customer service.

In order to establish the virtual factory applications, such as scheduler, simulator and product cycle time estimator, the first step is to construct a model that is parallel (or concurrent) to a real factory. However, the lack of powerful modeling tools for complex and highly-integrated systems often blocks the development speed of a virtual factory. This paper focuses its scope on the development of an emulator that can real-time generate and simulate the system. The proposed emulator is a three-tier architecture, object oriented, distributed and colored-timed Petri net based virtual factory emulator.

The distributed and objected oriented features rely on COM (common object model) and DCOM (distributed COM). A COM object consists of one or more interfaces, and an interface consists of properties, methods, and/or events. On the other hand, DCOM can be considered as a network version of COM. DCOM has three new elements built on COM: the technology for building remote objects; protocols for calling remote objects' methods; security for accessing remote objects' data. An ActiveX EXE server is a typical application of DCOM. By using COM/DCOM technology, the communication difference between applications is encapsulated. Namely, a client application can be easily built and call a COM/DCOM server, regardless of local machines or remote machines.

Colored timed Petri nets (CTPNs) have been proven to be a powerful tool for modeling many systems, especially those with similar components and/or parallel processing [2]. The proposed communication places are used to relate macro transitions among Petri nets. There are four kinds of communication places, i.e. pitch-up, pitch-down, catch-up and catch-down places. Each pitch-up (or pitch-down) place in one Petri net has a matched catch-down (or catch-up) place in another Petri net. The state between the matched communication places is concurrent; i.e., when a token runs into or moves from a communication place, the same activity is occurred in its matched communication place. Petri nets with communication places not only have the hierarchical modeling ability but also have the distributed modeling ability. Namely, it is a hybrid architecture. In a critically hierarchical architecture, one layer is tightly related to its first upper layer and its first lower layer. However, in a hybrid architecture, one layer can be related to any other layers in a model. Because of the superior modeling ability, the Petri net with communication places is a powerful modeling tool for distributed emulator.

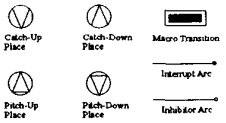


Fig. 1 Some extended places and transitions for Petri nets

Finally, the proposed three-tier architecture, object oriented, distributed and colored-timed Petri net-based virtual factory emulator is justified by a flexible manufacturing system.

2. Analysis and Design of Colored Timed Petri Net Emulator

A. Overview of System Architecture

A simulation process can be divided into three stages: modeling, simulation and simulation analysis. In modeling stage, a modeling tool with user friendly interface is required to construct and express a model for a real system. In simulation stage, a simulation engine is required to drive the simulation process. In this stage, the simulation performance should be considered. Finally, in simulation analysis stage, the simulation outputs have to include the event history, statistic lists and other useful information. UML (Unified Modeling Language) is used as the analyzing tool. The required output information in simulation analysis stage is always case by case.

In order to satisfy these requirements, the following elements are designed for an emulator in this paper:

1. Colored timed Petri nets with macro transitions and communication places are selected to be the

modeling tool.

2. A Petri net designer with user-friendly interface is designed to construct colored timed Petri net models. A Petri net viewer is designed to monitor and analyze the simulation conditions.
3. A Petri net engine is designed to execute the simulation process. The engine includes an interpreter to interpret and execute the codes in transition regions.
4. A Petri net simulation driver is designed to setup and call Petri net engine.
5. A communication component, which includes Windows sockets, is designed for the communication between communication places.
6. A relational database is designed to store and share Petri net models and markings.

An overview of the three-tier architecture is shown in Fig. 2.

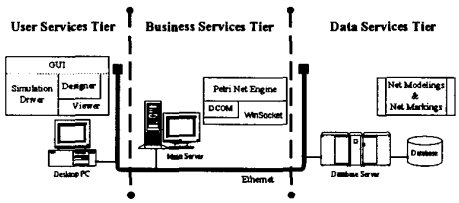


Fig. 2 Physical architecture of colored timed Petri net emulator

B. Petri Net Engine

Petri net engine for CTPN emulators exists in the middle tier of the three-tier architecture. It is designed as an ActiveX EXE server – PetriNetEng.exe. Since Petri net engine is a server component, the client application can use Petri net engine through DCOM technique. The main function of Petri net engine is to encapsulate the business policy of CTPNs, especially the firing rules. The following subsections will describe the design of Petri net engine.

(1) Class Design of Petri Net Engine Kernel

In order to manipulate operations over a CTPN model, Petri net engine has to know the whole architecture of the CTPN model firstly. Therefore, a number of classes are designed to display a CTPN model. A CTPN model, which is displayed by the CPetriNet class, mainly consists of places, transitions, arcs, colors, and tokens. The abstract classes CPlace and CTransition are designed to show the abstract operations of the Petri net elements. CNormalPlace, CPitchUpPlace, CPitchDownPlace, CCatchUpPlace, and CCatchDownPlace classes inherit the abstract class CPlace and show the concrete Place elements. Similarly, CNormalTransition, CImmediateTransition and CInhibitorTransition classes inherit the abstract class CTransition and show the concrete transition elements. The connections between places and transitions are presented by CInputPlace and COutputPlace classes, instead of using CArc class. A CInputPlace object shows an input place for a specific transition. In contrast, a COutputPlace object shows an output place for a specific transition. The arc type is denoted as an attribute within CInputPlace and COutputPlace classes. The residual classes used to display the architecture of a CTPN model are CToken and CColor classes. They are used to show the tokens and colors in a CTPN model respectively.

The CTableInfo class and RDO (remote data object) provide the function of connecting to a database. A CTableInfo object is used to store the database information, including the ODBC data source name, database name, user's identification and user's password. RDO is a commercial object developed by Microsoft Corporation. It provides the connection frame between program languages and ODBC database servers. Because RDO can access any database severs that provide ODBC driver, the connection between Petri net engine and database servers is much more flexible.

(2) Communication Mechanism

When a large Petri net is divided into several small Petri nets, the synchronization among Petri nets is provided by communication places. Actually the concurrence is arrived by sending messages between nets. To implement this idea, an ActiveX DLL server – PNComm.dll – is designed to support the communication between Petri nets.

A PNComm sever implements a CPNComm class. Each object of CPNComm class uses a fixed Windows socket to listen for an incoming connection for some ports. When the listening socket receives a connection request, a new socket is created to accept the request and prepare to get data. At the same time, the listening socket returns to listen for the next incoming connection. After the dynamically created socket has got all the data and stored the data into a buffer (an object of CDataBuffer class), it is destroyed immediately. Since the sockets used to get data are created dynamically, a Petri net engine with a CPNComm object can get data from several other Petri net engines concurrently. In addition to getting data, a CPNComm object also uses a fixed socket for sending data. Petri net engine can indirectly use this socket to send communication places' information, such as adding tokens to a communication place or removing tokens from a communication place.

Fig 3 shows the interactions between a CPetriNet object and a CPNComm object.

(3) Interpreter

There are three regions accompanying with a transition. Their functions are described in Fig. 4.

Guard Region: The codes in a guard region define the constraints of enabling a transition. Hence, a transition can only be enabled by the specific tokens whose color values can satisfy those constraints.

Action Region: The codes in an action region define the activities when a transition is firing. A general language, like meta language or C++, may be adopted to edit codes. Users can operate a global variable or send specific messages by coding this region. The action region can be further divided into three regions (Fig. 4):

1. Activation region: The codes in this region are executed immediately when a transition fires.
2. Continuous region: This region only resides in timed transitions. The codes in continuous region are continuously executed during the firing interval. If users need to send alarm messages continuously, or something else, they can edit an alarm function in this region.
3. Deactivation region: The codes in this region are executed immediately when the firing process terminates.

Deposit Region: The codes in a deposit region define how the color values are assigned to each output tokens.

In order to interpret and execute the codes in these regions. An interpreter is designed to accomplish this task. The interpreter can interpret all of the codes and restore them into a data structure before running simulation. Therefore, in simulation stage, the Petri net engine can execute the program codes efficiently.

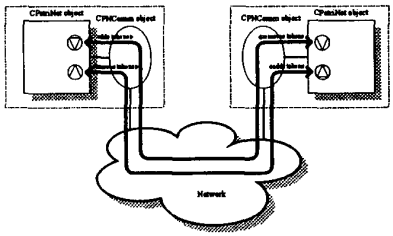


Fig. 3 The interactions between CPetriNet object and CPNComm object

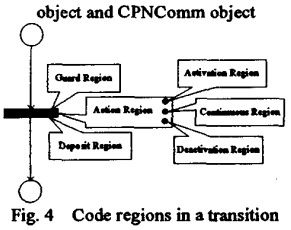


Fig. 4 Code regions in a transition

(4) Algorithm Design for Solving Conflicts

There are two different conflicts that may happen in a Petri net model. The conflict between tokens conflict of selecting resources or the conflict of selecting entities in a real system. The conflict between transitions means the conflict of selecting common resources or the conflict of selecting paths in a real system.

- For the conflict between tokens:

The policies for deciding which token should be selected often depend on tokens' attributes. The generally used policies are listed below:

- Decision by some color values: A color is an attribute in a real system. In the real world, an activity always happens only when some attributes are matched. For example, a job may only be worked on some kind of machines. This policy points out this special property evidently.
- Decision by token's priority: Two conditions are considered in this policy. One is 'the highest priority first' (HPF) and the other is 'the lowest priority first' (LPF). If we use HPF policy, the token with the highest priority is selected. On the contrary, if we use LPF policy, the token with the lowest priority is selected.
- Decision by the token's arrival time: Two conditions are considered in this policy. One is 'first in first out' (FIFO) and the other is 'first in last out' (FILO). If we use FIFO policy, the first arrival token is selected. On the contrary, if we use FILO policy, the last arrival token is selected.
- Random selection: If a system does not care which token is selected first, this policy can provide a good solution. This policy is often called 'service in random order' (SIRO).

- For the conflict between transitions:

The proposed methods for solving this conflict are described below:

- Decision by the total busy time: This is only suitable for timed transitions. The transition with the least total busy time is selected to fire. For example, if a transition represents a server, then the server that has the least working time is assigned to provide service for a customer.
- Decision by the time delay: Again, this policy is only suitable for timed transitions. By using this policy, the transition with the least time delay is selected to fire. For example, if a transition represents a machine, then the machine with the least process time (time delay) is selected to work.
- Decision by a router: This is especially suitable when simulating a flow-shop system. The token has a color to denote which transition should be selected to fire. On the other hand, the codes in a transition's guard region define what color values can make this transition fire.
- Decision by looking forward n steps: When using this policy, the transition, which can gain the maximum benefit in the next n steps, is selected to fire.

Since the conflict between tokens always happens in a place, it is easy to solve by adding a queue type to places. A Q-algorithm [8] for detecting if a transition is enabled can help us to know which transitions are conflictive. If a transition is enabled, it gives each enabling token an order. Therefore, after applying this algorithm to each transition, we can know which transitions are conflictive by detecting the ordering queue of tokens. An F-algorithm used to decide which transitions can fire is also developed in [8]

C. Petri Net Designer and Petri Net Viewer

The Petri net designer provides a graphical user interface to construct Petri net model directly. Users can use the designer to set places, transitions and arcs on a form, and edit their attributes. On the other hand, the Petri net viewer provides a number of classes to draw the Petri net elements. Though Petri net designer and Petri net viewer are logically separate, they are implemented in the same way. The Petri net designer and Petri net viewer shares the same classes to express a Petri net model. The classes designed in the Petri net viewer and Petri net designer is similar to those classes designed in Petri net engine. The main displays of Petri net designer are shown in Fig. 5.

D. Design of Database Tables

To reuse and share Petri net models, the models have to be stored persistently. A relational database is designed to provide this service. The database has to store not only the static architecture but also the dynamic behavior setting of Petri nets. The definitions of each database table can be found in [8]. It should be noted that the term MACRO means a subnet; i.e., it represents a subsystem. In addition, the term PETRINET means the collection of MACROs; i.e., it represents the whole system.

E. ActiveX Control for Voice System

Manufacturing execution system (MES) is the kernel of shop floor control systems [3]. The architecture of MES is demonstrated in Fig.6. MES manages the activities and dispatches resources in a factory. Its objective is to make a factory operate regularly and then achieve the production tasks.

Virtual manufacturing system (VMS) is concurrent to the physical manufacturing system (PMS). It reflects the factory's states to the supervisor and monitor. Based on monitoring of the responding states, the supervisor can corporate the factory's operation activities. On the other hand, the monitor can detect if any machines work irregularly. If the monitor finds some problems on the machines, then the troubleshooter is called to solve the problem. The troubleshooter tries to find the solution from the knowledge base first. If a solution is found, then the knowledge is applied to solve the problem, otherwise maintenance engineer is called on duty.

A number of researches had applied Petri nets to design the supervisor, monitor and troubleshooter [2,4]. In this paper, a voice system is further developed to enhance the notification ability of the troubleshooter. Traditionally, the engineers have to spend a lot of effort

to find out troubles by monitoring or querying the control system. The process is time-consuming. With the assistance of voice system, the troubleshooter can notify the engineers actively.

The voice system can dial automatically and record the acceptor's message to a mailbox if necessary. An ActiveX control for playing sound and controlling RS-232 is designed to help implement the voice system.

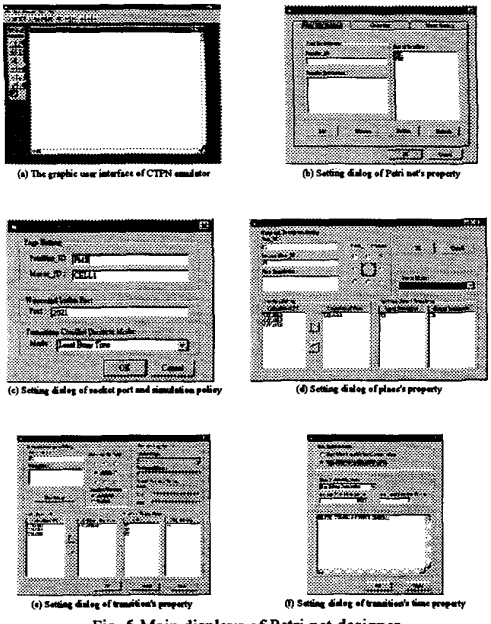


Fig. 5 Main displays of Petri net designer

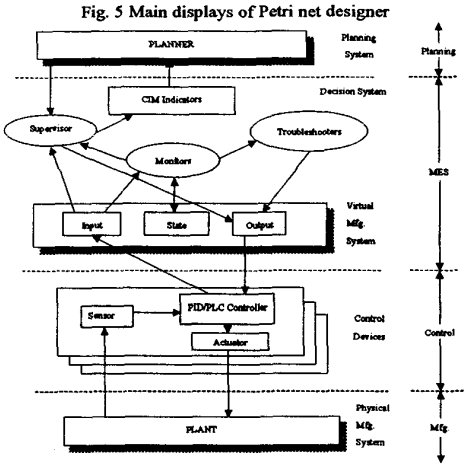


Fig.6 Architecture of MES

3. Integration of WWW and CTPN Emulator

The WWW is also powerful for virtual factories. For the potential buyers, a company can post their catalog and work-sheet applets on its Web site to allow customers to examine and simulate the functions of its products before buying them. For the customers who have ordered the products, a company can provide each customer an account and a password to allow them to query the production status about their ordered products via the Web server. These are the functions of customer service system. Inside a company, the manufacturing information, such as part location and machine states, can be posted on the Web site to allow the operators to search for the parts and the engineers to monitor the factory's operation state. These applications make WWW an immensely powerful tool for virtual factories.

Since CTPN emulator is a powerful modeling tool for virtual factories, the idea of transferring it into WWW version is advised. Two techniques can be used to implement the idea. One is Sun's Java applet, the other is Microsoft's ActiveX document. In this paper, Microsoft's ActiveX document is adopted to implement the WWW version of CTPN emulator. The transfer from desktop version to WWW version is quite easy via using Microsoft's ActiveX document technique, especially for those programs based on a three-tier architecture. The business services tier and the data services tier can be continuously used without any modification. The programmer has to only rewrite the graphical user interface with ActiveX document if the classes are well designed in advance. After completing these jobs, the CTPN emulator can be exposed on a Web page.

As mentioned before, when the Web users try to explore the Web page of CTPN emulator, the browser need not to download the whole program and install it. The browser only grabs the necessary components.

The related files for WWW version of CTPN emulator are shown in Fig. 7. Fig. 8 gives its graphical user interface.

4. Case Study

A. Case Problem

This case study will discuss the scheduling problem of an FMS (flexible manufacturing system). The scheduling problem is very difficult because an FMS allows different combinations of batches and machines to manufacture different kinds of products. Petri nets and colored timed Petri nets have been used to solve real-time scheduling and job dispatching problem in an FMS [2,6,9]. Those researches have proven Petri nets and colored timed Petri nets are good at dealing with scheduling problem.

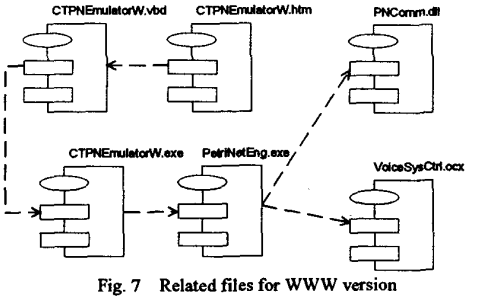


Fig. 7 Related files for WWW version of CTPN emulator

The scheduler is one of applications of a virtual factory. The discussed FMS has three cells and one AGV, shown as Fig.9. Six kinds of products are processed in this FMS and each product has its own routing. There are one robot, two buffers and three machines in Cell 1. All of machines in Cell 1 are precisely the same and they can process each kind of products. In Cell 2, there are one robot, three buffers and three machines. M4 can process Prod.1 ~ Prod.3; M5 can process Prod.4 ~ Prod.6; and M6 can process all kinds of products. Prod.1 to Prod.3 wait on Buffer3 for processing and, in contrast, Prod.4 ~ Prod.6 wait on Buffer4. Finally, Cell 3 has one robot, two buffers and two machines. Both M7 and M8 are batch-run machines with 2-capacity and they can process all kinds of products. In this case, the planning of robots' moving path is ignored. The relevant researches about the

trajectory planning of robots can be found in [3]. The transportation among cells relies on the AGV. The machines have different time specifications for each product and the AGV has different time specifications for each route among cells.

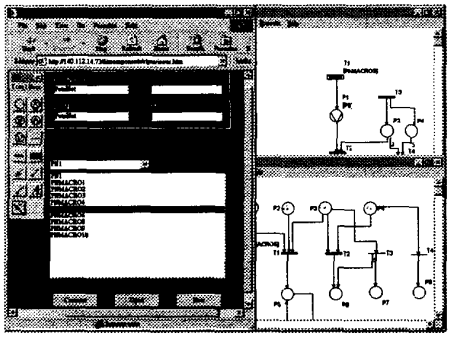
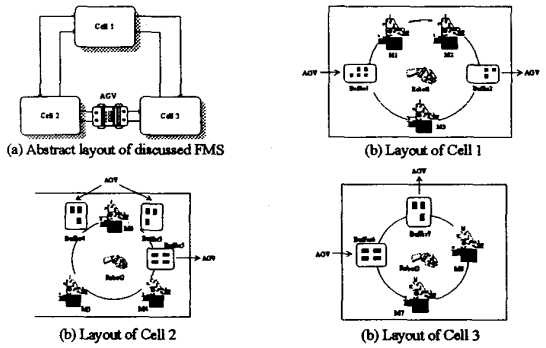


Fig. 8 WWW version for CTPN emulator

B. CTPN Models of Discussed FMS

From the view of MES, a real-time scheduler is a part of decision system. In general, the factory model and the factory's real-time information in VMS are used to decide the schedule. Therefore, a model of the studied FMS should be built first. The CTPN model of SCHEDULER is shown in Fig.11. The detailed routing information of SCHEDULER is encapsulated in transitions' deposit-code regions. Similarly, the different operation times of machines and AVG are described in transitions' time-code region. The hierarchical architecture of these CTPN models is demonstrated in Fig. 10. Table 2 shows an example of CTPN properties for each cell can be found in [8].



(a) Abstract layout of discussed FMS

(b) Layout of Cell 1

(b) Layout of Cell 2

(b) Layout of Cell 3

Fig. 9 Layout of FMS

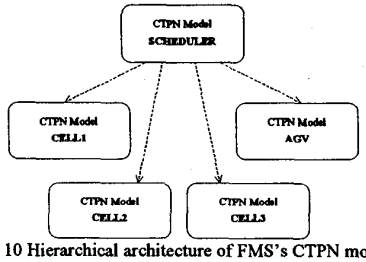


Fig. 10 Hierarchical architecture of FMS's CTPN models

C. Simulation Results

Since the CTPN emulator developed in this paper provides several policies to solve conflicts between transitions and between tokens, the scheduler can use these policies as dispatching rules. When a system is modeled as a combination of several subnets, like this case study, each subnet can apply one kind of policy to dispatch jobs. This is more superior to other simulation software. In addition, a distributed CTPN emulator provides an additional benefit, i.e. the network bandwidth and communication time between computers can be certainly examined. Furthermore, if users want to get more detailed information, such as the peak queue length, they can use suitable time distribution to simulate the system.

In order to synchronize the time base of each subnets, a time-setting server is designed to provide the reference time. The computer, which runs this server, can send its system time to other computers. Hence, the CTPN emulator can modify the computer's system time to the reference time.

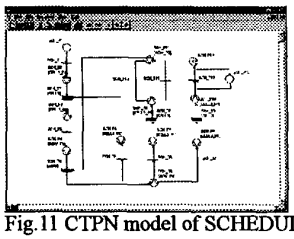


Fig.11 CTPN model of SCHEDULER

In the simulation, the queue type of all places is assumed to be FIFO. The time distribution mode of all timed transitions is deterministic. The system has twelve parts, two for each product, waiting for processing. The parts entering the system follow the order: PROD1 -> PROD2 -> PROD3 -> PROD4 -> PROD5 -> PROD6 -> PROD1 -> PROD2 -> PROD3 -> PROD4 -> PROD5 -> PROD6. The diagram of distributed simulation is shown in Fig. 12. Now, three different combinations of dispatching rules for each subnet are used to demonstrate the simulation functionality of the CTPN emulator. These combinations of dispatching rules are listed in Table 3. In Case 1, each subnet uses Random rule to solve the conflict of parts' path. In Case 2, AGV and SCHEDULER subnets use Random rule and the others use LPT rule to solve the conflict of parts' path. In Case 3, AGV and SCHEDULER subnets use Random rule, CELL1 and CELL3 subnets use LBT rule, and CELL2 use LPT rule to solve the conflict of parts' path. Since the queue type of each place is FIFO, the conflicts among resources and among parts are solved by the FIFO rule concededly.

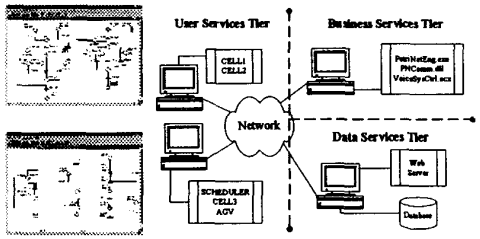


Fig. 12 Diagram of distributed simulation

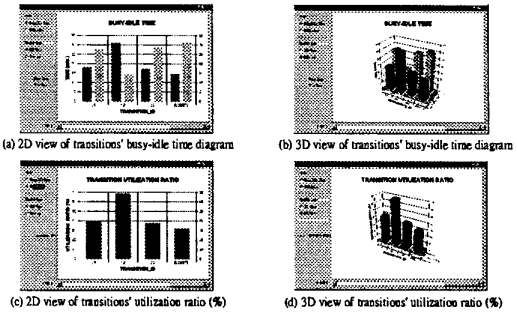
The CTPN emulator provides users both output files and graphical display to examine the simulation actions. The CTPN emulator also provides users a dialog to examine transitions' busy time, idle time and utilization ratio. An example is shown in Fig.13.

The simulation results are summarized in Table 4. From the results, we can generate the most suitable dispatching for tools. A Gantt chart of machines for Case1 is shown in Fig. 14.

Table 1 Dispatching rules for each simulation case

Net Case	CELL1	CELL2	CELL3	AGV	SCHEDULER
Case1	Random	Random	Random	Random	Random
Case2	LPT	LPT	LPT	Random	Random
Case3	LBT	LPT	LBT	Random	Random

Random: Random Selection, LPT:Least Process Time, LBT: Least Busy Time



(a) 2D view of transitions' busy-idle time diagram (b) 3D view of transitions' busy-idle time diagram

(c) 2D view of transitions' utilization ratio (%) (d) 3D view of transitions' utilization ratio (%)

Fig. 13 Busy time, idle time and utilization ratio diagrams

Table 2 Simulation results

(a) Total processing time

Case	Case1	Case2	Case3
Time	44'44"	44'18'	46'43"

(b) Departing order of parts

Case	Order
Case1	1, 4, 5, 6, 1, 2, 6, 5, 4, 3, 2, 3
Case2	1, 4, 5, 1, 6, 2, 4, 5, 6, 3, 2, 3
Case3	1, 4, 5, 6, 1, 4, 2, 2, 3, 5, 6, 3

(c) Utilization ratio of tools (unit: %)

Machines:

Case	M1	M2	M3	M4	M5	M6	M7	M8
Case1	39.8	68.5	37.6	60.95	47.65	57.6	42.0	35.4
Case2	64.85	40.3	42.5	58.1	48.1	62.6	41.7	37.2
Case3	44.5	50.8	44.5	55.1	50.9	55.1	39.4	33.0

Robots:

Case	Robot1	Robot2	Robot3
Case1	32.2	23.47	25.5
Case2	32.5	23.7	25.7
Case3	30.8	22.5	24.4

AGV:

Case	AGV
Case1	41.85
Case2	36.7
Case3	40.94

5. Conclusion

In this paper, a three-tier architecture based colored timed Petri net emulator is proposed and implemented. Such an environment can help users construct and simulate a virtual factory easily. The functions of the proposed emulator have been proven in the scheduling problem of an FMS. The proposed emulator adopts the concepts of stochastic timed transitions, macro transitions and communication places. These additional Petri net elements provide several powerful capabilities. Stochastic timed transitions help us to model and simulate systems' timing behaviors reasonably. Macro transitions help us to model a complex system by several smaller subnets and communication places provide the communication ability between these subnets to achieve the concurrence. In addition, the Q-algorithm and F-algorithm are proposed to solve the resource conflict and the path conflict problems for run-time detecting and solving the conflicts between transitions and between tokens. The emulator was integrated with WWW and voice system. Since the proposed emulator is implemented on the basis of three-tier architecture and object-oriented method, it can be easily expanded and maintained. However, some other features may be added to the proposed emulator, such as develop a transaction server to manage ActiveX servers and achieve the fault tolerance, develop more ActiveX servers for various applications, and consider event-driven mode.

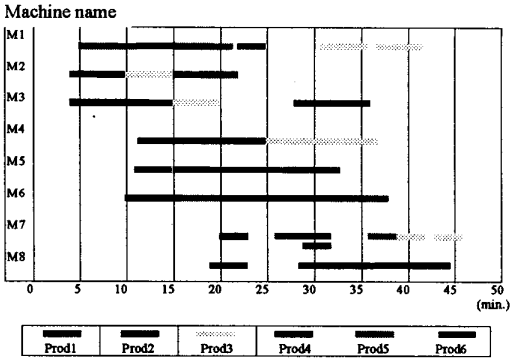


Fig. 14 Gantt chart of machines for Case1

References

[1] R.S. Fuo, Y.H. Su, S.C. Chang, Y.W. Lee, T.L. Chou, "Real Competitiveness via Virtual Fab," Proceedings of the 1997 National Conference on Management of Technology, pp. 156-163, 1997.

[2] C.H. Kuo and H.P. Huang, "Colored Timed Petri Net Based Statistical Process Control and Fault Diagnosis to Flexible Manufacturing Systems," IEEE Int. Conf. on Robotics and Automation, New Mexico, Vol. 4, pp. 2741-2746, April, 1997.

[3] G.R. Liang, "Fault Recovery in Automated Manufacturing System Using Extended HTM," Proceedings of the Asia-Pacific Conference and CIIE National Conference, pp. 591-596, 1994.

[4] Y.H. Liu, "Multi-user Remote Control of Flexible Conveyor System Using Virtual Manufacturing Devices," Master Thesis, Department of Industrial Engineering, National Chiao Tung University, 1995.

[5] S.P. Sethi, C. Sriskandarajah, G. Sorger, J. Blazewicz, W. Kubiak, "Sequencing of Parts and Robot Moves in a Robotic Cell," The International Journal of Flexible Manufacturing Systems, pp. 331-358, 1992.

[6] C.T. Shen, "Automatic Petri-Net Generator for Modeling and Scheduling of Flexible Manufacturing Cell," Master Thesis, Department of Information Engineering, National Taiwan University, 1995.

[7] D.M. Upton and A. McAfee, "The Real Virtual Factory," Harvard Business Review, pp. 123-133, July-August, 1996.

[8] C.F. Yeh, "Development of Colored-Timed Petri Net Emulator based on Three-Tier Architecture," Master Thesis, Department of Mechanical Engineering, National Taiwan University, 1998.

[9] M.C. Yeh, "Development of Shop Floor Control System for Flexible Manufacturing Systems," Master Thesis, Department of Mechanical Engineering, National Taiwan University, 1995.