JAVA/CORBA Implementation of Petri Nets for Control of Manufacturing Systems

Choy Min Chee
Department of Electrical Engineering
National University of Singapore
10 Kent Ridge Crescent, Singapore 119260
Tel: (65) 874 4857
E-mail: eng70390@ nus.edu.sg

ABSTRACT

The intense competition present in the manufacturing industry has renewed interest in the issues of increasing productivity through state-of-art manufacturing technology. Gaining technological edge requires the harnessing of innovative concepts and original ideas in solving complex manufacturing planning and control problems. Flexible manufacturing system (FMS) is a concept that has evolved since the 1970s that attempts to achieve the efficiency of automated, high-volume mass production while retaining the flexibility of low-volume job-shop production. Petri Net is a mathematical modeling concept that is suitable for portrayal of discrete event dynamic systems. As such, the Petri Nets concept has been adopted in this project to model, monitor and control the different modules of a FMS in a real time manner. Each module of the FMS is designed according to the Petri Nets model and run in an independent, generic application written entirely in Java. The synchronization and communication between these modules are achieved using the Common Object Request Broker architecture (CORBA). Jobs created based on the master production schedule (MPS) generated by the existing Material Resource Planning (MRP) module are dispatched into the factory network where they are processed and routed until completion. Real-time scheduling *using CEXSPT is performed on the jobs as they are routed through the factory network. In addition, the MPS is dynamically modified according to the finished jobs' status.*

1 INTRODUCTION

In recent years, fluctuating market demand for manufacturing products, shorter lead-time, smaller but more frequent order quantities are some factors which makes forecasting of demand for a manufacturing product extremely challenging. The concept of Material Requirement Planning (MRP) in an assemble-to-order environment aims to alleviate the problem. In the assemble-to-order environment, the manufacturing system is required to possess a high degree of flexibility and some forms of real time control over the production schedule. As such, the flexible manufacturing system concept together with Petri Net are used in this project to provide modeling, control and simulation of manufacturing systems in the assemble-to-order environment.

The flexible manufacturing system is a concept which aims to accomplish the benefits of the job shop and the flow shop by achieving high volume of production in a flexible manner. The figure below shows a typical model of a flexible manufacturing system. This model will be used for the simulation of the factory scenario.

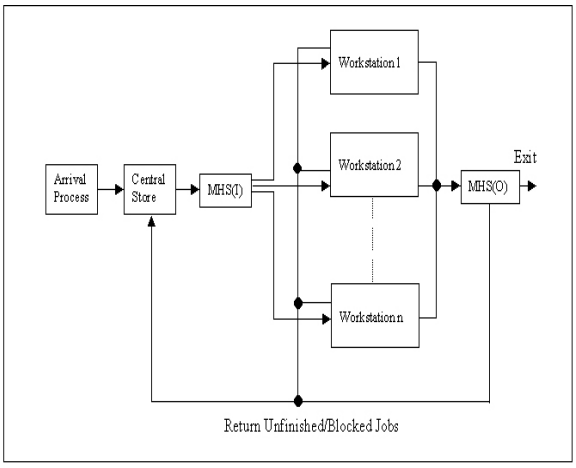

Fig. 1.1 A Typical FMS model

By characterizing the FMS as a discrete-event system, the real time controller is designed using Petri Nets, a mathematical tool that is suitable for modeling discrete event dynamic systems.

2 PETRI NET

Petri nets have an origin dating back to 1962, when Carl Adam Petri wrote his PhD thesis on the subject. Since that time, Petri nets have been accepted as a powerful formal specification tool for a variety of systems including concurrent, distributed, asynchronous, parallel, deterministic and non-deterministic. A manufacturing system can be viewed as a sequence of discrete events. Manufacturing operations that take place simultaneously represent the concurrent characteristic of a dynamic system. In addition, there are asynchronous operations and event driven operations, which can result in situations such as deadlock and conflict. Petri Net is chosen for analysing such a discrete event dynamic system and to provide models for analysis, simulation and control of manufacturing systems. An ordinary or simple Petri Net is a bipartite graph that can be represented in various mathematical forms.

The equation below is the quintuple representation of a Petri Net with n Places and m Transitions chosen for its completeness:

$$PN = (P, T, F, W, M)$$

- $P = \{ p_1,\ldots,p_n\}$ is a finite set of Places.
- $T = \{ t_1,\ldots,t_m\}$ is a finite set of Transitions.
- F represents the flow relation, or a set of Arcs. It comprises of the set of directed arcs from P to T ($P \times T$, input arcs) and the set of directed Arcs from T to P ($T \times P$, output arcs). Mathematically,

$$F \subset (P \times T) \cup (T \times P)$$

- W represents the weight assigned to each Arc. This attribute determines the number of Tokens needed to fire a Transition as well as the number of Tokens that are released by a Transition.
- M , the marking of the Petri Net, is a n-dimensional vector whose I^{th} component $M(p_I)$ represents the number of Tokens in the I^{th} Place, p_i. Hence, the initial distribution of tokens in a Petri Net model is denoted by M_0. Subsequently, the marking of the Petri Net can be changed when Transitions fire or after Tokens are released from Transitions.

In order to enhance the factory simulation, the Petri Nets components in this project are further categorized and given new functionalities while adhering to the general rules of General Stochastic Petri Nets. Transitions are categorized into Dispatcher-type, Process-type, Exit-type, Return/Blocked-type and Return/Finished-type. In addition, the time delay between the two events that separate the firing of a Transition is classified into immediate-type, deterministic-type and exponentially distributed-type. Places are categorized into Buffer-type, Resource-type and Arrival-type. Each Place of Buffer-type or Arrival-type can perform Conditionally Expedited Shortest Processing Time (CEXSPT) scheduling on the jobs in its queue. Lastly, Tokens are categorized into Resource-type and Job-type.

3 CORBA ARCHITECTURE

CORBA (Common Object Request Broker Architecture) is a specification of an architecture and interface that allows an application to make request of objects (servers) in a transparent, independent manner, regardless of platform, operating system or locale considerations. This specification was adopted by the Object Management Group (OMG) to address the complexity and high cost of developing distributed object applications. The CORBA paradigm follows two existing methodologies, distributed client-server programming and object-oriented programming and it is used in this project to facilitate communication and synchronization among the distributed FMS modules. The following figure shows the object request broker (ORB) of CORBA with reference to [3].

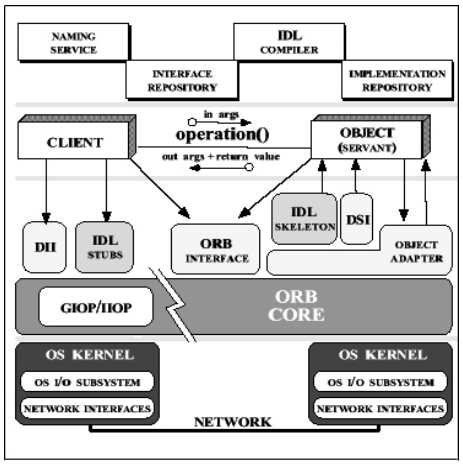

Fig. 3.1 The CORBA ORB Architecture

The client requests a service from the object implementation. The ORB transports the request that invokes the method using object adapters and the IDL skeleton. The client has an object reference (refer to the next section), an operation name and a set of parameters for the object and activates operations on this object. The Object Management Group / Object Model defines each operation to be associated with a controlling parameter, implemented in CORBA as an object reference. The client does not know the location of the object or any of the implementation details. The request is handled by the ORB, which must locate the target object and route the request to that object. It is also responsible for getting results back to the client.

The request makes use of either the dynamic invocation (dynamic link) or the IDL Stubs (static link) interface. Static links use the IDL stubs (as local function calls) and dynamic requests use the DII. The object implementation is not aware of the difference. If the interface was defined in the IDL and the client has an interface for the target object, the IDL stub is used. This stub is specific to the target object. The DII is used when the interface is not known at runtime. The DII uses information stored in the interface repository to establish the request. Requests are passed from the ORB to the object implementation through the IDL skeleton. The object implementation is made available by using information stored in the implementation repository. The object implementations encapsulate state and behaviors of the object. CORBA only defines the mechanisms to invoke operation, it does not define how activated or stopped, made to persist, etc. The mapping of the object reference to the object implementation is provided by a primary ORB service known as the object adapter. The following section describes the object adapter with reference to the object adapter that is used in this project, the Portable Object Adapter (POA).

3.1 Portable Object Adapter

An object adapter (OA) serves as mediator between the ORB and the servant, providing the ORB with a consistent interface for interacting with user code, and being flexible in cooperating with the servant. Different object adapters could then be provided to suit the various server requirements, and an implementation could choose between them and select the most appropriate. OA have a public interface that is used by the object implementation and a private interface that is used by the IDL skeleton.

Unlike its predecessor the BOA, the Portable Object Adapter is not a singleton component. Many POA instances can exist in a server organized in a hierarchical structure. The Root POA is created by the ORB, new POAs can then be created by the user as children of existing ones. Each POA maintains its own Active Object Map, a table mapping the currently active objects to servants (entities for servicing objects). Objects are activated within a particular POA instance, and henceforth associated with their POA, identified by a unique Object Id within its namespace. Synchronization between POAs is achieved by POA Managers, which control the readiness of one or more POAs to receive requests. Apart from having control over synchronization, the POA provides many hooks that enable a user to influence request processing:

- The life cycle of servants can be controlled and monitored by Servant Managers.
- Default Servants can service many objects at once.
- Adapter Activators can be used to create new POAs if necessary.

4 DESIGN AND IMPLEMENTION

The Petri Nets Simulator is a Java-based application for modeling, control and simulation of FMS modules using Petri Nets. The methodology used in developing such a complex system is to decompose and modularize the overall system using the object-oriented paradigm. Since Petri Nets are used to model the flexible manufacturing system, its various components such as Place, Transition and Tokens are developed as generic Java objects in the Petri Nets Simulator and they serve as the building blocks for the factory model. Each instance of the Petri Nets Simulator provides the avenue for designing and hosting of one FMS module.

In this project, separate instances of the Petri Nets Simulator representing the different FMS modules as shown in Fig. 1.1 reside in several computers in a distributed manner. These computers can belong to the same subnet or different subnets but all of them will form the factory network. Communication among distributed modules is achieved using the Common Object Request Broker Architecture and each of them can act as client and server concurrently using the Java multi-threading feature. From the customers' demands, periodical forecasts and the various attributes which are factored into MRP module e.g. Safety Stocks, the MPS of several products are generated dynamically and stored in different tables of the Oracle database. Based on the MPS of the products, job batches representing the various bills of materials for the products are created by the Job Generator module. Following that, these jobs are sent to the factory network by calling the job creation method available in the remote CORBA object which is part of the central storage module of the factory network. Jobs are routed through the network till completion and real time scheduling is performed throughout the process. Upon completion, the status of the jobs will be feedback to the Job Generator module which also provides the feature for updating the common database. The database is also made available to the front-end manufacturing portal. The figure below shows the overall system diagram of the factory network.

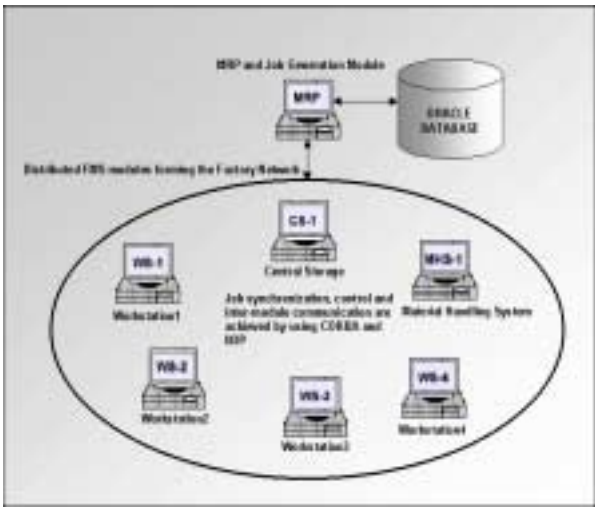

Fig. 4.1 Overall System Diagram

The context diagram of a single FMS module is as shown below.

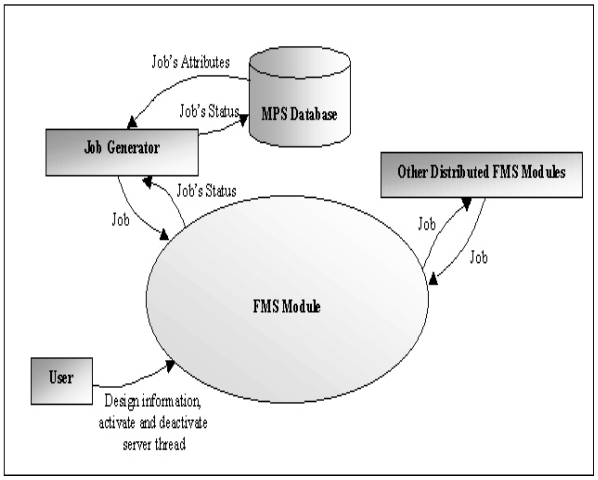

Fig. 4.2 Context Diagram of a FMS module

As can be seen from the context diagram, in order to achieve data integrity and to systematically monitor database access, the FMS module does not have read or write data into the Oracle database. Instead, the Job Generator module obtains the jobs' attributes from the database based on the MPS of that product and generate jobs that are sent to the factory network using methods implemented in the CORBA objects. During the designing and abstraction of the Petri Nets Simulator, 11 main classes are identified and the partial view of the class diagram in UML is as shown below.

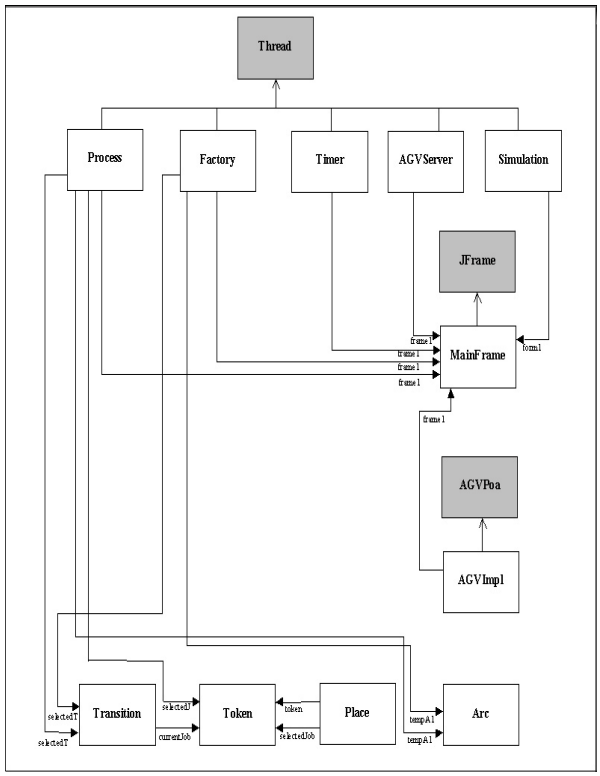

Fig. 4.3 Partial Class Diagram of the Petri Nets Simulator

There are 3 main categories of object classes. The Petri Nets Component Classes are abstractions of components of Petri Nets. The Design and GUI classes provide a user-friendly environment for designing generic Petri Net models and testing the models' attributes based on the various Petri Net's properties. Lastly, the Factory classes comprises of threads for the simulation of factory scenario as well as the implementation of the CORBA object whose methods can be called by other distributed FMS modules.

Visibroker ORB which comes with Inprise Application Server 4.0 is used as the CORBA ORB and its dynamic directory service Osagent is used as the implementation repository. The Osagent provides facilities which are used by both the client and the object implementations. It is used in this project in substitution to the Naming Service due to its simplified API that eases the complexity of discovering distributed objects. Moreover, Osagent provides support for fault tolerance and it has a in-built Round Robin load-balancing algorithm.

5 DEPLOYMENT AND SIMULATION

The factory network that is simulated in this project comprises of 6 FMS modules designed according to the FMS structure shown in Fig. 1.1 while aligning to the concepts of FMS given in [1] and [2]. Besides the central storage CS-1 and the material handling system MHS-1, there are 4 workstations identified by WS-1, WS-2, WS-3 and WS-4. The actual deployment architecture of the factory network is as shown below.

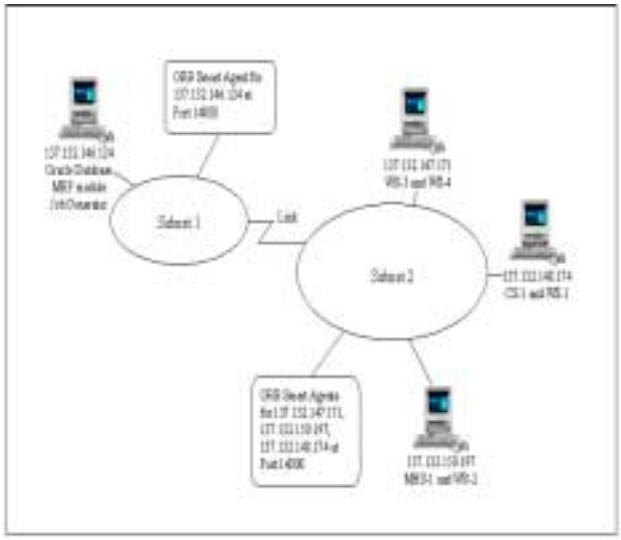

Fig. 5.1 Deployment Architecture of the Factory Network

One Osagent is started on each terminal to provide better fault-tolerance. After the invocation of the Petri Nets Simulator in the different distributed terminals, the Petri Nets design of each FMS module is loaded into the application. Following which, the object implementations of these modules and their POA names are registered into the Osagents by clicking the button found at the MenuBar to activate the Factory thread. After this is done, the factory network is ready to receive jobs from the Job Generator. New job batches will arrive at CS-1 or the Central Storage facility for the FMS. Following which, material handling process is performed before the job is dispatched to its designated workstation. After processing, the job will be routed to the MHS-1 for material handling process. If the job has finished its process sequence, it will exit the system. The job ID of the finished job will also be sent back to the Job Generator module where the MPS in the Oracle database is updated. Otherwise, the job is routed back to CS-1 for futher dispatching.

Note that there is no startup sequence for each of the FMS modules. New FMS module can join the factory network at any time by activating its Factory thread and designated

jobs will be routed to it subsequently. This ensures the overall scalability of the factory network.
For the purpose of this simulation, new batch of jobs are made to arrive only after the previous batch have finished processing. This is to ease the monitoring of the jobs' status across the distributed terminals in order to obtain simulation results for analysis. Altogether, 4 types of simulation are carried out: short process simulation with CEXSPT scheduling, short process simulation without CEXSPT scheduling, long process simulation with CEXSPT and long process simulation without CEXSPT scheduling. The figure below is the screenshot showing the arrival of a new job batch at the CS-1.

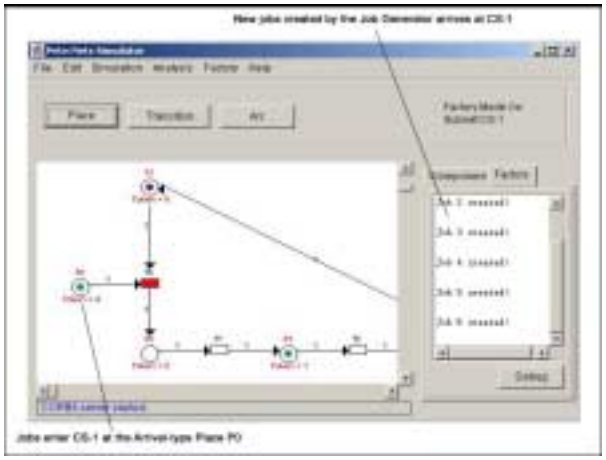

Fig. 5.2 Jobs Arrival at CS-1

The figure below shows MHS-1 with the finished status of the jobs.

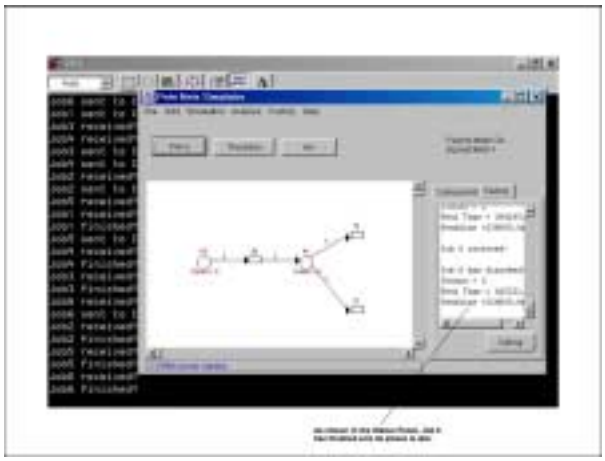

Fig. 5.3 Jobs Finished at MHS-1

From the simulation, it is found that the effect of CEXSPT is not obvious when the job processing times are small in the short process simulation. On the other hand, when the job processing times are extended in the long process simulation, using CEXSPT for real-time scheduling does result in fewer late jobs as compared to FIFO. Therefore, it can be concluded that the Petri Net modeling technique and the presence of CEXSPT for real-time scheduling are crucial factors affecting the performance of the factory network. The following tables show the results of the long process simulation conducted for a batch of 6 jobs.

Job ID	Status	Actual Processing Time (msec)
1	Late	262,049
2	Late	298,975
3	Early	330,002
4	Late	253,312
5	Late	344,207
6	Late	359,569

Table 5.1 Long Process Simulation without CEXSPT

Job ID	Status	Actual Processing Time (msec)
1	Operationally Late	212,456
2	Early	248,310
3	Operationally Late	277,130
4	Late	248,656
5	Late	226,745
6	Operationally Late	258,873

Table 5.2 Long Process Simulation with CEXSPT

In addition, through the use of CORBA, communication and synchronization between distributed objects in the FMS modules have been achieved across different platforms seamlessly during the simulation.

6 DISCUSSION

There are various issues that cause limitations or problems affecting the performance of the Petri Nets Simulator as a real-time controller. These issues are either present in a FMS module or the inter-module communication mechanism. Firstly, since Java multithreading is used extensively in the Petri Nets Simulator for implementation of concurrent processing, the factory simulation and real-time control within the FMS module are affected by the issues of thread synchronization, memory consumption by threads and unpredictability of thread scheduling and processing.
Secondly, the synchronous nature of CORBA operation invocations and the relatively long duration of remote calls make the CORBA-based application particularly vulnerable to the threat of a blocking GUI and thus affect the inter-module communication mechanism. This is make worse by the non-deterministic nature of the server latency.
Finally, the application of Petri Nets to certain real world sizable systems is questionable due to the state explosions problems encountered in the modeling and design stages. This can be inferred from the various case studies that are

presented in [1] and [2]. However, this problem is solved in the project using OOD and distributed objects.

7 CONCLUSION

Using the Petri Nets Simulator, the simulation of the factory scenario has been successfully carried out in the Laboratory of Concurrent Engineering and Logistic (LCEL). The Petri Nets Simulator has achieved scalability through the use of OOD and CORBA. The mode of operation is decentralized and each FMS modules in the simulation are proven to be capable of performing independent job routing and real-time scheduling. Through intelligent job routing, jobs are not discarded when their designated workstations are down. This improves the robustness of the factory network. Finally, the deployment of the Petri Nets Simulator over different operating systems results in minimal variation on its performance. Thus, the Petri Nets Simulator can be concluded to be a portable Petri-Net controller.

REFERENCES

[1] Alan A. Desrochers, Robert Y. Al-Jaar, "Applications of Petri Nets in Manufacturing Systems", IEEE Press 445 Hoes Lane, P.O. Box 1331 Piscataway, NJ 08855-1331 1-800-678-IEEE

[2] Zhou Meng Chu, and Kurapati Venkatesh, "Modeling, Simulation, And Control Of Flexible Manufacturing Systems: A Petri Net Approach", World Scientific Publishing Co. Pte Ltd., 1999

[3] Douglas C. Schmidt, "Developing Distributed Object Computing Application with CORBA", Elec. & Comp. Eng. Dept, University of California, Irvine

[4] Chung-Hsien Kuo, Han-Pang Huang, "Integrated Manufacturing System Modeling and Simulation Using Distributed Colored Timed Petri Net", Robotics Laboratory, Department of Mechanical Engineering National Taiwan University, Taipei, TAIWAN 10674, R.O.C.

[5] Douglas C. Schmidt, Fred Kuhns, "An Overview of the Real-Time CORBA Specification", IEEE Computer Special Issue, June 2000