Transactions on Information and Communications Technologies vol 6, © 1994 WIT Press, www.witpress.com, ISSN 1743-3517

Petri net simulation of flexible manufacturing systems

L. Chen & J. Johnson
Department of Computer Science, University of Regina, Regina, Saskatchewan, Canada, S4S 0A2

Abstract

Flexible manufacturing systems (FMS) are a class of systems exhibiting concurrency, asynchronicity and distributedness. In this project the Petri net is incorporated along with discrete events simulation to simulate scheduling different types of job processes in FMS. In order to get high throughput and uniform loading on each station in FMS, a Large-sized Petri Net (LPN) model has succeeded in finding all the possible process sequences for jobs in FMS. The SIMSCRIPT language is used to find, for each job, the state of each workstation in all possible sequences and then to select an optimal sequence for doing the job. In contrast with relying on static optimization, in Petri net scheduling we are able to find the online optimal sequence dynamically. The results suggest that Petri net scheduling can be successfully employed to study online scheduling of FMS using an expert systems approach. In addition, Large-sized Petri nets (LPN) are developed. This kind of LPN can be formally established, graphically represented and can be easily implemented by software. An algorithm for analyzing LPN is developed.

1 Introduction

Flexible manufacturing systems (FMS) are discrete systems where the variables take integers and the materials are processed as pieces or sets. Since the performance of the system design must be evaluated based on the machining of different kinds of workpieces which require different sequences of operations and have significantly different machining times, it means that the type of workpieces is a variable and the arrival rate of workpieces is a random variable (e.g., Poisson distribution). Besides, the process time for workpieces is also a random variable and the sequences of operations are non-deterministic. Another feature of FMS is concurrency which means that some actions need several conditions to be satisfied simultaneously and that some actions can be processed in parallel. This requires timing con-

siderations which suggests that stochastic simulation is a suitable approach for FMS evaluation and design.

There are many approaches for modeling or simulating FMS, each having its own advantages and problems. Now we discuss the approaches which will be used in the project.

1.1 Stochastic system simulation

The predominant theoretical approach to stochastic scheduling is that of queuing theory. In this approach the jobs are assumed to arrive in a random process with known distribution. They queue until their assigned machine is free. Then a job is selected from the queue and assigned to that machine according to some predetermined priority rule. The processing time for the assigned job is assumed to be a random variable with a known distribution. Stochastic modeling is used for simulating real time systems and for finding sensitive system factors for optimization. The research done in this area beholds open problems which do not appear to be able to be solved by this method. One of the major problems with stochastic simulation is that a predetermined priority on machine sequencing is required.

1.2 Petri Net simulation

A Petri net is defined as a bipartite directed multigraph consisting of a set of vertices and bag of directed arcs. The term 'bipartite' means that a set of vertices can be divided into two disjoint subsets P (Place) and T (Transition). The structure C of a Petri net can be expressed by four kinds of sets: place P, transition T, input I, and output O, as C = (P,T,I,O) [1].

Thus, a Petri net consists of transitions and places with both multiple inputs and outputs. When simulating a Petri net using a token (or marking), a represented token in a place indicates that the corresponding condition is 'true'. A transition 'fires' when all of its input conditions are true, i.e., all of its input places are marked with a token. The firing then removes one token from each input place and deposits one token in each output place.

Petri net simulation is suitable for discrete and concurrent systems. The dynamic behavior of sequences of operations in FMS can be represented using Petri nets. Events such as cutting tool failure, machine breakdown, and part-type changing can be explicitly shown in the model. The system is simulated non-deterministically, i.e., the net provides different choices for the simulated sequences. Research into using Petri nets for modeling FMS shows that they have limited power for this purpose because they lack a sense of time. Transitions 'fire' instantaneously, and there is no concept of *clock*.

1.3 Combining Petri net with Stochastic modeling

We wish to combine stochastic modeling with the Petri net method to simulate FMS more accurately. There are many choices. One method is to build the model by using a *Timed Petri net* (TPN) which incorporates petri net transitions with a stochastic time distribution. When the simulated system has large loading conditions, the TPN solution becomes harder to compute. Also, writing concurrent simulation programs for a Petri net is more complex

than if a notion of time is excluded. Therefore we use an approach which first uses a Petri net to get all possible sequences, and then uses stochastic simulation to dynamically choose an optimal sequence for each job routing. The SIMSCRIPT simulation language is used to write the stochastic simulation program because it can easily represent concurrent behaviors with very simple yet powerful code. In addition, simulation statistics useful for verifying our new Petri net variant are automatically generated by the SIMSCRIPT system. Petri net modeling affords efficiently for finding all the possible scheduling sequences and therefore helps simplify the SIMSCRIPT programming task.

2 Large-sized Petri net representation

The advantage of using Petri nets is that they provide graphical models with formal methods of analysis. However, graphical representation of Petri net (PN) models becomes difficult even for medium-sized systems since such graphs tend to become inconveniently large.

2.1 Basic definitions

We first present basic definitions of Petri nets [1]. Following the basic definitions, we give definitions for Large-sized Petri nets representations.
In a Petri net $C = (P, T, I, O)$ we have the set of places

$$P = \{p_1, p_2, ..., p_m\} \quad (m > 0)$$

and the set of transitions

$$T = \{t_1, t_2, ..., t_n\} \quad (n > 0)$$

with

$$P \cap T = 0.$$

Input and output set: input $I : T \rightarrow Pb$ and $P \rightarrow Tb$; output $O : T \rightarrow Pb$ and $P \rightarrow Tb$ (b means bags).
Marked Petri net: In order to reflect the execution of a Petri net, tokens are used to mark the places of a Petri net in such a way that a place with a token indicates the readiness of it for firing a transition. A marked Petri net M = (C,μ) is a Petri net structure $C = (P, T, I, O)$ with a marking μ where μ is a set of tokens. This is also sometimes written as

$$M = (P, T, I, O, \mu).$$

Petri net Graph: In the graphic representation of a Petri net a circle represents a place and a bar represents a transition. Tokens are represented by small dots in the circles.

Now let us consider a Large-sized Petri net where many operations need to share the same resource. If the resource is used by one operation then the others have to wait until the resource is released. In classic PN representation this resource place would have many output arcs to the operations which ask for it and many input arcs from the operations which release it. This makes the PN graph difficult to understand. To solve this problem, we

propose a *Large-sized Petri net graph representation (LPN)* as illustrated in Figure 3. A resource in free state can be used by any operation. For example, machine M1 in free state can be represented by the place P3 in Figure 3. If there is a token in it, then the machine M1 is available for any required operations.

All the resources in a free state place are separated from the main PN graph. These places are indirectly connected with the main graph by labeled input and output arrows which are its original input and output transitions.

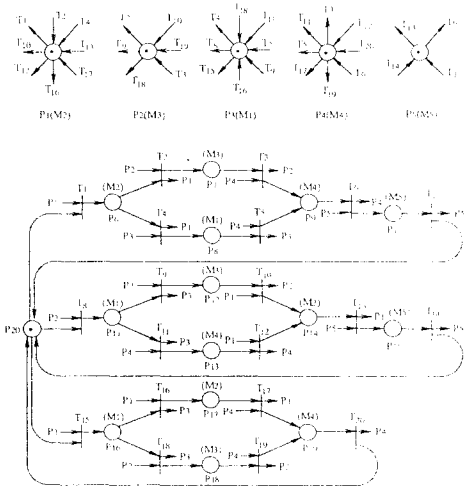

Figure 1: An example of a large-sized petri net graph

2.2 Basic Rules for Using LPN Representation

For representing an FMS in LPN, the following basic rules are used:

1. Put all the places which are resources in free state on top of main Petri net.

2. Draw out all the paths of all the parts which need to be machined in the same time period in the main PN.

3. For each part in the PN, if the part needs to use a resource, that is, it must pass a transition, then that transition must have an input arrow from the required resource place. In LPN we do not need to directly connect that resource place with the transition. Instead we draw an arrow labeled with that resource place number. Also when the part is finished with that resource it must pass the other transition and at this transition we again draw an arrow to the resource place labeled with the resource output.

4. Each resource has a number of input and output arrows labeled with transition number which correspond to the main PN.

In handling different job types in the system at the same time, each type of job has its own path in LPN, corresponding to the operation sequences for that type of job. In an LPN all the paths begin with an initially marked

place and end at that place. There exists a one to one correspondence from the cyclic firing sequences in LPN to the job operation sequences in FMS.

3 An algorithm for analyzing LPN

Some algorithms have been proposed for analyzing PN [5,13]. While this place-transition-token representation is used to indicate the static behavior of the system, the dynamic behavior of the system can be obtained by firing transitions. The union of all reachable states is the state-transition graph. The parts cycling through an FMS are then defined as sets of closed paths in this graph. An algorithm has been developed to identify those paths as well as deadlocks in the graph.

The following LPN notation is used in this algorithm:

LPN = (P,T,C,M,F) where
P = the set of places
T = the set of transitions
C = the incident matrix of LPN
M = the matrix of all changed states in a cyclic path.
F = the finite set of firing sequences

The incidence matrix C of the LPN is used to store the relations among the nodes (transitions and places) and is defined as:

C = (C(t,p)) where $t \in$ T, $p \in$ P such that:
C(t,p) = -1 if p is an input place for transition t
C(t,p) = 1 if p is an output place for transition t
C(t,p) = 0 otherwise.

M is a matrix to store all markings for the transition firing in a cyclic. F is a list used to store the firing sequences while searching for a cycle.

A table is used to help develop the algorithm (See table of Figure 2). In this table, the set of places, P, appears on the left side of the table and the set of transitions, T, along the upper side of table. M(II,0) appears to the left of P indicating the initial markings. C appears in the area created by P and T.

The purpose of this algorithm is to search for all the possible cycles and deadlocks in the graph. It works in a similar way to tree-searching algorithm. The algorithm begins with a description of the initial system. The successors of a node are determined by firing one of the applicable transitions. The associated state description of each successor is checked to see if it equals to one of the previous state descriptions in the path. If the current state equals one of the previous states then a cycle is found. If there is a node from it then no transition can be fired and this path is deadlocked. If a deadlock or a cycle is not found, the process of firing transitions continues. A stack is used for back tracing. Whenever a transition T_i is fired, we push the current state onto the stack. When a cycle or deadlock is encountered, the current firing list is printed out, and the searching will continue with a node popped from the stack. The cycles and deadlocks are recorded so that they will not be searched again. The searching procedure is continued until the stack is empty, which means that all the possible cycles and deadlocks in the graph have been reached.

As can be seen from the example shown in Figure 3, a cycle of transitions $T1 \rightarrow T2 \rightarrow T3 \rightarrow T6 \rightarrow T7$ is first found by the algorithm, corresponding to operation sequence of workstations 2, 3, 4, 5 of job type 1. Then the

M(II,0)	P\T	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20
1	P1	-1	1		1						-1		-1	1			-1	1			
1	P2		-1	1						-1	1								-1	1	
1	P3				-1	1			1	1		1				-1	1		1		
1	P4			-1		-1	1					-1	1					-1		-1	1
1	P5						-1	1						-1	1						
0	P6	1	-1		-1																
0	P7		1	-1																	
0	P8				1	-1															
0	P9			1		1	-1														
0	P10					1	-1														
0	P11							1	-1			-1									
0	P12								1	-1											
0	P13										1	-1									
0	P14										1		1	-1							
0	P15													1	-1						
0	P16															1	-1		-1		
0	P17																1	-1			
0	P18																		1	-1	
0	P19																	1		1	-1
1	P20	-1						1	-1						1	-1					1

Figure 2: Table expression of incident matrix for large-sized petri net model (P: places; T: transitions)

algorithm traces back to transition T6, and further to T1, where a new cycle of transitions $T1 \rightarrow T4 \rightarrow T5 \rightarrow T6 \rightarrow T7$ is found. This cycle gives another operation sequence of workstations 2, 1, 4, 5 of job type 1. The algorithm continues until it finds all the cycles and deadlocks in the graph.

4 Simulation Example

The system has one AGV (automated guided vehicle) which moves the workpieces around the system and travels at a speed of 5 feet/second. There are five work-stations in the system, each of which has a number of machines. The work-station layout is given as system parameters. One feature of the system is the equal priority assigned to both the AGV and the work-stations. Every type of job has equal processing priority. For simplicity, we assume there are enough buffers for workpieces waiting in the queue of each work-station. The workpieces are of three types. The parts of type-one have a four phase operation schedule, the required sequence of work-stations is M2, M3, M4, and M5; Alternating operation sequence is M2, M1, M4, and M5. Type-two parts also have a four phase operation schedule, one is M1, M3, M2, and M5, another is M1, M4, M2, and M5. Type-three parts have a three phase operation schedule, i.e., M1, M2, and M4, or M1, M3, and M4. Furthermore, there is a requirement concerning the production mix that has to be met: type-one parts must account for 50% of the overall production, type-two for 30%, and type-three for 20%. The interarrival time for the new arriving workpieces obeys an exponential distribution with mean value of 0.15 hour. There is an I/O station for workpiece input and output from the system. The system uses FIFO principle for workpiece processes. All the work-stations and AGV can work concurrently.

For every moment of time the state of the system is represented by the number and the distribution of marked places in the system. A system

activity element (place) could be either working or free at each moment. For this reason we let the place nodes P1, P2, P3, P4, and P5 represent work-stations 2, 3, 1, 4, and 5 which are at the free state; P20 represents a new workpiece (regardless of its type) which is waiting for processing; The remaining places (P6 – P19) stand for the specific matchings between work-stations and job types. For example, P6 represents that work-station 2 is machining job-type 1.

When the system is about to change one state to another state, it needs to check if the preconditions for this change are satisfied. If so then the system will change into a new state. Transition nodes are defined as: T1, T2, T3, ... , T20 indicate which conditions should be satisfied and what resources can be released at the moment when a job requires some operations (resources). For example, T2 indicates that if work-station 3 is free and work-station 2 has finished operation for processing job-type 1 then job-type 1 can be transferred to work-station 3 for processing. Meanwhile work-station 2 is released, and can be used for the other jobs. Initially all work-stations in the system are free and only one workpiece has arrived.

By using the foregoing description, we can draw a Large-sized Petri net for the model system, as shown in Figure 3. Figure 2 gives the incidence matrix corresponding to the LPN in Figure 3. The algorithm given in section 2 is used to find all the possible cycles in the LPN graph. These cycles correspond to all the possible operation sequences for all type of jobs, and are taken as input data for the SIMSCRIPT simulation program.

4.1 Simulation results

The approach developed in this project is a dynamically optimal job routing strategy. Usually the static optimal job routing is used as a system job routing strategy. The routing strategy is called *general job routing* if a job routing is randomly picked when there is no optimal routing. In order to compare this approach with others we can use the same set of data to do the modelling in different types of job routing strategies. The approach of dynamic optimal routing gets higher throughput than any other method. It is important to see that this approach results in a uniform load on each work-station in FMS. In the dynamic optimal routing the average number of jobs in the queue is 8 and for static optimal routing the average number of jobs in the queue is 22. For general routing the number is as high as 32. Considering that there are only a limited number of buffers for each work-station, a large number of jobs in the queue would cause a system blockage. Also the average total job delay in the queue is shorter than in the other strategies.

Furthermore this approach can be used for online scheduling for FMS. If the type of job changes, the model of LPN can be modified by simply adding a new path for the new type job, and the corresponding places and transitions can be easily changed in the incidence matrix. Also if some work-stations break down, then LPN incidence matrix will send a signal, which will cause that path to deadlock and the job may be continued following another job routing sequence. Another advantage of this approach is that LPN programs find all the possible sequences and SIMSCRIPT can simulate those results quickly.

	dynamic optimal routing		static optimal routing		general routing	
Throughput	208		191		139	
AGV utilization	0.434		0.400		0.384	
workstation	average number of jobs in the queue	probability of working	average number of jobs in the queue	probability of working	average number of jobs in the queue	probability of working
1	8.77	0.859	23.06	0.912	0.66	0.635
2	1.41	0.731	0.39	0.711	0.20	0.472
3	1.74	0.755	1.74	0.793	11.37	0.898
4	0.77	0.661	0.37	0.515	32.80	0.951
5	4.71	0.819	1.82	0.639	1.14	0.607
Job type	average total job delay in queue	average total transporter delay	average total job delay in queue	average total transporter delay	average total job delay in queue	average total transporter delay
1	3.39	0.07	4.73	0.07	6.53	0.06
2	1.86	0.06	4.09	0.05	7.87	0.06
3	3.19	0.07	5.12	0.06	7.92	0.07

Figure 3: Comparison of three types of job routing strategies

5 Conclusions

The approach of using Large-sized Petri net and stochastic simulation of FMS provides a more flexible model for development. The Large-sized Petri net (LPN) provides a method for graphically and rapidly simulating FMS. Using the SIMSCRIPT language can substantially reduce both programming and project time. By design, it offers language, program, and data structures that make modeling much easier to develop and modify.

The purpose of using this approach is to find out the optimal online dynamic job scheduling strategies. Simulation results have shown this approach more efficient than other job scheduling approaches.

References

[1] Peterson, J.L.,*Petri Net Theory and the Modeling of Systems*, Englewood Cliffs (NJ), Prentice-Hall, 1981.