An Automatic Petri-Net Generator for Modeling a Flexible Manufacturing System

Tien-Hsiang Sun *Li-Chen Fu*

Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.

Abstract

Petri-Net is widely used in the field of modeling a flexible manufacturing system (FMS) because of its intrinsic features such as concurrency, asynchronization, conflict, and event-triggering. But designing a Petri-Net Model for an FMS is extremely tedious and difficult for debugging. Therefore, an automatic Petri-Net generator which can generate a bug-free model for an FMS is rather useful and practical. In this paper, a systematic method to construct a Timed Place Petri-Net model for modeling an FMS is proposed, and is implemented as an Automatic Petri-Net Generator, with a Graphic User Interface, through which a user can input the system information of a practical FMS, such as the number of automated guided vehicles (AGVs), the number of machines, and their geometric relationship, as well as process flows of parts to be processed. Then, this generator will use these pieces of data to generate the corresponding Petri-Net Model.

1 Introduction

Petri-Net has evolved into an elegant and powerful graphical modeling tool for asynchronous concurrent event-driven systems. The wide ranging application areas of Petri-Net include communication protocols [18, 19], distributed systems [20], database [22], multi-processor memory systems [21], complier and operating system [23, 24], formal languages [25, 26]. In additional, Petri-Net are very useful tool for modeling [4, 5, 17], simulation [6, 17], and scheduling [7] of discrete event systems, such as flexible manufacturing systems, and also for designing and implementing controllers for manufacturing [11, 12]. In the earlier work of our laboratory, a Petri-Net model is built to model the detailed behavior of an FMS and a schedule is generated by running an A^* search on the Petri-Net model. The result is quite satisfactory.

However, designing a Petri-Net model for an FMS is extremely tedious and difficult for debugging. For this reason, it is extremely necessary to have a Petri-Net generator to generate a bug-free Petri-Net model for an FMS. A generator is a program that can generate some specified program or data structure from the relevant system information input by users. For example, in a UNIX system, there are Yacc as Paser Generator and Lex as Scanner Generator. And there is also some similar software in the context of Petri-Net, for example, Design/CPN is an interactive computer tool for performing modeling and simulation with Color Petri-Net. In this paper, a systematic method to construct a Timed Place Petri-Net (TPPN) model for modeling an FMS is proposed, which inherits some excellent features in modeling from our early work but is now more thoroughly and systematically refined. This systematic modeling method is implemented as an automatic Petri-Net generator (APNG). The term "Automatic" means that users do not need to design the Petri-Net model by themselves, but the Generator will create all kinds of Petri-Net model of any FMS subject to users definition. By using such a generator, building a Petri-Net model of a FMS is no longer a difficult and time-consuming task. User only need to input the system information of a practical FMS, such as the number of automated guided vehicles (AGVs), the number of machines, and their geometric relationship, as well as process flows of parts to be processed. We also develop a graphic user interface to help user input these information more easily.

The organization of this paper is described as follows. In Section 2, the detailed systematic Petri-Net modeling method will be introduced. In Section 3, a practival example of using the proposed machinery will be presented. Finally, conclusions are provided in Section 4.

2 Systematic Modeling Method

In this paper, we select the Timed place Petri-net (TPPN), in which time is associated only with places and all transitions are instantaneous, to model our manufacturing systems. Because the markings of the TPPN are deterministic during the evolution of the firing sequence from initial marking, we can undoubtedly use the markings of the TPPN to describe the states of the system and all the reachable markings can represent the state space of the modeled system.

Fig. 1 shows the hierarchical view of the entire system model, which contains two major sub-models. One is called Transportation Model and the other is called Process-Flow Model. The objective of the Transportation Model is to model the behavior of the AGV traveling from the current stop to its destination stop, and that of the Process-Flow Model is to describe the behavior of the part routing and resource assignment. The two sub-models, of course, are interactive with each other to undertake the necessary actions in response to the triggering from another.

The Transportation Model is decomposed into several sub-modules, each with different characteristics. One is Transportation Layout Module, which models the relative positions of AGV stop points and the path segments connecting adjacent stop points, and the other is Movement Control Module,

0-7803-2129-4/94 $3.00 © 1994 IEEE

which models the routing control rule of an AGV moving from one stop point to another. However, if there is more than one AGV in the system, the Push Control Module has to be additionally included in order to avoid the AGV collision problem. Each of the above modules is further composed of several micro modules called Transportation Unit Module, Movement Control Unit Module, and Push Control Unit Module. Similarly, the Process Flow Model is composed of various unit modules : (1) AGV Link Unit Module is used when a part need to be transported by AGVs. (2) Cell Link Unit Module is to describe the cell material handling system (robot, rail guided vehicle) when the system consists of several machine cells. (3) Virtual Link Unit Module characterizes the situation where two adjacent operations can be processed by the same machine, (4) Job Start & End Module describes the pool of readily processed parts and of finished parts of a job. (5) Buffer Link Unit Module is used to represent the situation where a part is moved from a machine input buffer into the machine for processing and a processed part is moved out from the machine to its machine output buffer.

The entire model is synthesized by these sub-modules or modules and micro modules which are in conjunction with one another using common places, and hence every place in the model will be associated with a unique name.

2.1 Modeling of a Transportation System

The Transportation Model can be decomposed into several micro-models (called modules), each with different characteristics, one is Transportation Layout Module and the other is Movement Control Module. However, the Push Control Module has to be additionally included if the number of the material handling carriers is greater than one, i.e., if a multiple-AGV system is installed. The following are detailed descriptions of each of these micro-models and we will provide algorithms to show how the Petri-Net generator can automatically generate each of these micro-models.

Transportation Layout Module

The purpose of the Transport Layout Module is to model the layout of the transportation system. The basic concept of this model is described as follows. When an AGV needs to move from the current stop to the next adjacent stop, it needs to receive a 'ticket' of movement first to know its destination and then acquires the control right of the next adjacent stop to make sure that stop is free at the moment. If both of these conditions are satisfied, it can thus start its traveling to the next adjacent stop. At the same time, the control right of the current stop will be released to allow another AGV to use it as a destination or pass-by stop.

Algorithm 1 shows how the generator can automatically generate the full Transportation Layout Module. Some constants used in the algorithms of this paper are described as follow :

- N_SP is the total number of stop points of the transportation layout of an FMS.
- N_AGV is the total number of AGV of an FMS.
- S_ARC is the set of arcs of the Layout Directed Graph of an FMS.

Algorithm 1 :
Construct_Transportation_Layout_Module

Step 1: FOR $a = 1$ TO N_AGV
Step 2: FOR every $(i, j) \in S\_ARC$
Step 3: Construct a Transportation Layout Unit Module (as shown in Fig. 2)
Step 4: NEXT
Step 5: NEXT

The notation of places used in Fig. 2 are explained as follows:

1) st_i_a, represents the stop at a workstation. A token in st_i_a means AGV a is now staying at stop i.
2) ctrl_i, represents the control right of stop i.
3) tk_i-j_a, represents a ticket that drives AGV a to move from stop i to stop j.
4) mv_i-j_a, represents the moving of AGV a from stop i to j.
5) mv_ok_a represents the status of completing the AGV movement along a segment path.

It is noteworthy that up to now the Transportation Model has already embedded a control rule into it so that no vehicle collision problem will ever occur.

Movement Control Module

The Movement Control Module corresponds to the decision-making Petri-Net model for AGV routing. Each time when AGV a wants to move from the current stop to some other stop, it must determine its route of movement first. The determined route can be sent to the Transportation Layout Module through 'ticket' places tk_i-j_a to entail the AGV a to move along that route. Therefore, for each stop, we need to have a corresponding Petri-Net model to take care of the routing decision, guiding the traveling path from other stops to it.

Because there may be more than one path from stop point j to stop point i, the Petri-Net generator must find all paths from each stop point to a specified stop point. This can be done by traversing the Layout Directed Graph, and algorithm 2.1 shows how this is done by the generator and Algorithm 2.2 describes the procedure of generating Movement Control Module.

Algorithm 2.1 : Find_All_Path_To_Stop_Point(i)

Step 1: ARRAY $Status[1, \ldots, N\_SP]$
Step 2: MATRIX $Answer[1, \ldots, N\_SP][1, \ldots, N\_SP]$
Step 3: Initialize STACK
Step 4: Initialize $Status[1, \ldots, N\_SP] = 0$
Step 5: Initialize $Answer[1, \ldots, N\_SP][a, \ldots, N\_SP] = 0$
Step 6: PUSH i into STACK
Step 7: $Status[i] = 1$
Step 8: WHILE STACK is not empty
Step 9: POP j out of STACK
Step 10: $Status[j] = 2$

Step 11: FOR $k = 1$ TO N_SP
Step 12: IF $(k, j) \in S\_ARC$ AND $Status[k] = 0$ THEN
Step 13: $Status[k] = 1$
Step 14: PUSH k into STACK
Step 15: $Answer[k][j] = 1$
Step 16: END IF
Step 17: NEXT
Step 18: END WHILE
Step 19: RETURN *Answer*

The meaning of the array *Status* and the matrix *Answer* are respectively described as follows :

- $Status[i] =$
  - 0 : point i has not been traversed;
  - 1 : point i has been traversed but not expanded yet;
  - 2 : point i has been traversed and expanded.
- $Answer[i][j] =$
  - 0 : there is no arc from point i to point j or the arc is no use for the traversal;
  - 1 : the arc from point i to point j is useful in the traversal

The notations of places which are different from those in the Transportation Layout Unit Module are described as follows.

1) The place mv_st_i_a represents the request of AGV a moving from the current stop to stop i.
2) The place at_st_i_a represents the request of AGV movement from the current stop to stop i is fulfilled.
3) The place mv_wait_i_a means a waiting for the mv_ok_a condition to generate a 'ticket' for the next segment path.

Note that mv_st_i_a and at_st_i_a places are both control places that are used to inform the Transportation Model to move an AGV to stop i arising from the request made by Process Flow Model and then to resume the processing of parts in Process-Flow Model, respectively. The palce mv_wait_i_a is an intermediate place and is used only to wait for the place mv_ok_a to be marked in the Transportation Layout Module after interior firing.

Algorithm 2.2 : Construct_Movement_Control_Module

Step 1: MATRIX $Path[1, \ldots, N\_SP][1, \ldots, N\_SP]$
Step 2: FOR $a = 1$ TO N_AGV
Step 3: FOR $i = 1$ TO N_SP
Step 4: Construct a Movement Control Unit Module (a) (Fig. 3 (a))
Step 5: $Path$ = Find_All_Path_To_Stop_Point(i)
Step 6: FOR every $j, k \in \{1, \ldots, N\_SP\}$
Step 7: IF $Path[j][k] = 1$ THEN Construct a Movement Control Unit Module (b) (Fig. 3 (b))
Step 8: NEXT
Step 9: NEXT
Step 10: NEXT

Push Control Module

In a production system, the transporting system may have more than one carriers which transport materials among different workstations. Though, there no longer exist collision problems, the traffic jam problems due to multiple carriers must be considered and solved. The objective of the Push Control Module is again to introduce a control method into the Petri-Net model with an aim to guaranteeing the jam-free condition among carriers.

The method we provide is a way called *push-AGV* strategy, whose basic idea is described as follows. When a busy AGV found that a stop on its traveling route was occupied by another freed (idle) AGV, it will send a push command to that idle AGV to ask it to leave that stop. After a 'ticket' is sent to that idle AGV, it starts to move to its next adjacent stop and releases the occupancy of its previously residing stop. When that stop is completely released, the busy AGV can resume its traveling on the same route path.

The Push Control Module can be constructed based on Push Control Unit Modules, as shown in Fig. 4. Consider every three adjacent stop points. If AGV a is at stop i and AGV b is at stop j and AGV a wants to move to stop j while AGV b is idle, then AGV b will be pushed by AGV a to the other stop point, e.g. stop k. So, this module sends a token to the place tk_j-k_b in the Transportation Layout Module and, when AGV b completes its movement, the module will send a token to mv_ok_b so as to inform AGV a to start its movement. Algorithm 3 summarizes the procedure of generating the Push Control Module.

Algorithm 3 : Construct_Push_Control_Module

Step 1: FOR every $(i, j) \in S\_ARC$
Step 2: FOR every $k \in \{1, \ldots, N\_SP\}$
Step 3: IF $(j, k) \in S\_ARC$ THEN
Step 4: FOR every $a, b \in \{1, \ldots, N\_AGV\}, a \neq b$
Step 5: Construct two Push Control Unit Modules (Fig. 4) : one for AGV a pushing AGV b and the other for AGV b pushing AGV a
Step 6: NEXT
Step 7: END IF
Step 8: NEXT
Step 9: NEXT

2.2 Modeling of a Process Flow

A process flow in manufacturing can be seen as a sequence of *part transportations* by the material handling system and the transporting system among different workstations, and *part processing* on numerically controlled NC machines. The Petri-Net model we proposed to represent the process flow of each part type is called Process-Flow Model, which models the technological precedence constraints for processing parts and normally provides more than one alternative route to accomplish the processing.

There are five micro modules that we use to generate the Process Flow Model, while we now introduce one by one in the following.

1. *AGV Link Unit Module* : is used for describing the situation where a part is transported via the AGV system, which starts from a call of an AGV to the source stop point, then the loading operation, and commanding the AGV to the goal stop point, and finally ends with the unloading operation. The behaviors of AGV Link Unit Module is that, when a part needs a service of an AGV at stop point i and if there is a freed AGV as well as the target resource (*resource_2*) is free, then the module issues a moving command to the Movement Control Module of the Transportation Model by placing a token in the place "mv_st_i_a" to drive AGV a to move to the source stop point at which the part is residing. When the movement is complete, the Movement Control Module will then place a token in the place "at_st_i_a" and then starts the loading operation, which takes some time. Once the loading operation is done, this module will send another moving command to the Movement Control Module by placing a token in the place "mv_st_j_a to drive AGV a to move to stop point j and at the same time release the *resource_1*. Similar to the above description, when the movement is done, there is a token in the place "at_st_j_a", which is then followed by the unloading operation. Finally, when the unloading operation is completed, the mission for the AGV in the AGV Link Unit Module is considered accomplished.
2. *Cell Link Unit Module* : is used for describing the situation where a part is transported by the Cell Material Handling System, e.g. (Robot or RGV), when a part in the Cell needs to be transported from a location to another location in the cell.
3. *Virtual Link Unit Module* : is used for describing a situation where a part can be processed with two successive operations in the same machine.
4. *Job Start & End Module* : is used to symbolize the starting and the ending of a Job.
5. *Buffer Unit Module* : is used for describing the situation where a part is passed from the machine input buffer into the machine for operation, and is then passed to the machine output buffer when the operation is done.

By using these five unit modules, a process flow of a job can be easily mapped to the corresponding Petri-Net. The algorithm to build the Process Flow Model, because of the limiting space, is not presented here and can be found in [34]

3 Implementation

The implementation of the systematic Petri-Net modeling method is through a program called Automatic Petri-Net Generator. By further, we also combine the generator with a search based scheduling method which is proposed by our earlier work [32]. The integrated version of the above modeling as well as scheduling programs is design by an Object-Oriented method and programming with C++. Compilation is done via GNU C++ Ver 2.5.7 on Sun SPARC II Workstation.

We use an FMS in Automation Lab. of Department of Mechanical Engineering at National Taiwan University as our target system. The components of this FMS contains an L/U station, a robot, an AS/RS, a Lathe machine, a Milling machine, a Machine Center and two AGVs. The layout of this target system is shown in Fig. 5. Note that, the Machine Center has input and output buffer, Lathe and Milling machine are in the robot cell and has no input and output buffer, each of them using the common input and output buffer of the robot cell. That is, parts which enter or leave Lathe or Milling machine must be transfered by robot. Table 1 represents the job requirements of an example. By the running the Automatic Petri-Net Generator, the final Petri-Net model contains of 377 places, 429 transitions, 838 transition output arcs, 802 transition input arcs. After the model is generated, the search algorithm next searches on the reachability graph, and finally the program outputs a schedule.

4 Conclusion

In this paper, we proposed a systematic Petri-Net modeling for an FMS. The entire model is composed of two sub-model, one is Transportation Model and the other is Process Flow model. The objective of Transportation Model is to model the behavior of AGV traveling from the stop at which it currently stays to its destination stop, which must also ensure that the collisions are avoided. And, the Process-Flow Model is used to model the behavior of the part routing and resource assignment several algorithms are used to construct both Transportation Model and Process Flow Model, which then comprise the Automatic Petri-Net Generator (APNG). An A* search based scheduling method was also proposed to provide an appropriate schedule.

References

[1] C. A. Petri, "Kommunikation mit Automaten," PhD thesis, University of Bonn, Germany, 1962.

[2] J. L. Peterson, "Petri nets," Comput. Surveys, vol. 9, Sep. 1977.

[3] T. Murata, "Petri Nets : Properties, Analysis and Applications," Proc. IEEE, vol. PORC-77, no. 4, pp. 541-580, Apr. 1989.

[4] Y. Narahari and N. Viswanadham, "A Petri net approach to modeling and analysis of flexible manufacturing system," Ann. Operation Res., vol. 3, 1985, pp. 449-472.

[5] Chin-Jung T., Li-Chen F. and Yung-Jen H., "Modeling and Simulation for Flexible Manufacturing Systems using Petri Net", The second International Conference on Automation Technology, pp.31-38, vol. 4, 1992.

[6] K. Venkatesh, O. Masory, and J. Wu, "Simulation and robot scheduling of a just-in-time flexible automated system using timed Petri Nets", The second International Conf. Automation Technology, vol. 4, pp.73-79, July 1992.

[7] C. F. G. Bispo, J. J. S. Sentieiro, and R. D. Hibberd, "Adaptive scheduling for high-volume shops", IEEE Trans. Robot. Automat., pp.696-706, vol.8, Dec. 1992.

[8] C. Bispo and J. Sentieiro, "An extended horizon scheduling algorithm for the job-shop problem," Porc. 1st Int. Conf. CIM, May 1988.

[9] L. E. Holloway, B. H. Krogh, "Synthesis of feedback control logic for a class of Controlled Petri Nets", IEEE Trans. Automat. Contr., vol.35, pp.514-523, May 1990.

[10] L. E. Holloway, F. Hossain, "Feedback Control for sequencing specifications in Controlled Petri Nets", Proc. IEEE Conf. Robotics and Automation, pp.242-251,192.

[11] D. Crockett, A. Desrochers, F. DiCesare, and T. Ward "Implementation of a Petri net controller for machine workstation," in Proc. IEEE Robotics Automat. Conf., 1987, pp. 1861-1867.

[12] M. C. Zhou, F. DiCesare, and D. Rudolph, "Control of a flexible manufacturing system using Petri nets," in Proc. IFAC Congr., vol. 9, 1990, pp. 43-48.

[13] E. Kasturia, F. DiCesar, and Desrochers, "Real-time control of multilevel manufacturing systems using colored Petri nets," in Proc. IEEE Robotics Automat. Conf., 1988, pp. 1114-1119.

[14] K. H. Lee and J. favrel, "Hierarchical reduction method for analysis and decomposition of Petri nets," IEEE Trans. Syst. Man Cybern., vol SMC-15, no. 2, 1985, pp. 272-280.

[15] K. H. Lee, J. Favrel, and P. Baptiste, "Generalized Petri net reduction method," IEEE Trans. Syst. Man Cybern., vol. 17, no. 2, 1987, pp. 297-303.

[16] J. C. Gentina and D. Corbeel, "Colored adaptive structed Petri net: A tool for the automated synthesis of hierarchical control of flexible manufacturing systems," in Proc. IEEE Robotics Automat. Conf., 198", pp. 1166-1173

[17] Kimon P. Valavanis, "On the hierarchical modeling analysis and simulation of flexible manufacturing systems with extended Petri nets," IEEE Trans. Syst. Man Cybern., vol. 20, no. 1, 1990, pp. 94-109.

[18] G. Berthelot and R. Terrat, "Petri nets theory for correctness of protocols," IEEE Trans. Commun., vol. COM-30, 1982, pp. 2497-2509.

[19] M. Diaz, "Modeling and analysis of communication and cooperation protocols using Petri net based models," Comput. Net., vol. 6, 1982.

[20] J. M. Ayache, J. P. Courtiat, and M. Diaz, " REBUS, a fault-tolerant distribution system for industrial real-time control," IEEE Trans. Comput., vol. C-31, 1982, pp.637-674.

[21] W. E. Kluge and K. L. Lautenbach, " The orderly resolution of memory access conflicts among competing channel processes," IEEE Trans. Comput., vol. C-31, 1982, pp. 194-207.

[22] K. Voss, "Using predicate/transition-nets to model and analyze distributed database systems," IEEE Trans. Software Eng., vol. SE-6, 1980, pp. 539-544.

[23] J. L. Baer and C. A. Ellis, "Model design and evaluation of a compiler for a parallel processing environment," IEEE Trans. Software Eng., vol. SE-3, no. 6, Nov. 1977, pp. 394-405,

[24] J. D. Noe, "A Petri net model of the CDC 6400," Proc. ACM/SIGOPS Workshop on Systems Performance Evaluation, 1971, pp. 362-378.

[25] D. Mandrioli, "A note on Petri net languages," Information and Control, vol. 34, no. 2, 1977, pp. 169-171.

[26] J. L. Peterson, "Computation sequence sets," J. Comput. Syst. Sci., vol. 13, no. 1, 1976, pp. 1-24.

[27] Herve P . Hillion and Jean-Marie Proth, "Performance evalutaion of job-shop systems using timed event-graphs," IEEE Trans. Automat. Control, vol. 34, no. 1, Jan. 1989, pp. 3-9.

[28] Michael K. Molloy, "Performance analysis using stochastic Petri nets," IEEE Trans. Computers, vol. C-31, no. 9, Sep. 1982, pp. 913-917.

[29] Michael K. Molloy, "Discrete time stochastic Petri nets," IEEE Trans Software Eng., vol. SE-11, no. 4, Apr. 1985, pp. 417-423.

[30] C. V. Ramamoorthy, G. S. Ho, "Performance evaluation of asnchronous concurrent systems using Petri Nets", IEEE Trans. Software E., pp. 440-449, vol. SE-6, no. 5 Sep. 1980.

[31] Y.-L. Chen, T.-H. Sun, and L.-C. Fu, "A Petri-Net Based Hierarchical Structure for Dynamic Scheduler and Deadlock Avoidance," Proc. IEEE Intl. Conf. Robotics and Automat., 1994, pp. 1998-2004.

[32] C.-W. Cheng, T.-H. Sun, and L.-C. Fu, "Petri-Net Based Modeling and Scheduling of a Flexibile Manufactuuring System," Proc. IEEE Intl. Conf. Robotics and Automat., 1994, pp. 513-518.

[33] P.-S. Liu and L.-C. Fu, "Hierarchical Dynamic Scheduling for an FMS," Porc. 3rd. Intl. Conf. on Computer Integrated Manufacturing, 1992.

[34] Tien-Hsiang Sun, NTUCSIE, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, R.O.C., 1994.

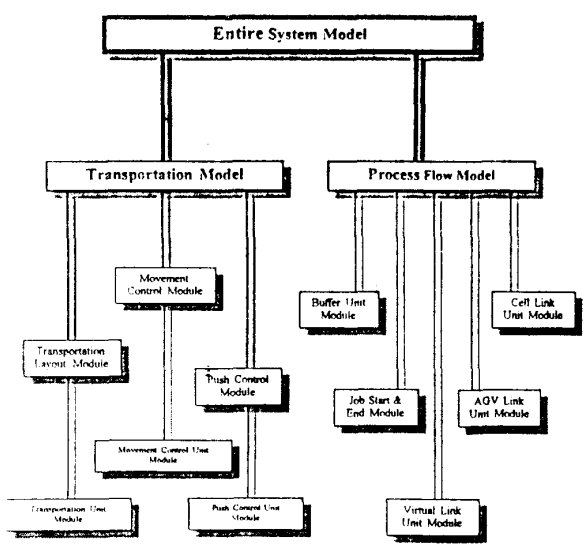


Figure 1: Hierarchical View of Model

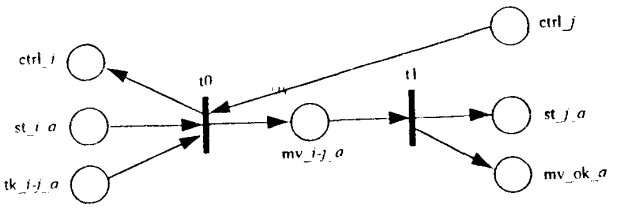
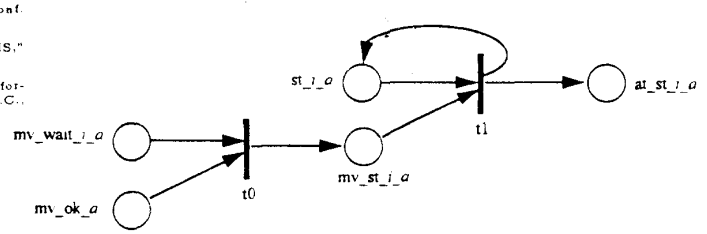
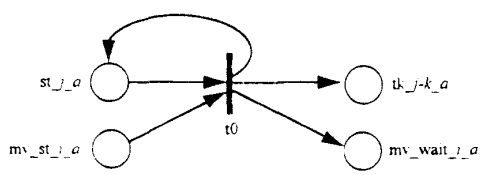


Figure 2: Transportation Layout Unit Module



Movement Control Unit Module (a)



Movement Control Unit Module (b)

Figure 3: Movement Control Unit Module

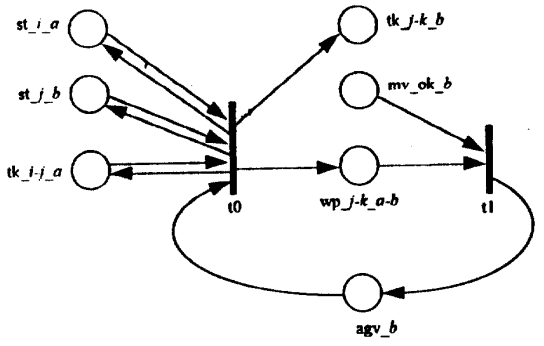


Figure 4: Push Control Unit Module

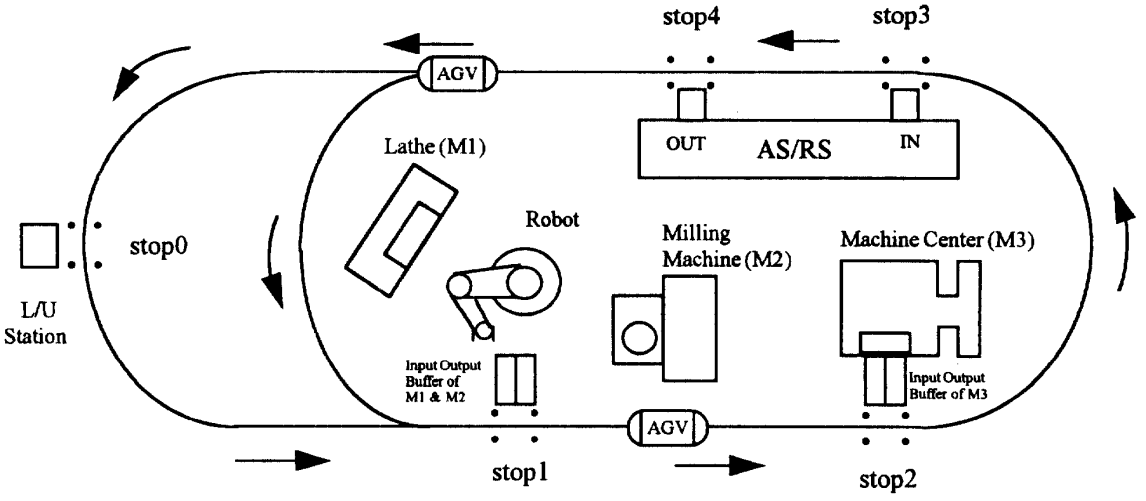


Figure 5: System Layout for Example 1

Job_1

Seq	OP_1	OP_2	OP_3
1	M1/410|M2/501	M3/135	M3/225|M2/160

Job_2

Seq	OP_1	OP_2
1	M3/250	M1/265|M2/125

Job_3

Seq	OP_1	OP_2	OP_3
1	M2/475	M3/440	M1/560|M2/470

Table 1: Job Requirement for Example 1