1991

A computer-aided simulation tool based on Petri nets for the design and analysis of FMSs

Dong-Soon Yim
Iowa State University

Order Number 9212207

A computer-aided simulation tool based on Petri nets for the design and analysis of FMSs

Yim, Dong-Soon, Ph.D.

Iowa State University, 1991

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A computer-aided simulation tool based on Petri nets for the design and analysis of FMSs

by

Dong-Soon Yim

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Department: Industrial and Manufacturing Systems Engineering
Major: Industrial Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa
1991

TABLE OF CONTENTS

GENERAL INTRODUCTION

A flexible manufacturing system (FMS) is a production facility consisting of flexible, numerically-controlled machines or work stations, automatic material handling systems, and control systems. Such systems offer several benefits compared with conventional manufacturing systems; shorter manufacturing cycle times, better resource utilization, decreased work-in process inventory, and flexible manufacturing capability. Implementation of FMS, however, creates serious problems for designers and engineers who are responsible for the design and operation of these systems. Modern FMSs are complex with a high degree of flexibility. They need a flexible operation and dynamic control over a turbulent environment. The technique used in FMS is to identify work pieces uniquely and to control their movements individually between the work stations assigned to perform the operation required. As a result, many different types of work pieces are simultaneously in-process, each following its own routing through the system. Because of these flexible characteristics, the planning, design, and control of FMSs are difficult tasks.

Discrete-event simulation is a widely used tool to solve these problems. The dynamic environment of manufacturing systems can be well analyzed using simulation techniques. In simulation, a computer is used to evaluate a model numerically over a time period of interest, and data are gathered to estimate the desired true char-

acteristics of the model. The main reasons for using simulation in FMSs are (1) to measure the performance and equipment utilization of the system, (2) to compare the performance of alternative designs, (3) to develop operating strategies for the control of work flow, and (4) to identify bottlenecks and other weaknesses in the system. Even though there are many tools now available for the simulation of manufacturing systems, the successful use of simulation is somewhat difficult. Failure is mainly due to (1) complexity of the modeling language, (2) differences between the simulation modeling concept limited by a simulation language and the system to be modeled, (3) difficulty in validation and verification of a model, and (4) no rigorous technique for the analysis of simulation output.

This dissertation is directed to the development of a computer-aided simulation tool for the design and analysis of FMSs. A computer-aided simulation tool is one which aids all activities of the simulation life cycle; modeling, model validation and verification, experiments, output analysis, and results presentation. To provide a successful environment for simulation projects, the separate modules of the simulation software must be integrated. The integrated software provides automatic-processing facilities to aid decision makers. This research is one step in this direction. The main objectives of this dissertation are:

- To develop a subclass of Petri nets suitable for modeling and analyzing FMSs.
- To propose the modeling methodologies for simulation of FMSs consisting of hardware components and control systems.
- To present a computer-aided simulation tool based on the new subclass of Petri nets.

In this study, a subclass of Petri nets, Conserved nets, is proposed and implemented in a syntax-oriented graphics tool for the creation of the simulation model of hardware components in FMSs. Petri nets have proved to be a powerful tool to model systems that exhibits synchronization and cooperation. To exploit Petri nets as a successful simulation modeling language, however, several extensions are required. Conserved nets are developed by extending ordinary Petri nets in order to be used for the modeling, analysis, and simulation of FMSs.

Petri nets provide a graphical simulation language instead of complex textual languages. The simulation model is constructed with primitive Petri net objects which are classified to correspond to hardware components of FMSs. Under the guidance of the interactive graphics tool, a model is incremented in a top-down fashion with the Conserved net modeling logic which guarantees conservativeness and liveness of the Petri net model. Model validation is performed by exploiting useful properties of Conserved nets and use of a Petri net animation.

Besides Petri net modeling of hardware components in FMSs, high-level, real-time control systems in FMSs must be modeled easily and accurately. Because the high-level control systems have an abstract and informal nature, the modeling of these systems is more difficult compared with modeling of hardware components. Although Petri nets are suitable to represent some features of FMSs, such as the distributed and concurrent nature of processes or the synchronization and conflicting properties among tasks in the use of shared resources, they have drawbacks to model high-level control systems. Instead of using Petri nets, the high-level control systems are modeled separately using a control rule specification language developed to facilitate the control rule modeling process.

The three subsystems—token player, Petri net model, and high-level control systems—interface each other during simulation. The token player executes the movements of tokens in a Petri net model, and interfaces with high-level control systems. The high-level control systems analyze the current status of the Petri net model and give commands to controllable tokens to resolve conflicts in Petri net execution.

Explanation of Dissertation Format

The format of this dissertation follows the alternate format described in the Graduate Thesis Manual of Iowa State University. It consists of three parts, each of them being an individual paper.

Part I: was developed under the guidance of Professor Thomas A. Barta. This part shows the development of Conserved nets, which are proposed to model, analyze, and simulate FMSs. A Conserved net is one in which token flows are conserved without any transformation during Petri net execution. The development of Conserved nets is mainly due to the fact that Petri net models of FMSs are required to have the conservativeness property. The structural properties and liveness conditions of the Conserved nets are described. The modeling and analyzing power of the Conserved nets is demonstrated with a case study.

Part II: was also developed under the guidance of Professor Thomas A. Barta. This part develops a Petri net-based simulation tool for the design and analysis of FMSs. In this tool, Conserved nets are implemented in Petri net objects and modeling logic. The modeling methods of several high-level, real-time control systems are included. Finally, the facilities of the developed tool are described, and the strength of the tool is demonstrated with a case study.

Part III: was developed with the help of Dr. Linn who is a professor in the Department of Industrial and Manufacturing Systems Engineering, Iowa State University. This part is an extensive simulation study using the Petri net-based simulation tool described in Part II. The objective of this study is to investigate the effect of push- and pull-based AGV dispatching rules in FMSs. A number of push- and pull-based AGV dispatching rules are proposed and compared via the simulation study. The developed simulation model consists of two modules: a Petri net model and AGV dispatcher. Experiment conditions and output analysis are included in the simulation study.

Finally, the strengths and weaknesses of the developed tool and future study are included in the general summary. The bibliography contains the references for the general summary and appendix (token player) and the related literature that are not listed in the references of each part.

PART I.

CONSERVED NETS FOR MODELING AND SIMULATION OF FMSS

Conserved Nets for Modeling and Simulation of FMSs

D. S. Yim and T. A. Barta

Department of Industrial and Manufacturing Systems Engineering

Iowa State University, Ames, Iowa 50010, USA

Abstract

In this paper, Conserved nets which are a subclass of Petri nets is proposed to facilitate the modeling, analysis, and simulation of FMSs. Conserved nets ensure that a Petri net model has the conservativeness property. From the structural characteristics of Conserved nets, liveness conditions are easily obtained. While hardware components of FMSs are modeled by using Conserved nets, high-level, real-time control systems in FMSs are separately modeled using the analysis results of the Conserved Petri net model. For the simulation of FMSs, a Petri net model and a high-level control system are integrated so that the high-level control system is responsible to resolve conflicts in the Petri net model. The modeling and simulation procedure is demonstrated with an example machine center.

Keywords: Petri nets, FMSs, Simulation.

Introduction

Basically, a Petri net is capable of modeling a multi-condition process which has concurrency and cooperation. Because of this capability, Petri nets have been widely used for the modeling and analysis of communications, operating systems, computer software and hardware, and manufacturing systems. In addition, Petri nets have been used as a simulation tool for discrete manufacturing systems.

The modeling and analysis power of Petri nets is well suited for the design of flexible manufacturing systems (FMSs) including low level control systems. However, ordinary Petri nets have limitations to describe complex systems. To increase the modeling power, a number of Petri net families have been proposed. Increasing of modeling power also increases the complexity and difficulty in analyzing important properties of a Petri net model such as conservativeness, liveness, safeness, and boundedness. As the model becomes complex, analysis based on reachability tree, invariant analysis, and reduction methods becomes difficult. As a result, two approaches have been developed. One approach is using subclasses of Petri nets by imposing some restrictions in modeling the systems. State machine [1], Marked Graph [2], Free choice Petri net [3], and Essentially Decision Free net [4] are among the subclasses of Petri nets. The other approach is modeling a Petri net which has desirable properties a priori. The synthesis of each resource activity cycle [5] and top down modeling by stepwise addition of arcs [6] are based on such an approach.

Another problem in the application of Petri nets to FMSs is its limitation to represent high level decision support systems. The control systems of FMSs are usually constructed and operated with a hierarchical structure. While low level control systems—machine level control systems—are well represented by Petri nets [7],

however, high level control systems that require high level decision capability with analysis of global system status and historical data are difficult to be modeled by Petri nets. A number of techniques have been proposed to combine the Petri nets with other modeling techniques such as SI nets [8], Expert system [9], and meta rules [10] in order to model high level control systems.

There are two objectives in this paper.

- To propose Conserved nets which are a subclass of Petri nets in order to facilitate the modeling, analysis, and simulation of Petri nets for FMSs.
- To demonstrate the design procedure of a high level decision system which can be incorporated into a Petri net model developed in this paper.

Conserved nets are proposed to model hardware components of FMSs easily and to ensure the conservativeness of a Petri net model. From the structural properties of Conserved nets, the liveness can be easily checked. The high-level control systems which are responsible for resolving conflicts in a Petri net model is constructed from the analysis results of the Conserved net model. Through a simulation of a Petri net model, useful information such as performance measures of a system, and detailed movements of parts is obtained.

A Petri Net for Modeling and Simulation of FMSs

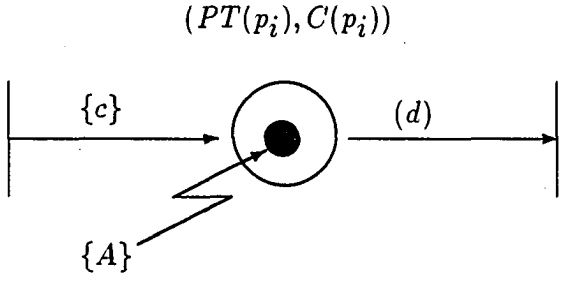
A Petri net is defined formally as the tuple $W = (P, T, A, M)$, where P is the set of places $(p_1, p_2, \cdots, p_n)$, T is the set of transitions $(t_1, t_2, \cdots, t_m)$, and A, M are functions. M is marking of P and the number of tokens in p_i is represented as $M(p_i)$. The set of $N = P \cup T$ is called a node set and an element of $n_i \in N$ is called

a node. The connection relationship between node n_i and node n_j is represented by $A(n_i, n_j)$. If a directed arc connects from n_i to n_j the value of $A(n_i, n_j)$ is 1. Otherwise the value of $A(n_i, n_j)$ is zero.

In addition to ordinary Petri nets, a number of Petri net families have been proposed to model complex systems by extending Petri nets. We added several elements to increase the modeling power of a Petri net, and they can be exploited for the simulation of FMSs.

1. Each place is a capacitated, timed place.
2. Each token is identified as an individual object, and belongs to a certain type.
3. Each output arc of a transition has attached to it a set of token types to flow.
4. Each output arc of a place can have attached to it a predicate for a decision on token movement.

To model a system with this Petri net, four types of specifications are necessary: specification of places, output arcs of places and transitions, and tokens (see Figure 1.1). A place represented by a circle has two attributes, time and capacity represented as $PT(p_i)$ and $C(p_i)$ respectively. A place has its own capacity to allow maximum number of tokens. When an arrived token in a place needs a time delay, the time is imposed on the token immediately. The tokens represented by dots are flow objects and resources in FMSs. Each token belongs to a certain class (token type) such as part, pallet, machine, AGV. Movement of some types of tokens in a net can be controlled by a token control system (high-level control system will be considered as a token control system). Each token type can assume several attributes.



$PT(p_i)$: Processing time of p_i.
$C(p_i)$: Capacity of p_i.
$\{c\}$: A set of token types.
(d) : Decision specification.
$\{A\}$: Token attributes.

Figure 1.1: Specification of Petri nets for modeling and simulation of FMSs

For example, part tokens need attributes such as routing and processing times. Also, resource tokens such as machine, robot, AGV, and man can contain a status attribute. Directed arcs represented by arrowed arcs are classified into output arcs of a transition and output arcs of a place. An output arc of a transition is specified to allow the flow of specific set of tokens. That is, tokens are combined and divided at a transition according to the specification of token flow attached to the output arcs of the transition.

An output arc of a place can be specified to define the decision choices for a token movement. In Figure 1.2, if there is a token in place p_1, it can move to the transition t_1 or t_2. The predicates (a) and (b) attached to each arc are related to decisions of token movement. For example, the state of a machine token can be either a success or a failure depending on whether the machine is running or down for repair. According to the state of the token, the token movement is determined.

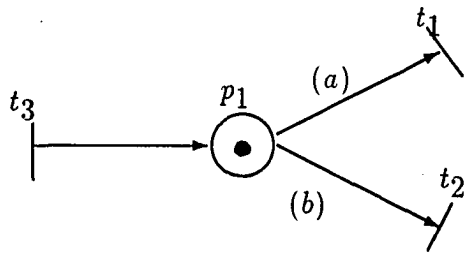


Figure 1.2: Specification of decision arcs

If predicate (a) is "success(p_1)" and (b) is "failure(p_1)" respectively, the token with success state in p_1 will move to t_1. When a predicate is not specified to the output arcs of a place, a token in a place will select one of output arcs randomly (i.e., select a transition to fire arbitrary).

Marking

A marking, M, is an assignment of tokens to the places of a Petri net. The number and position of tokens may change during the execution of a Petri net. Note that when two different tokens are combined and marked in a place, the total number of tokens in a net decreases by one due to the combination of two tokens. Even if the combined token is treated as one token in a net, however, it contains the two individual tokens. To identify the marking of individual tokens, $M_k(p_j)$ is defined to represent the number of token type k in a place p_j.

The net marking is restricted by the capacities of places. A marked token is classified as a token in *processing state* or a token in *waiting state*. When a token is assigned to a place which needs time delay, the token becomes the processing state instantaneously. After finishing the imposed processing duration, the token is in waiting state. $M_p(p_j)$ and $M_w(p_j)$ are the numbers of marked tokens in processing

state and in waiting state at place j respectively. It is obvious that $M(p_j) = M_p(p_j) + M_w(p_j)$.

Enabling

A transition t_i is enabled when the following conditions meet:

1. the input places connected by directed arcs have more than one token in waiting state, i.e., $M_w(p_j) \geq A(p_j, t_i)$ for all input places p_j of t_i, and
2. if fired, the capacity of its output places will not be exceeded, i.e., $C(p_k) - M(p_k) \geq A(t_i, p_k)$ for all output places p_k of t_i.

Transition firing

The enabled transition can fire instantaneously with the transition firing rules. When a transition t_i fires, the following events occur concurrently:

1. For all input places connected with directed arcs, the involved tokens are removed, and $M(p_j) = M(p_j) - A(p_j, t_i)$ for all input places p_j of t_i.
2. At the transition, the gathered tokens are combined or divided according to the specification of output arcs of the transition.
3. For all output places connected by directed arcs, the token is added, and $M(p_k) = M(p_k) + A(t_i, p_k)$ for all output places p_k of t_i.

Conserved Nets

Originally, the marking of tokens under transition firing rule is based on the deletion and creation of tokens. When a transition fires, tokens in the input places

are deleted and new tokens are created in the output places of the transition. In modeling an FMS, tokens represent resources or jobs in the system. These tokens are flow objects in the system, and must be conserved in a net. Rather than being based on the creation and deletion of tokens, the transition firing rule needs to consider the token movement such that tokens flow in a net without any transformation. After all, a useful Petri net model should have the conservativeness property from the following facts as discussed in [11].

- The number of resources is constant over time.
- In a closed queuing system, the number of jobs is constant.
- In an open queuing system, a job token that enters in the system is conserved until it leaves the system.

Originally, a marked Petri net $W = (P, T, A, M)$ is said to be strictly conservative [12] if

$$\sum_{i=1}^{n} M(p_i) = \text{constant, for any reachable marking } M.$$

Since it is common that, in a Petri net model, several tokens are combined into one token, and a combined token is divided into several tokens at a transition, strict conservativeness is not desirable. As a result, weighting vectors could be defined to allow more broad terms of conservativeness ([12], pp. 82): the marked Petri net is said to be conservative if the weighted sum of all reachable markings is constant. Alternatively we may use a more appropriate definition of conservativeness which can be applicable to Petri nets for the simulation of FMSs. When each original token is identified as an individual object, a combined token can also be identified as a token

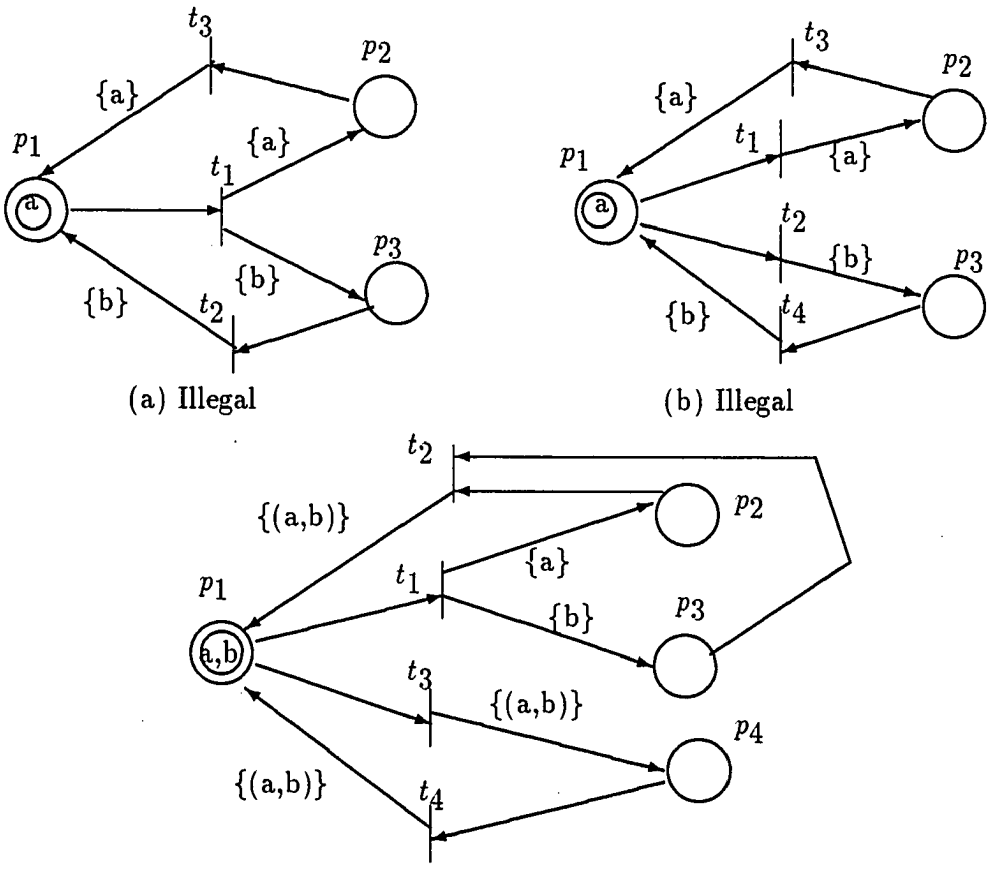
with several original tokens. As defined before, let $M_k(p_j)$ be the number of token type k in a place p_j. Then, we develop a following definition of conservativeness.

Definition 1.1: A marked Petri net $W = (P, T, A, M)$ is conservative in terms of each token type if

$$\sum_{i=1}^{n} M_k(p_i) = \text{constant, for any token type } k \text{ and any reachable marking } M.$$

This definition says that a marked Petri net is conservative when token flows at every node are conserved without any transformation in terms of each token type.

Our objective is modeling a Petri net which has the conservativeness property. This is possible by specifying allowable token flows at every arc between two nodes, and by properly assigning initial tokens in a net. Our Petri net allows token flow specification at the output arcs of a transition. The specification of token flows allows a set of token types through arcs. For example, the specification of token flow, $\{(a, b), c\}$, allows either a combined token of a and b type or a c type token alone. Note that the combined token is represented as a tuple of individual tokens. Let's consider several legal and illegal Petri nets in view of conservation of token flows. In Figure 1.3-(a), place p_1 has a token with token type a. By firing t_1, the token moves to p_2, but a new token with token type b is required to be created in order to be marked in p_3. It prohibits the Petri net from conservativeness. Note that, even if the token in p_1 is a combined token with a and b type, the conservativeness will not be satisfied. Likewise an a type token is deleted and a b type token is created by firing t_2 in Figure 1.3-(b). The Petri net in Figure 1.3-(c) has conservativeness since the deletion or creation of tokens is not required by firing any transition.



(a) Illegal

(b) Illegal

(c) Legal

Figure 1.3: Conservativeness of Petri nets

To have conservativeness, a Petri net model must be modeled with certain restrictions. These restrictions and requirements for conservativeness will be presented. For the prerequisite requirements, consider a Petri net without marking of tokens in a net. Thus, in the following, a net with marking and a net without marking will be distinguished as defined in [1].

Definition 1.2: A Petri net $W = (P, T, A, M)$ is called a Petri net system, and $G = (P, T, A)$ is called an underlying net (simply called a *net*) of a Petri net system W.

Before deriving the modeling rule for a net with the specification of conservative token flows, we will explain the restrictions on Petri net modeling and determination of token flows at each node. We call the nets with specification of conservative token flows Conserved nets which are a subclass of Petri nets. For the Conserved nets, two basic restrictions were set.

- Tokens with the same type are not allowed to be combined.
- Multiple arcs between two nodes are not allowed.

In a Conserved net, only a disjoint set of token types are combined at a transition. Also, there should be at most one arc between any two nodes. These restrictions do not decrease the modeling power of a Petri net. If it is necessary to combine tokens of the same type (e.g., combination of the same kind of resources to perform an operation), the tokens may be further classified into different types. By classifying token types in more detail, combinations of tokens of the same type can be avoided. By not allowing multiple arcs in a net, there is at most one arc between any two nodes.

Four kinds of token flows occur in a net. The possible token flows at each node are determined by examining the specification of token types attached to the output arcs of transitions. Let ${}^{\bullet}F_{n_i}$ and $F^{\bullet}_{n_i}$ be possible input token flow and output token flow at a node n_i in a net $G = (P, T, A)$.

1. Input token flows at a place.

   The possible input tokens at a place are determined by the union of token sets specified at the input arcs of the place. If a place p has n input arcs and the set of allowable token flows, a_i, is specified at the i th input arc, the possible input tokens at p is determined as

   $$ {}^{\bullet}F_p = \cup_{i=1}^{n} a_i $$

2. Output token flows at a place.

   When a token (combined or original) resides in a place p, it moves along the output arcs of the place without any transformation (note that there is at most one arc between any two nodes), i.e., $F^{\bullet}_p =^{\bullet} F_p$

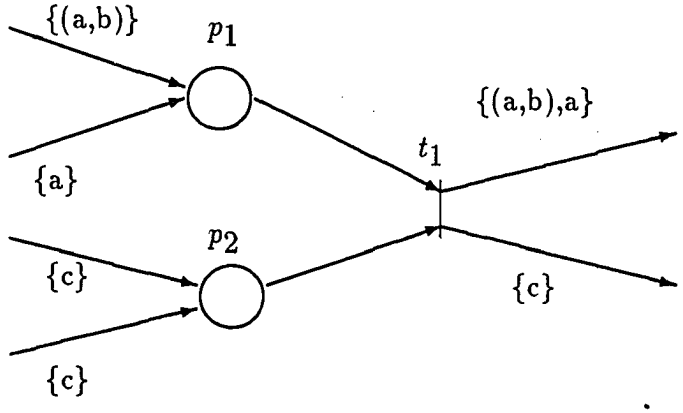
3. Input token flows at a transition.

   The possible input tokens at a transition are determined by the product of token sets from the output token flows of input places. If a transition t has n input arcs (i.e., n input places), and possible output tokens at the i th input places is $F^{\bullet}_{p_i}$, then the possible input tokens of t is:

   $$ {}^{\bullet}F_t = F^{\bullet}_{p_1} \times F^{\bullet}_{p_2} \times \cdots \times F^{\bullet}_{p_n} $$

4. Output token flows at a transition.

   The possible output tokens at a transition are determined by the product of



Node	Possible input tokens	Possible output tokens
p_1	{(a,b)} ∪ {a}={(a,b),a}	{(a,b),a}
p_2	{c} ∪ {c}={c}	{c}
t_1	{(a,b),a}×{c}= {(a,b,c),(a,c)}	{(a,b),a}×{c}= {(a,b,c),(a,c)}

Figure 1.4: Determination of token flows at nodes

token sets specified at output arcs of that transition. When a transition t has n output arcs, and each arc has attached to it a set of token types a_i, then the possible output tokens at t is:

$$F_t^{\bullet} = a_1 \times a_2 \times \cdots \times a_n \cdot$$

From the above results, the possible token movements at each node can be determined. An example in Figure 1.4 illustrates the determination of possible token flows at nodes. To guarantee conservativeness in a net, the input and output token flows at each node should be same. Finally, we develop the following definition of

Conserved nets.

Definition 1.3: $G = (P, T, A)$ in which the specification of token flows is attached to the output arcs of transitions is called a Conserved net if the following conditions hold in the net:

1. $A(n_i, n_j) = 1$ or 0, for any pair of nodes n_i and n_j.

2. When a transition has more than one input place,

$$\text{any element of } {}^{\bullet}F_{p_i} \neq \text{any element of } {}^{\bullet}F_{p_j}$$

where p_i and p_j are any pair of input places.

3. ${}^{\bullet}F_t = F_t^{\bullet}$, for any transition t.

Structural Characteristics of Conserved Nets

In this section, important characteristics of Conserved nets will be presented. Before we come to that, some basic definitions in Graph theory will be adopted.

Definition 1.4: In a net, a sequence of places and transitions, $p_1 t_1 p_2 t_2 \cdots p_n$, is a *directed path* from p_1 to p_n if transition t_i is both an output transition of place p_i and an input transition of place p_{i+1}, for $1 \leq i \leq n-1$.

Definition 1.5: In a net, a *directed path* from p_1 to p_n is a *directed circuit* if p_1 equals p_n.

Definition 1.6: A subnet G' of $G = (P, T, A)$ is defined as $G' = (P', T', A')$ where $P' \in P, T' \in T, A' \in A$. All places in a subnet should be able to mark a particular token type. Similarly, a subsystem $W' = (P', T', A', M')$ of $W = (P, T, A, M)$ is

defined by adding marking $M' \in M$ to the underlying subnet $G' = (P', T', A')$.

Definition 1.7: A subnet $G' = (P', T', A')$ is a closed subnet if all transitions connected with P' in G are T'.

From the definition of Conserved nets, the following characteristics are obtained.

Property 1: A Conserved net can be decomposed into subnets for the flow of each token type.

Remark: Initial tokens in a Conserved Petri net system are also decomposed into original token types which can be assigned to the corresponding subnets. If decomposed tokens cannot be assigned to the subnets, the Petri net system is a false model.

Property 2: A decomposed subnet of a token type flow is a strongly connected, closed subnet, and consists of several directed circuits.

Property 3: When two decomposed subnets of different token flows share common paths, the paths start and end with transitions.

Property 4: When two directed circuits in a subnet of a token type flow share common paths, the paths start and end with places.

The conservation of token flows requires the subnet for a token type to form circuits. If paths of a token do not form circuits in a net, then conservation will not be ensured. Therefore, it is necessary that subnets for every token type flow are constructed to form circuits. This requirement is natural logic in the modeling of

FMSs consisting of sets of shared resources and jobs (also, it is a basic logic in Activity Cycle Diagram [13]). It is not surprising that a number of proposed techniques for Petri net modeling of manufacturing systems exploited resource activity cycles [5].

When a Conserved net can be decomposed into two subnets for each token flow, they are sharing common paths, that is, they share common sequences of transition, arc, places. To ensure the conservation property in a net, a common path starts and ends with transitions because two different types of token can be combined only at a transition and this combined token can be divided into original tokens only at a transition.

In a subnet several directed circuits are combined, sharing common paths with each other. When the same token type is not allowed to be combined, there cannot be a transition which has more than one input place. That is, each transition in a subnet has only one input place and one output place (usually called a state machine). When two directed circuits are combined, therefore, the common path starts and ends with places.

Liveness of Conserved Petri Net Systems

Dead-lock in a Petri net system occurs when there are transitions which cannot fire. A transition is live if it is not dead-locked. In the analysis for liveness of conserved Petri net systems, subnets and subsystems of Conserved nets will be considered. Hereafter, when we refer to a subnet, it is a closed subnet of a token type flow decomposed from a Conserved net. In addition, we assume that combined initial tokens can be decomposed into individual token types which can be assigned to the corresponding subnets. We will follow the formal definition of liveness referred to in

the literature [12].

Definition 1.8: A transition t of a marked Petri net system $W = (P, T, A, M)$ is said to be live if and only if , for all reachable markings M, there exists a sequence of transition firings which results in a marking in which t is enabled. A Petri net system is said to be live if all its transitions are live.

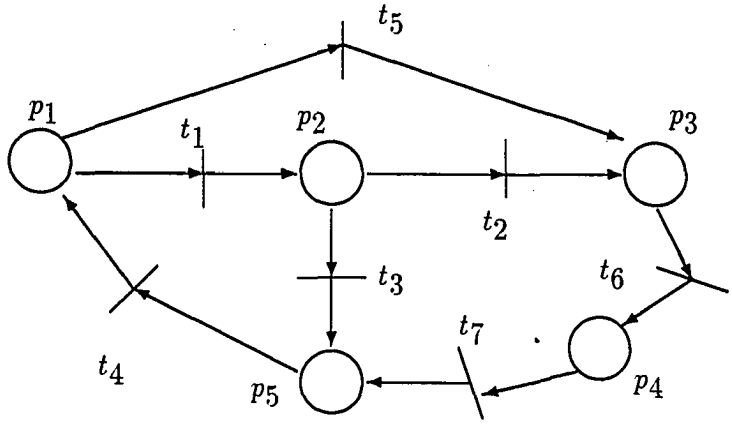
We develop definitions for two types of subnet/subsystems in Conserved Petri net systems.

Definition 1.9: A subnet/subsystem in which output arcs of places have no decision specification is called *r-net/r-system*, and a subnet/subsystem in which output arcs of places have a decision specification is called *d-net/ d-system.*

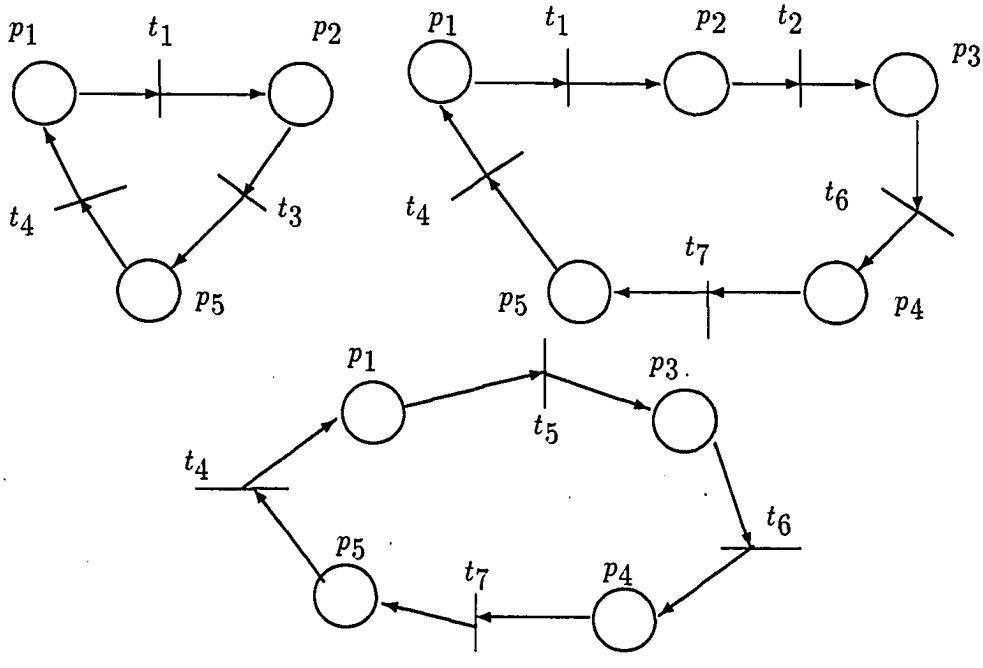
When a place in a *r-system* has more than one output arc, the marked token in the place will move to any one of the arcs randomly whenever the connected transition meets enabling conditions. But, in a *d-system*, a marked token in a place which has more than one output arc must move along one of the output arcs according to the decision specifications attached to the arcs. At below, we develop three propositions concerning liveness conditions of Conserved Petri net systems:

Proposition 1.10: An *r-system*, $W = (P, T, A, M)$, is live if and only if the number of tokens in the system, $\sum_{p_i \in G} M(p_i)$, is greater than zero and less than $\sum_{p_i \in G} C(p_i)$, where G is a underlying net of W.

Proof: It is clear that a *r-system* (Figure 1.5) is not live if the system does not contain a token. When a place in a *r-system* contains tokens, an output transition of the place can be enabled only if $M(p_i)$ of an output place, p_i, of the transition is less

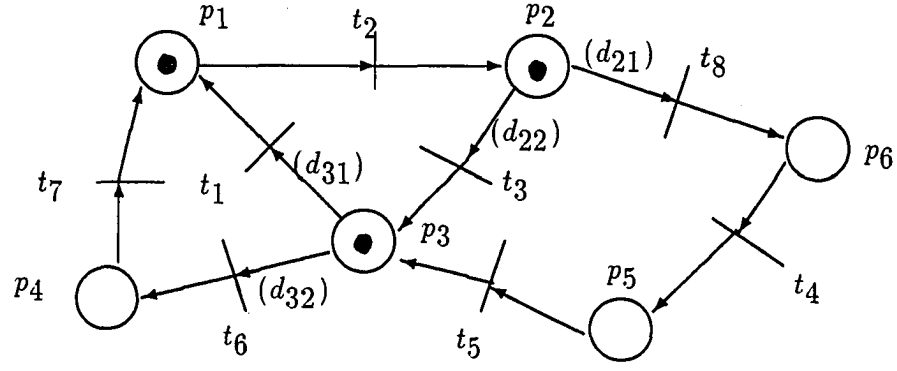


(a) A *r-net*

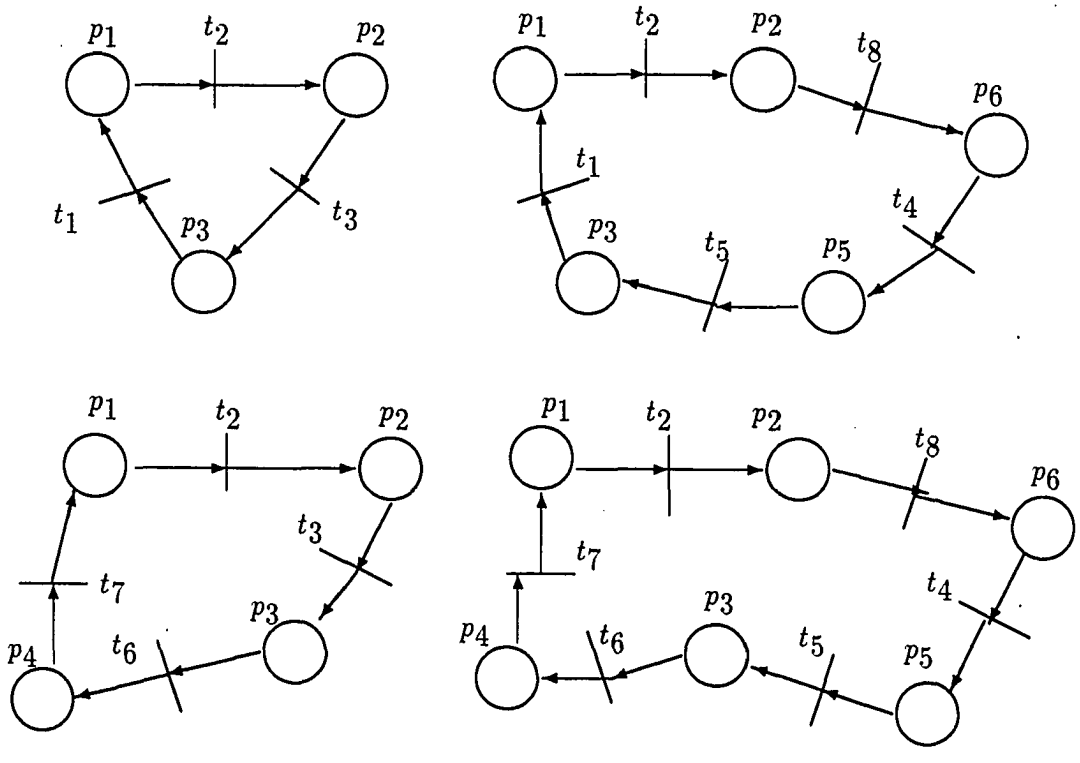


(b) Circuits of a *r-net* (a)

Figure 1.5: The structure of a *r-net*



(a) A *d-net*



(b) Circuits of a *d-net* (a)

Figure 1.6: The structure of a *d-net*

than the capacity, $C(p_i)$. Note that every transition in the *r-net* has only one output place and one input place. If every place, p_i, in the *r-system* W is marked with $M(p_i)$, which equals to $C(p_i)$, there is no transition which can be fired. Otherwise, there is more than one transition to be fired. Assume that *r-system* W is live. Since it is live, there is more than one transition which can be fired. From the transition enabling conditions, the number of tokens, $M(p_i)$, in the output place, p_i, of the enabling transition is less than the capacity, $C(p_i)$.

A *r-net* is a state machine with finite capacity. Without considering capacity, a state machine is live if and only if the net contains at least one token [1]. When considering capacities of places, the capacitated state machine can be reduced into a macro place with the capacity of $\sum_{p_i \in G} C(p_i)$ from the result of Murata and Komoda [14]. Therefore, the proposition is a natural consequence of the previous work on state machine.

Proposition 1.11: A *d-system*, $W = (P, T, A, M)$, is live if the number of tokens in the system is greater than zero and less than $min\{\sum_{p_i \in G_k} C(p_i), i = 1, 2, ..., m\}$, where G_k is the k th directed circuit in $G = (P, T, A)$, and m is the number of directed circuits in G.

Proof: It is trivial that the *d-system* W is not live if the net does not contain tokens. In a *d-net*, several circuits are combined sharing common paths (Property 4). If every place, p_i, in any circuit of W is marked with $M(p_i)$ which equals to $C(p_i)$, there is a possibility of a dead-lock. Consider an example *d-system* in which capacity of every place is one as depicted in Figure 1.6. If the tokens in p_2 and p_3 try to

fire transitions t_3 and t_1 respectively, and a token is marked in p_1, dead-lock occurs because there is no room for the token movement. But, if the token in p_3 can fire the transition t_6, dead-lock can be avoided. In a *d-net* the output arcs of a place are specified with some decision for token movement. Sometimes, a token in a place must fire a specific transition according to the decision arc specification. So, as in the above example, the tokens in p_2 and p_3 may have missions to fire the transitions t_3 and t_1 respectively. Therefore, there is a possibility of a dead-lock if the number of tokens in any circuit in G equals to the sum of capacities of places in that circuit. Consequently, W is live if the number of tokens in a net is greater than zero and less than the sum of capacities of places in any circuit.

Proposition 1.12: When two subsystems W_1 and W_2 which are live are combined sharing a common path which starts and ends with transitions, the combined system is live if and only if the following conditions are avoided:

(i) non-sharing places of a subsystem are not marked with tokens, and

(ii) all non-sharing places in the other subsystem are marked with tokens of the same number as the capacity.

Proof: From the reduction rule of a state machine by Murata and Komoda [14], the combined net can be reduced into three macro place and two transition. Consider an example net in Figure 1.7. The reduced net consists of two non-sharing places (P_1 and P_3), one sharing place (P_2), and two transitions (t_1 and t_2). The capacity of a macro place is represented as the sum of capacities of places included in the macro place. As shown in the Figure 1.7-(d), each reduced subsystem forms a simple circuit. Since two subsystems are live there is more than one token in each circuit.

The following conditions for dead-lock hold from the transition enabling rules of a Petri net.

1. The transition t_1 cannot be enabled if and only if

   1.1 $M(P_1) = 0$ or $M(P_3) = 0$, or

   1.2 $M(P_2) = C(P_2)$.

2. The transition t_2 cannot be enabled if and only if

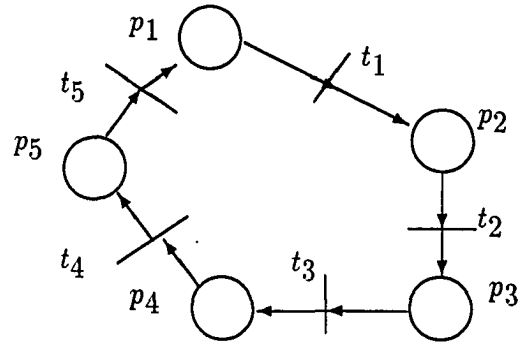
   2.1 $M(P_2) = 0$ or

   2.2 $M(P_1) = C(P_1)$ or $M(P_3) = C(P_3)$.

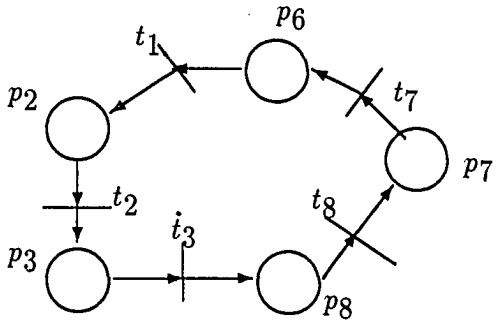
To be live, transition t_1 or t_2 should be enabled. If t_1 is fired, then $M(P_2) > 0$, and $M(P_1)$ and $M(P_2)$ are less than $C(P_1)$ and $C(P_3)$ respectively. Therefore, transition t_2 can be enabled. If t_2 is fired, then $M(P_2) < C(P_2)$, and both $M(P_1)$ and $M(P_3)$ are greater than zero. Therefore, transition t_1 can be enabled. Consequently, if both t_1 and t_2 cannot be fired at the same time, then dead-lock occurs. Both transitions cannot be fired if and only if the combination of the above two conditions holds. Only the combination of (1.1) and (2.2) holds under the condition that the number of tokens in each cycle is greater than zero and less than total capacity of places in each cycle. Therefore, if and only if either (1) $M(P_1) = C(P_1)$ and $M(P_3) = 0$, or (2) $M(P_3) = C(P_3)$ and $M(P_1) = 0$, dead-lock occurs.

Remark: A Conserved Petri net system is conservative if it is live.

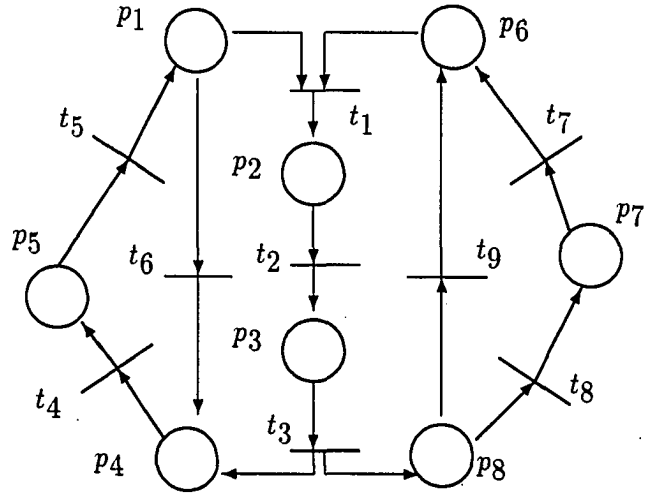
It is easy to see that a Conserved Petri net system has conservativeness.



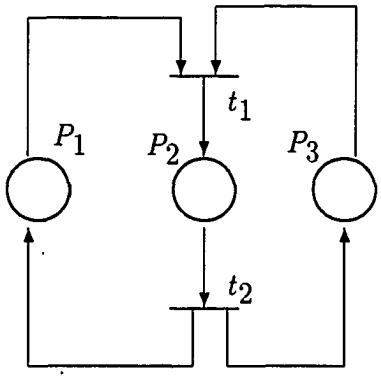
(a) A subnet G_1



(b) A subnet G_2



(c) A net combined with G_1 and G_2



(d) Reduction of net (c)

Figure 1.7: The combination of two subnets

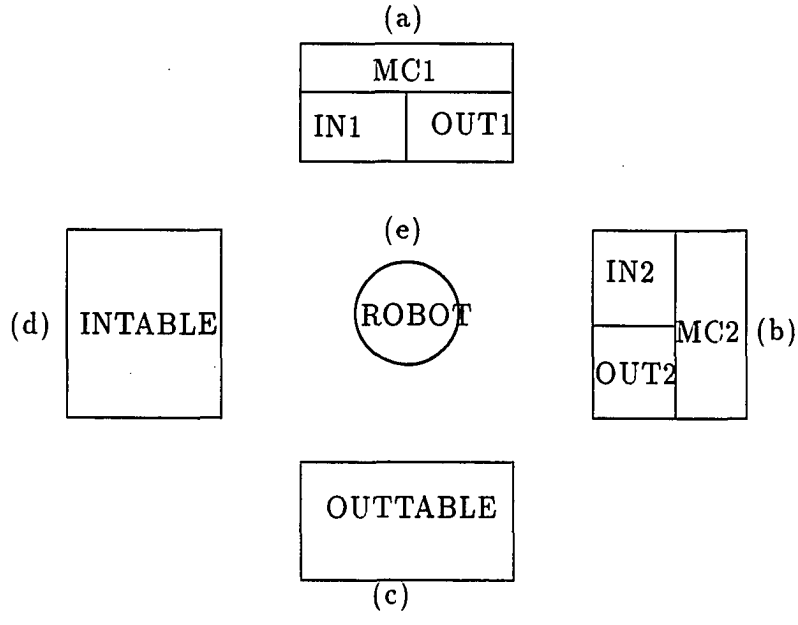


Figure 1.8: A machine center with a robot

An Example: A Machine Center with a Robot

The modeling concept of Conserved nets and high-level control rules will be explained with an example machine center referred in the paper of Gentina and Corbeel [13]. The machine center is composed of two transfer conveyor benches designed for loading and unloading (INTABLE and OUTTABLE), two autonomous machines, MC1 and MC2, with two transfer benches each (IN and OUT), and a robot which distributes parts between the input, output transfer conveyor benches, and two machines (Figure 1.8). We assumed that two types of parts are introduced alternately into the machine center as in Table 1.1. Each part has its own routing: part type 1 is machined MC1 first, then MC2, and part type 2 is machined MC2 first, then MC1.

Table 1.1: Job routings in a machine center

Part type	Routing	Process time (min)	Part mix
1	d,a,b,c	2,10,15,2	1/2
2	d,b,a,c	2,10,15,2	1/2

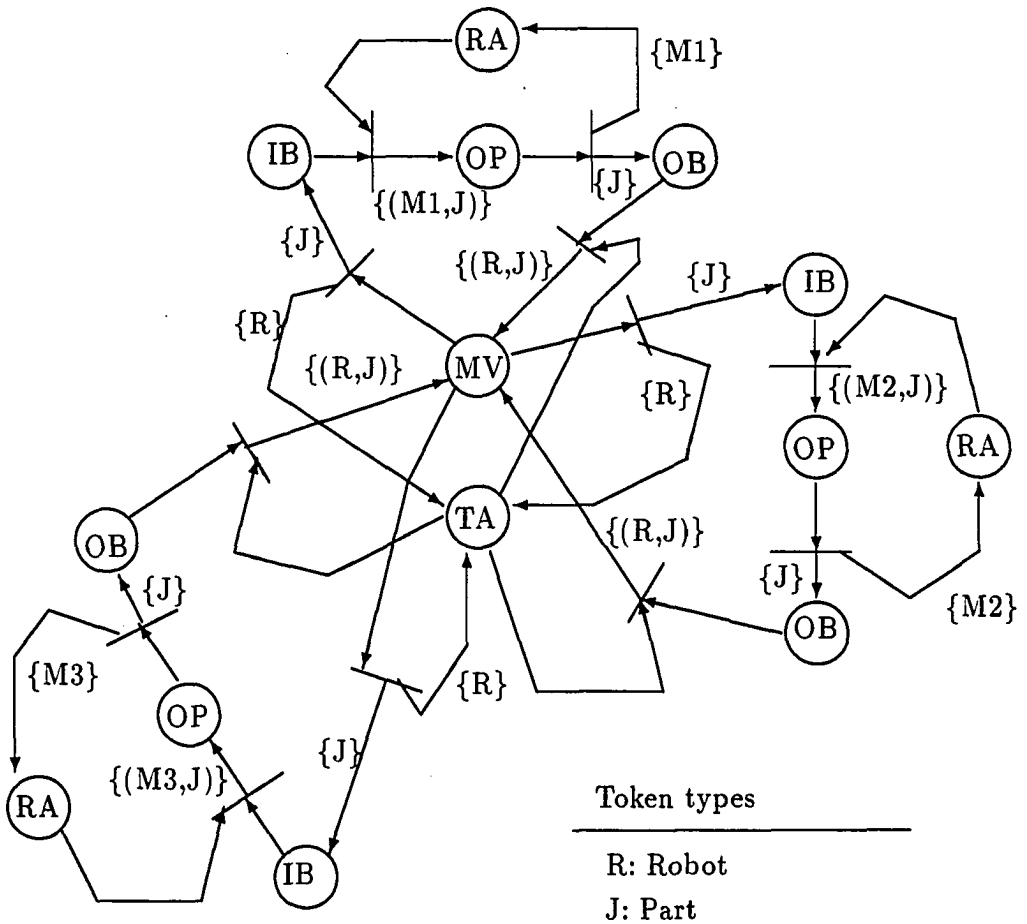
* Robot pickup time: 0.5 min.
* Robot delivery time: 0.5 min.

Figure 1.9 shows a Petri net model of the machine center using a Conserved nets. To make a Conserved net, we assume that the system forms a closed queuing network such that the number of parts is constant in the system. There are five types of tokens, i.e., part, robot, MC1, MC2, and LOAD/UNLOAD. In this Conserved net, we have five subnets for each token type, and all places have capacity of one. From the liveness conditions of Conserved nets, the following results can be obtained.

- Each subnet of a resource token (i.e., MC1, MC2, or LOAD/UNLOAD), is an *r-net*, and includes two places (i.e., the capacity of the subnet is two). Since only one token can be assigned in each subnet, the subsystems are live (Proposition 1.10).

- The subnet of the robot token is a *d-net*. The output arcs of places in the subnet has decision specifications regarding the robot movement. In the subnet, 9 circuits containing two places are identified. Since only one robot token is in the system, the subnet is live (Proposition 1.11).

- The subnet of the part token is a *d-net*. Each output arc of a place has attached to it a decision specification according to job routings. In the subnet, three circuits exist, and every circuit has capacity of four. If the number of part tokens is less than four, therefore, the subsystem is live (Proposition 1.11).



Token types

R: Robot
J: Part
M1: Machine 1
M2: Machine 2
M3: Load/unload

Place types

OP: Machine operation
RA: Machine idle
IB: Input buffer
OB: Output buffer
TA: Robot idle
MV: Robot moving

Figure 1.9: A Conserved net model of a machine center

- In the combined net from subnets of part tokens and a machine token (MS1, MS2 or LOAD/UNLOAD), dead lock occurs (Proposition 1.12) if

  (1) RA place of a machine token has no token, and all places of part tokens have tokens, or

  (2) OP place of a machine token has no token, and part tokens are not in the system.

- In the combined net of subnets of part tokens and a robot token, dead lock occurs (Proposition 1.12) if

  (1) TA place of a robot token has no token, and all places of part token have tokens, or (2) MV place of a robot token has no token, and all places of part tokens have not tokens.

The problem in this system is designing a robot control rule to provide an efficient part flow. Sometimes, a dead-lock due to bad part movements occurs. To handle this situation, some error recovery routine may be required. The more desirable method may be to design a sound control rule to avoid the dead-lock phenomenon. In this study, the latter approach will be discussed. From the analysis results, the following strategies for control rules are required to prevent dead-locks in the system.

- The number of part tokens in the system should be more than zero.
- At most one circuit of part tokens have tokens in OB, IB, and OP places in the circuit.
- When a circuit of part tokens has tokens in OB, IB, and OP places, the MV place should not have a part token to be routed to the IB place of that circuit.

- When a circuit of part tokens has tokens in OB, IB, and OP places, the token in the OB should be moved to the next destination as soon as possible.

Figure 1.10 shows an extended Petri net model in which robot movement is described in detail in order to be used for a detail simulation. In addition, the robot control system based on the above results was designed and incorporated with the Petri net model for the simulation. By accomplishing a simulation, it is possible to estimate performance measures such as output rate, flow time, and queue size in the system. In addition, the animation of a Petri net model provides validation of the control rule. In modeling the Petri net, the following logic is employed.

- The Petri net model for robot movements consists of CP, LN, PU, DL, and TA. PU and DL represent the pick-up and delivery process of the robot respectively. TA represents the idle status of the robot. The time taken to move the robot is imposed on the LN place. The output arcs of these places are decision arcs. When a token is in these places, the control system will give a command to the token to resolve conflicts (i.e., to determine the transition to fire).

- Each machine consists of an input and an output buffer (IB, OB), operation (OP), and machine resource available (RA). The part token in the IB place moves to the OP place whenever the machine is available. After finishing processing time in the OP place, the part token moves to the OB place, and a machine token moves to the RA place.

- At each node, the specification of token flow meets the conservation rule (i.e., ${}^{\bullet}F_t = F_t^{\bullet}$) except at the JC and JD places.

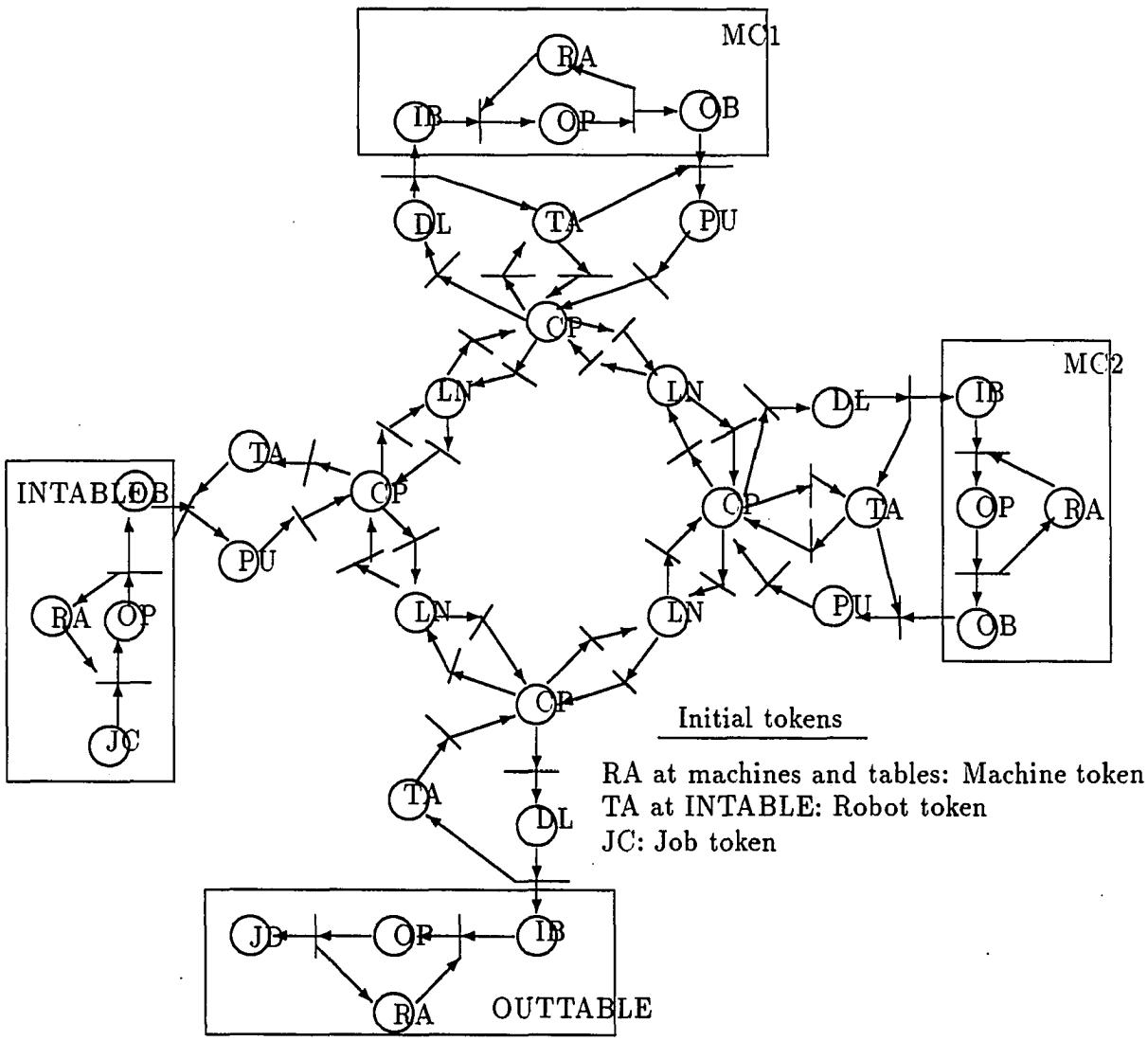


Figure 1.10: A Petri net model for the simulation of a machine center

- Job tokens are created at the JC place in the input transfer bench whenever a job token leaves the JC place in order to guarantee more than one part token in the system. The job token creation rule regarding the next job type is attached to the JC place.
- Job tokens are deleted at the JD place in the output transfer bench.
- Every place has a capacity of one.
- Initially, machine tokens are assigned to each RA places, a robot token to TA place adjacent to INTABLE, and job token to JC place.

To avoid the dead-lock phenomenon discussed before, the robot control rule is modeled by giving a priority to each movement of the robot token. Note that the number in a place name means the each table and machine. The number 1, 2, 3, and 4 refer to MC1, MC2, INTABLE, and OUTTABLE respectively. For example, OB1 is a OB place of the MC1. Each rule is listed in a sequence of high priority.

< Robot control rules >

1. If M(OB1) = 1 and M(OP1) = 1, then send a part token in the OB1 to the next destination.
2. If M(OB2) = 1 and M(OP2) = 1, then send a part token in the OB2 to the next destination.
3. If M(IB1) = 0 and M(RA1) = 1, then send a part token to IB1.
4. If M(IB2) = 0 and M(RA2) = 1, then send a part token to IB2.
5. Send the longest waiting part token in OB's to the next destination.

When the MC1 is blocked because of a full output buffer of MC1, the part in the output buffer has the highest priority. The second rule is applied to MC2. The third (fourth) rule says that when MC1 (MC2) is experiencing the starving of parts, send the part to MC1 (MC2) if possible. This rule will provide the high utilizations of machines. The final rule select a part with the longest waiting time at output buffers. The part movement is possible only when the IB of the next route for a selected part is not full with tokens. The detailed description of control rules, and the method of assigning a command to a robot token will not be given.

To evaluate and validate the proposed control rule, experimental simulation was accomplished. Figure 1.11 shows the output of the simulation. The developed robot control rule is considered to be a desirable rule since the simulation results show that:

1. during the simulation time (1440 min), dead-lock did not occur, and

2. the utilization of MC1 and MC2 is 100 %.

In a real system, it will be impossible to achieve 100 % utilization of machines because of random effects such as machine break down and fluctuation of processing times. In this simulation experimentation, however, those factors were not considered.

Conclusion and Remarks

In this paper, a subclass of Petri nets, called Conserved nets, has been proposed to be exploited for modeling, analysis, and simulation of FMSs. The desirable properties of a Petri net model of FMSs can be easily checked by using developed conditions for liveness of Conserved nets. While hardware components of FMSs, including the low level control system, are modeled by Petri nets, the high level control system of

1. Place statistics

Simulated time: 1443.5

No	Type	Machine	No. of pass	Util.	Ave. wait	Ave. queue	Capacity
1	CP		243	0.00	0.00	0.00	1
2	CP		234	0.00	0.00	0.00	1
3	CP		332	0.00	0.00	0.00	1
4	CP		194	0.00	0.00	0.00	1
5	LN		100	0.04	0.00	0.00	1
6	LN		191	0.07	0.00	0.00	1
7	LN		187	0.07	0.00	0.00	1
8	LN		95	0.03	0.00	0.00	1
9	TA	IN	98	0.00	8.02	0.55	1
10	PU	MS1	95	0.03	0.00	0.00	1
11	DL	MS1	97	0.03	0.00	0.00	1
12	PU	IN	97	0.03	0.00	0.00	1
13	TA	OUT	96	0.00	0.02	0.00	1
14	DL	OUT	93	0.03	0.00	0.00	1
15	TA	MS1	93	0.00	0.58	0.04	1
16	TA	MS2	143	0.00	0.20	0.02	1
17	DL	MS2	97	0.03	0.00	0.00	1
18	PU	MS2	95	0.03	0.00	0.00	1
19	IB	MS1	97	0.00	7.14	0.48	1
20	OB	MS1	95	0.00	1.56	0.10	1
21	OP	MS1	96	1.00	0.00	0.00	1
22	RA	MS1	95	0.00	0.04	0.00	1
23	IB	MS2	97	0.00	11.87	0.80	1
24	OB	MS2	95	0.00	1.24	0.08	1
25	OP	MS2	96	1.00	0.00	0.00	1
26	RA	MS2	95	0.00	0.07	0.01	1
27	IB	OUT	93	0.00	0.00	0.00	1
28	OP	OUT	93	0.13	0.00	0.00	1
29	RA	OUT	93	0.00	13.52	0.87	1
30	JD	OUT	93	0.00	0.00	0.00	1
31	OB	IN	98	0.00	14.66	0.10	1
32	OP	IN	99	0.14	12.58	0.86	1
33	RA	IN	98	0.00	0.00	0.00	1
34	JC	IN	100	0.00	14.44	1.00	1

Figure 1.11: Simulation output of a machine center

1. No. of job produced

```
Total time  =    1443.50
No of jobs produced - Type 1  =     47
No of jobs produced - Type 2  =     46
Total no of jobs produced     =     93
```

2. Robot token statistics

No. of robot	Moving time with part	Waiting time with part	Moving time without part	Waiting time without part
1	477.50 (33 %)	0.00 (0 %)	96.00 (7 %)	870.00 (60 %)

Figure 1.11 (Continued)

FMSs is modeled separately in order to resolve the conflicts in the Petri net model. The modeling logic of Conserved nets and a robot control system is demonstrated with an example machine center. It shows that Conserved nets well represent the hardware components of FMSs. Also, useful information for the design of control systems can be easily obtained from the analysis of a Conserved net model. Finally, the simulation of a Petri net model provides a useful tool to validate control systems and the configuration of hardware components in an FMS.

The structure of software for modeling and simulation of Conserved nets was not described here. Readers who are interested in it can refer to [15].

References

[1] E. Best and P. S. Thiagarajan, "Some Classes of Live and Safe Petri Nets," in *Concurrency and Nets: Advances in Petri Nets* (Edited by J. Hartmann, K.V. Genrich, and G. Rozenberg), Springer-Verlag, Berlin; New York, 1987.

[2] F. Commoner and A. W. Holt, "Marked Directed Graphs," *J. of Computer and System Sciences*, No. 5, 1971.

[3] M. Hack, "Analysis of Production Schemata by Petri Nets," Technical Report 94, Project MAC, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1972.

[4] B. H. Krogh and R. S. Sreenivas, "Essentially Decision Free Petri Nets for Real-time Resources Allocation," *Proc. IEEE Int. Conf. on Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1987.

[5] C. L. Beck and B. H. Krogh, "Models for Simulation and Discrete Control of Manufacturing Systems," *Proc. IEEE Int. Conf. Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1986.

[6] Meng Chu Zhou and Frank Dicesare, "A Petri Net Design Method for Automated Manufacturing Systems with Shared Resources," *Proc. IEEE Int. Conf. on Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1990.

[7] R. Valette, "Nets in Production Systems," *Lecture Notes on Computer Science*, 255, Springer-Verlag, Berlin; New York, 1986.

[8] Antonio Camurri and Macercello Frixione, "Structured Representation of FMS Integrating SI-NETS and High Level Petri Nets," *Applied Artificial Intelligence*, Vol. 4, 1990.

[9] Javier Martinez, Pedro Muro, and Manuel Silva, "Modeling, Validation and Software Implementations of Production Systems Using High level Petri nets," *Proc. IEEE Int. Conf. on Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1987.

[10] J. C. Gentina and D. Corbeel, "Colored Adaptive Structured Petri Nets: A Tool for the Automatic Synthesis of Hierarchical Control of FMSs," *Proc. IEEE Int. Conf. Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1987.

[11] Y. Narahari and N. Vishwanahham, "A Petri net Approach to the Modeling and Analysis of Flexible Manufacturing Systems," *Annals of Operations Research*, Vol. 3, 1985.

[12] J. L. Peterson, *Petri net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[13] G. K. Hutchinson and A. T. Clementson, "Manufacturing Control Systems: An Approach to Reducing Software Costs," *Robotics and Computer integrated Manufacturing*, Vol. 1, No. 3/4, 1984.

[14] Tomohiro Murata and Norihisa Komoda, "Liveness of Sequence Control Specifications described in Capacity Designated Petri Net using Reduction," *Proc. IEEE Int. Conf. on Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1987.

[15] D. S. Yim and T. A. Barta, "Petri net-based Simulation Tool for the Design and Analysis of FMSs," Working Paper, Department of Industrial and Manufacturing Systems Engineering, Iowa State University, Ames, Iowa, 1991.

PART II.

PETRI NET-BASED SIMULATION TOOL FOR THE DESIGN AND ANALYSIS OF FMSS

Petri Net-Based Simulation Tool for the Design and Analysis of FMSs

D. S. Yim and T. A. Barta

Department of Industrial and Manufacturing Systems Engineering

Iowa State University, Ames, Iowa 50010, USA

Abstract

Simulation has been recognized as an invaluable tool in designing and analyzing FMSs. In this paper, a Petri net-based simulation tool is presented to aid the simulation projects in the manufacturing area. We developed Petri net modeling methodology in order to exploit Petri nets for the simulation of FMSs. While hardware components of FMSs are modeled by hierarchically-classified Petri net objects, real-time control rules in high-level control systems are separately modeled and integrated with a Petri net model so that they resolve conflicts occurring in Petri net execution. The facilities of the developed tool are described. Also, the use of the tool is illustrated via a case study.

Keywords: Petri nets, Simulation, FMSs.

Introduction

Petri nets are a formal, graphical modeling tool well suited to the description of distributed and concurrent systems which exhibit synchronization and cooperation. Because of these capabilities, the Petri nets are a widely used tool for the modeling and analysis of communication systems, computer software and hardware, and manufacturing systems. In addition, Petri nets have been used for the simulation of discrete manufacturing systems. Törn [1] proposed extended Petri nets for the application of discrete-event simulation. He introduced the basic requirements of Petri nets for the purpose of simulation; inhibitor arcs, timed nets, colored tokens, queues, test arcs and interrupt arcs. For an application to the simulation of manufacturing systems, Bruno and Morisio [2] proposed extended Petri nets, Prot net. They developed the simulation tool based on object-oriented programming. Alanche, et al. [3] described the structure of a Petri net-based simulator, called PSI. The PSI consists of a token player, calendar, and statistical functions.

Even if several extensions and tools are developed in order for Petri nets to be exploited in the simulation of manufacturing systems, there exist limitations in representing complex FMSs with Petri nets. Petri nets can be useful in the representation of some features of FMSs, such as the distributed and concurrent nature of processes or the synchronization and conflicting properties among tasks in the use of shared resources [4]. However, it is rather difficult to model high-level control systems in FMSs (e.g., scheduling rules, vehicle dispatching rules) with Petri nets. To reduce this difficulty, several methodologies have been proposed. Martinez, et al. [5] integrated high-level Petri nets with a knowledge-based system; coordinate subsystems with the local controllers are modeled using high-level Petri nets, and scheduling

rules are represented as a knowledge-based system. Similarly, Camurri and Frixione [4] proposed Structured timed colored Petri nets to represent low-level coordinate subsystems, and used SI-nets which are based on the frame-based semantic net for the modeling of the high-level scheduling system.

As briefly discussed, it is necessary to increase the modeling power of Petri nets to model and simulate complicated FMSs. Also, the high-level control systems should be added to Petri net models. This paper presents an approach to exploit Petri nets for successful simulation projects. The main objectives are:

- To develop Petri net objects and the modeling methodology which is suitable for the modeling of hardware components of FMSs, including the low-level controller.
- To develop modeling methodology for high-level control systems in FMSs which are easily incorporated into Petri net models.
- To implement these concepts and methodologies into a computer-aided simulation tool for modeling, animation, and analysis of FMS specifications.

To facilitate simulation modeling, the place and token objects in Petri nets are classified to correspond to hardware components of FMSs. A Petri net model is constructed with these Petri net objects in a top-down fashion. To ensure well-formed Petri nets, Conserved nets [6] in which token flows are specified to guarantee a conservativeness property are presented. Liveness conditions of Conserved nets can be exploited to aid the modeling of a live Petri net model. In addition to Petri net models, high-level control systems are separately modeled, and integrated with the Petri net model to resolve conflicts occurring in the simulation. These ideas are

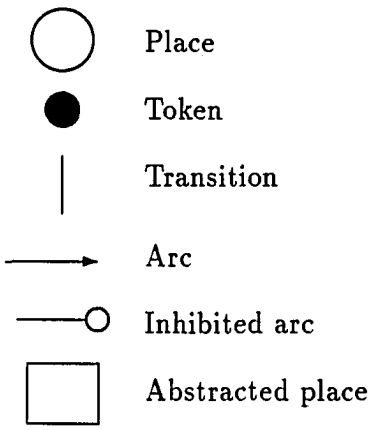


Figure 2.1: Elements of Petri nets

implemented in a computer-aided simulation tool. A simulation procedure with the developed tool is demonstrated with an example FMS.

Petri Nets for Simulation of FMSs

Specification of Petri nets

A top-down modeling procedure that ensures desirable properties in a Petri net model is proposed. Besides the basic elements of Petri nets (i.e., places, transitions, directed arcs, and tokens), inhibitor arcs and abstracted places are included to facilitate the modeling of complicated systems (Figure 2.1). The inhibitor arc connecting between a place and a transition prevents the transition from firing when the connected place has tokens. The top-down modeling methodology allows abstraction of the detailed levels into a concise representation. In this Petri net model, a rectangle represents an abstracted place.

The Petri net used here for the simulation of FMSs allows several attributes in

Petri net elements.

- Each place is a capacitated, timed place.
- Each token is identified as an individual object, and belongs to a certain class.
- Each input arc of a place has attached to it a set of token types to flow.
- Each output arc of a place can have attached to it an expression for the decision on token movement.

To model a system with this Petri net, four types of specifications are necessary: specification of places, input arcs and output arcs of places, and tokens. A place represented by a circle has two attributes, time and capacity. A place has its own capacity to allow the maximum number of tokens. When a token arrives in a place that needs a time delay, the token becomes in processing state immediately. After the imposed time delay, the state of the token changes to a waiting state. Three types of processing times are imposed depending on the situation. A place-attached processing time, mean time between failure (MTBF), and a token-attached processing time are classified considering the characteristics of FMS simulation.

The tokens represented by dots are flow objects and resources in an FMS. Each token is belong to a certain class (token type) represented by a color such as parts, pallets, machines, and AGVs. Some types of tokens are controlled by a high-level control system. Each token type can assume several attributes. For example, part tokens have attributes, routing and processing times. Also, resource tokens such as machines, robots, AGVs contain a status attribute (break-down or not).

Directed arcs represented by arrowed arcs are classified into input arcs of a place and output arcs of a place. An input arc of a place is specified to allow the flow of

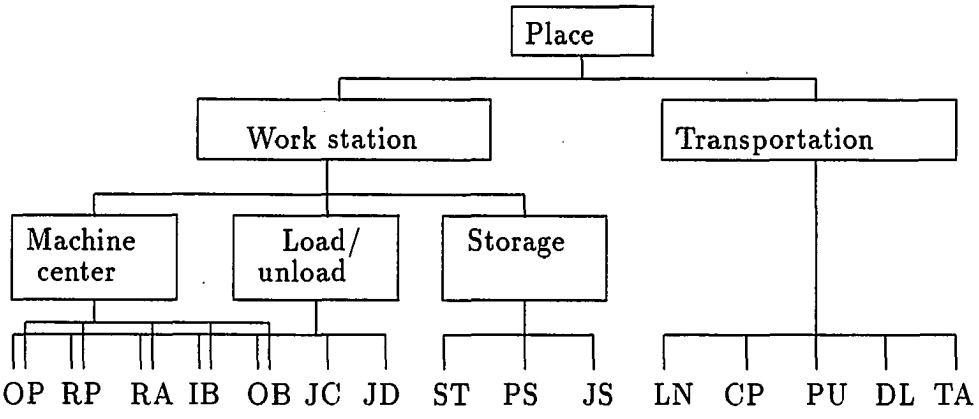


Figure 2.2: Classification of place objects

a specific set of tokens. An output arc of a place can be specified to define decision choice for a transition firing. If decision specifications are not attached to output arcs of a place, a transition is selected randomly among a set of enabled output transitions of the place. In addition to directed arcs, inhibited arcs are represented by a small circle instead of an arrow.

Petri net objects for modeling of FMSs

The places and tokens for the modeling of FMSs were classified and specified hierarchically to facilitate the modeling process of hardware components. The classified places—work station places and transportation places—correspond to hardware components of FMSs (see Figure 2.2 and Table 2.1).

In addition to place objects, token objects are classified into active tokens and passive tokens as shown in Figure 2.3. The active tokens such as AGVs, machines, personnel resources, and robots can move to the next node autonomously, or are controlled by a high-level control system. The passive tokens such as parts, pallets,

Table 2.1: Place objects for Petri net modeling of FMSs

Objects	Mnemonic name	Token types[a] through input arcs	Attributes[b]
Operation	OP	(A,J,P)	MTBF, Process time Decision arc
Repair	RP	(A,J,P)	Repair time
Input buffer	IB	J or (J,P)	Capacity Token link method
Output buffer	OB	J or (J,P)	Capacity Token link method
Resource available	RA	A	
Job creation	JC	No token	Token creation rule
Job deletion	JD	J	
Control point	CP	C or (C,J,P)	Decision arc
Line	LN	C or (C,J,P)	Process time
Transporter available	TA	C	Decision arc
Pick-up	PU	(C,J,P)	Pick-up time
Delivery	DL	(C,J,P)	Delivery time
Storage	ST	(J,P)	Capacity Token link method
Job storage	JS	J	Capacity Token link method
Pallet storage	PS	P	Capacity

a. Token types.
A: Autonomous tokens.
C: Controlled tokens.
J: Job tokens.
P: Passive tokens except job tokens.
(A combined token is represented as a turple of each token type.)
b. The capacity of every place is one except IB, OB, ST, JS and PS.

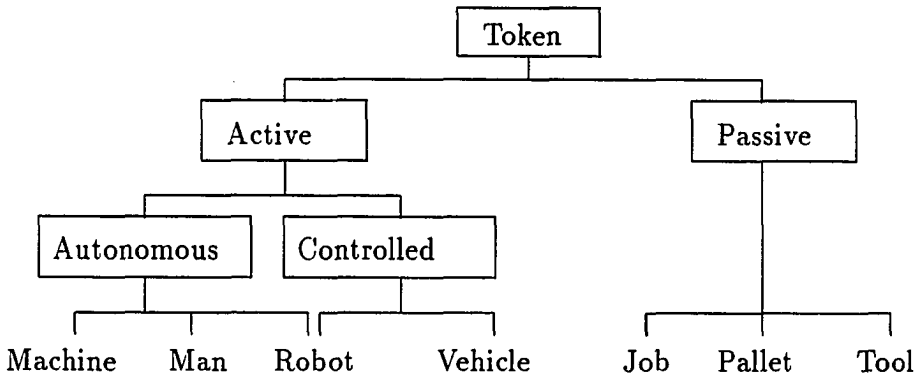


Figure 2.3: Classification of token objects

and fixtures cannot move to other nodes without the combination with active tokens.

The classified places shown in Table 2.1 have the following characteristics:

- All places have a capacity of one to allow the maximum number of tokens except IB, OB, ST, JS, and PS places. The capacities of those places can be specified by users.

- OP, RP, LN, PU, and DL places are timed places. The OP place takes three types of time values: a place-attached, MTBF, and a token-attached time. RP, LN, PU, and DL places takes only a place-attached time value.

- The token flows at each place are conserved except the JC and JD places. While job tokens are created in a JC place, these tokens are deleted in a JD place.

- Input arcs of a place type have specific token types to flow.

  - JC, JD, IB, ST, OB, JS, and PS, places allow only passive tokens.

  - OP, RP, LD, UD, PU, and DL places allow the combination of passive tokens and active tokens.

- RA and TA places allow only active tokens.
- LN and CP places allow either active tokens or the combination of active tokens and passive tokens.

- The output arcs of transportation places, CP, PU, DL, and TA, have attached to them a decision specification of token movements. The next place to move for a token in the place is specified at output arcs of the place. In OP place, there are two types of decision arcs; an arc for a success status and an arc for a breakdown status. When the processing time (place-attached or token-attached time) of a marked place is less than remaining MTBF of the place, the status of the place is success. Otherwise, the status becomes breakdown.

Some places must be specified with token-related rules. JC places need token creation rules. A job arrival pattern at an FMS is usually classified as a static demand or a dynamic demand. The static demand is the case where there is a fixed number of jobs, all having the same arrival time, in an FMS. Under the dynamic demand environment, the jobs are continuously arriving according to some arrival pattern. For the static demand, job types and the number of jobs in each job type can be specified. Several job token creation rules are provided in this simulation tool. They deal with which type of job token is created next; (1) SAME: same type of job token which was created previously, (2) LRJT: The largest remaining job type, and (3) SPT: Shortest processing time. Also, for the dynamic demand, each job type and the corresponding distribution of inter-arrival times of jobs can be specified at a JC place.

IB, OB, JS, and ST places need token-link rules. When a token arrives at these

places, the token is linked in a token list of that place according to the specified rule. This rule corresponds to a machine-to-part allocation rule in FMSs. In this modeling tool, several rules to link a job token into a token list are supplied; (1) PRIO: predefined priority, (2) SPT: shortest processing time, (3) MRN: minimum remaining number of processes, and (4) FIFO: first-in first-out.

Well-formed Petri net model

The token movement in a Petri net system will be well performed without impediments only when the Petri net model is well-formed. In order for a Petri net model to be well-formed, it must have several properties: safeness, boundedness, conservativeness, and liveness. The validation of a Petri net model (i.e., simulation model) and the analysis of the modeled system are possible by examining the properties. A Petri net model of FMSs is required to have the following important properties as discussed in [7].

Safeness and boundedness If places of Petri nets have physical meanings (e.g., buffer, storage, location of vehicles), safeness and boundedness ensure that a modeled system has an absence of overflows. The number of tokens in a place should not exceed the specified capacity. Since our Petri net for the modeling and simulation of FMSs limits the number of tokens in a place (i.e., capacitated place), however, these properties are unnecessary to examine.

Conservativeness If a marked Petri net model is conservative, then the number of tokens of each token type is constant in all reachable markings. A token in our Petri net modeling methodology represents a resource or a job in a system. Therefore,

conservativeness must be met by the following facts.

1 The number of resources is constant over time.

2 In a closed queuing system, the number of jobs is constant.

3 In an open queuing system, a job token which enters the system is conserved until it leaves the system.

In an open queuing system, conservativeness of job tokens is not required. But, once the job token is created, it should not be transformed until it is deleted at appropriate places. The classified place objects provide token creation and deletion places to handle this situation. Therefore, besides the token creation and deletion places, the token flows at every node (i.e., a place or a transition) should be conserved.

Sometimes, in a real system, a part is decomposed into several parts by a certain operation (e.g., metal cutting operation). In this case, the original part can be considered to consist of several decomposable parts. By letting an initial job token be several decomposable token types, the conservativeness of each token type will be maintained.

Liveness Liveness implies the absence of dead-locks in a modeled system. Dead-locks can easily occur in operating a real system. For example, in operating an AGV system, collisions among AGVs and blocking problems are common. Therefore, it is necessary to detect dead-lock of a system in the modeling process.

Modeling rules for a well-formed Petri net

After modeling a system with Petri nets, the desirable properties of the system can be revealed by analyzing the Petri net model. However, the complexity of the model is drastically increased with the number of states and events in a net and the introduction of inhibited arcs. The desirable methodology is, therefore, to impose restrictions on the Petri net modeling process to ensure the required properties a priori. That is the reason that several subclasses of Petri nets (e.g., State machine [8], Marked graph [9], Free-choice nets [10]) and several modeling methods (e.g., resource activity cycle [11], bottom-up modeling by adding arcs step by step [12]) have been proposed. Conserved nets [6] were also proposed to model FMSs by the authors. Conserved nets are a subclass of Petri nets which provide simple analysis of the desired properties. The requirements of modeling a well-formed Petri net model using Conserved net are as follows.

1. At every transition, the token flow should meet the conservation rule (i.e., input token flow = output token flow).

2. Each subnet of active token types and passive token types except job token type should meet liveness conditions 1 and 2 in Appendix.

3. When several subnets are combined sharing common paths, any pair of subnets should meet liveness condition 3 in Appendix.

4. Passive tokens should be combined with active tokens to move to other nodes. Therefore, a subnet of passive tokens should be combined with subnets of active tokens by sharing common paths.

5. A token creation place should have an appropriate token creation function. To be live, there should not be a shortage of job tokens. Also, when a token arrives at a token deletion place, the token should be deleted immediately to prevent overflows of job tokens in a net.

6. When inhibited arcs are introduced into a net, the analysis of properties becomes difficult. These arcs may cause unpredictable dead-locks in the system. It is recommended that inhibited arcs not be used if possible.

Since passive tokens cannot move to the next place without combination with active tokens, a direct concern is liveness in subnets of active tokens which are combined with subnets of passive tokens. When subnets of active tokens are live, the combined Petri net is live if there will not be a shortage and overflow of passive tokens. When passive tokens are resources such as pallets, fixtures, and tools, the number of these tokens should be constant in a net. But, when passive tokens are job tokens, the subnets of these tokens may not be closed nets in that job tokens are created and deleted in the nets. When this open net is combined with subnets of active tokens, the combined net is live if the number of job tokens in the net is less than the total capacity of places for job tokens, and more than or equal to one at any time.

Control Systems in FMSs

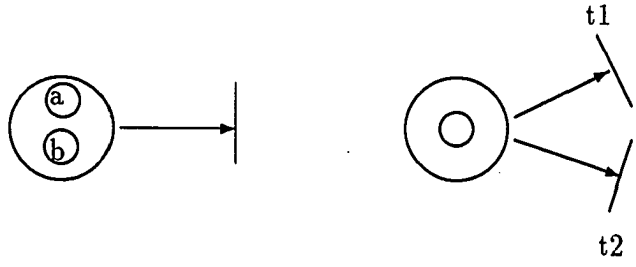
A simulation model for the design and analysis of FMSs contains features for hardware components and control systems. The hardware component model describes the physical elements of FMSs such as work stations, material handling equip-

ments, and storage units. The modeling of these elements is rather easily performed because they are decomposed into manageable elements and modeled with a formal description. On the other hand, the fact that the control logic has an abstract and informal nature makes the modeling of control systems difficult. As the manufacturing environment proceeds toward automated manufacturing systems, the control systems become more sophisticated, needing high accuracy and reliability.

The main purpose of the simulation is to analyze and design an FMS by examining performance measures such as production rate, resource utilization, work in process, and flow time. These measures are greatly affected by three-tier decision rules classified by Suri and Whitney [13]. The two upper decision systems are off-line decision rules in that they do not directly control the FMS hardware components. For a simulation model, sets of alternatives regarding the off-line decisions are provided with input data such as part-mix, system configuration, batch size, and balancing. The third level decision systems are real-time control rules; they analyze the real-time data and directly control the hardware components. Under the third level decision systems, we should decide

- Real-time scheduling of jobs
- Job routing and control of material handling systems.

Our modeling methodology provides a Petri net for the modeling of hardware components. In a Petri net model, several types of conflicts can occur so that a decision about transition firing is required. Usually, two types of conflicts exist in a Petri net model [5]: color conflict and path conflict. The color conflict occurs when there are several tokens in an input place of a transition as shown in Figure 2.4.



Color conflict Path conflict

Figure 2.4: Conflicts in Petri nets

By firing the transition, a token to be involved in the token movement must be selected among the set of tokens in the place. The path conflict occurs when there are several output transitions of a place. When a place has a token, a transition to fire should be selected among a set of output transitions. To resolve these conflicts, some decision is required, and this decision is related to real-time control rules. At below, the modeling methodology of two real-time control rules—job release rule and AGV dispatching rule—is described.

Job release rule

When it is necessary to resolve a color conflict, and a dynamic token selection rule based on the current status of a system is required, the rule will be attached to the corresponding transition. Job release rules are considered to be included in the token selection rule to resolve a color conflict.

Job release rules control the introducing of parts into a system. They determine the timing and sequence of releasing each part into the system. In an FMS, the job release rule is closely related to the available pallets at the load area. Usually, a pallet is restricted to serve certain types of jobs. Therefore, a job cannot be released

when there is no available pallet with a like type. This relationship between pallets and jobs necessitates the classification of job release rules into job selection rules and pallet selection rules. The following rules were included in the simulation tool.

1. Job selection rules
    - SPJL: Load the part which has the smallest proportion of jobs launched.
    - SPT: Load the part which has the shortest total processing time.
    - FIFO: Load the part with first-come first-serve basis.
    - NEP: As each pallet is unloaded, reload it with a like job type if possible.
2. Pallet selection rules
    - LUP: Select the least utilized pallet.
    - LWP: Select the pallet which has the longest waiting time.

In operating a real system, a job release rule may combine the above two selection rules. In this case, the rule of higher priority should be specified. For example, under the job oriented rule, a job is selected first, then a pallet which can serve the selected job is selected. Under the pallet oriented rule, a pallet is selected first followed by the selection of a job which can be loaded with the selected pallet.

Control of material handling systems

When material handling equipments are represented as tokens in a Petri net, these tokens are controllable tokens, and must be given the complete paths to move in the net. To accomplish the movement of controllable tokens, a high-level control

system should have the capability to analyze the current status of the Petri net and historical data, and to assign a path to the controllable token. The path for the token is given so that the path conflict is resolved. At below, we will concentrate on the control of Automated Guided Vehicle (AGV) systems since this system is widely used recently.

In operating an AGV system, three types of selections are required: an idle vehicle, a part to move, and the next process of the selected part. Egbelu and Tanchoco [14] classified the AGV dispatching rules into work center initiated rule and vehicle initiated rule. In the work center initiated rule, a work center which has a part to move to the next operation selects a vehicle among a set of idle vehicles. In the vehicle initiated rule, an idle vehicle selects a work station to serve among the work stations which have parts waiting at output buffers for the next operation.

In the FMS environment, the vehicle initiated rule involves the selection of the part to move and the next process for the selected part among the set of parts and processes simultaneously requesting the service of any vehicle. Under the push rule, an idle vehicle first selects the part among the set of parts waiting for an AGV in output buffers of work stations. Then, it selects the next process for the selected part among the set of alternative processes. But, under the pull rule, the selection sequence is reversed. An idle vehicle first selects a process which is highly demanding a replenishment of parts. Then it selects the part which can move to the selected process among the set of available parts in output buffers.

The following rules were included in the developed AGV dispatching system. Some of these rules were adopted from the literature [14,15,16].

1. Vehicle selection rule

- LIV: Select a vehicle with the longest idle time.
- NV: Select the nearest vehicle.

2. Part selection rule

   - LWT: Select a part with the longest waiting time.
   - MQS: Select a part in output buffer which has the maximum queue size.

3. Process selection rule

   - LIT: Select a work station which is experiencing the longest inter-arrival time of parts.
   - MRIQ: Select a work station whose input buffer has the maximum remaining queue space.
   - MWQ: Select a work station whose input buffer has the minimum queue size in terms of processing time.

Facilities of Petri Net-Based Simulation Tool

The developed simulation tool consists of several subsystems including modeling, executive, and output analysis (Figure 2.5). Figure 2.6 depicts modeling procedure emphasized on user interface. A Petri net graphics editor provides a graphic language for the modeling and simulation instead of a complicated textual programming language. Once the Petri net model is completed, the system automatically transforms the model to the internal representation. The high-level, real-time control rules are modeled, and transformed to a C program by control rule transformer which is based

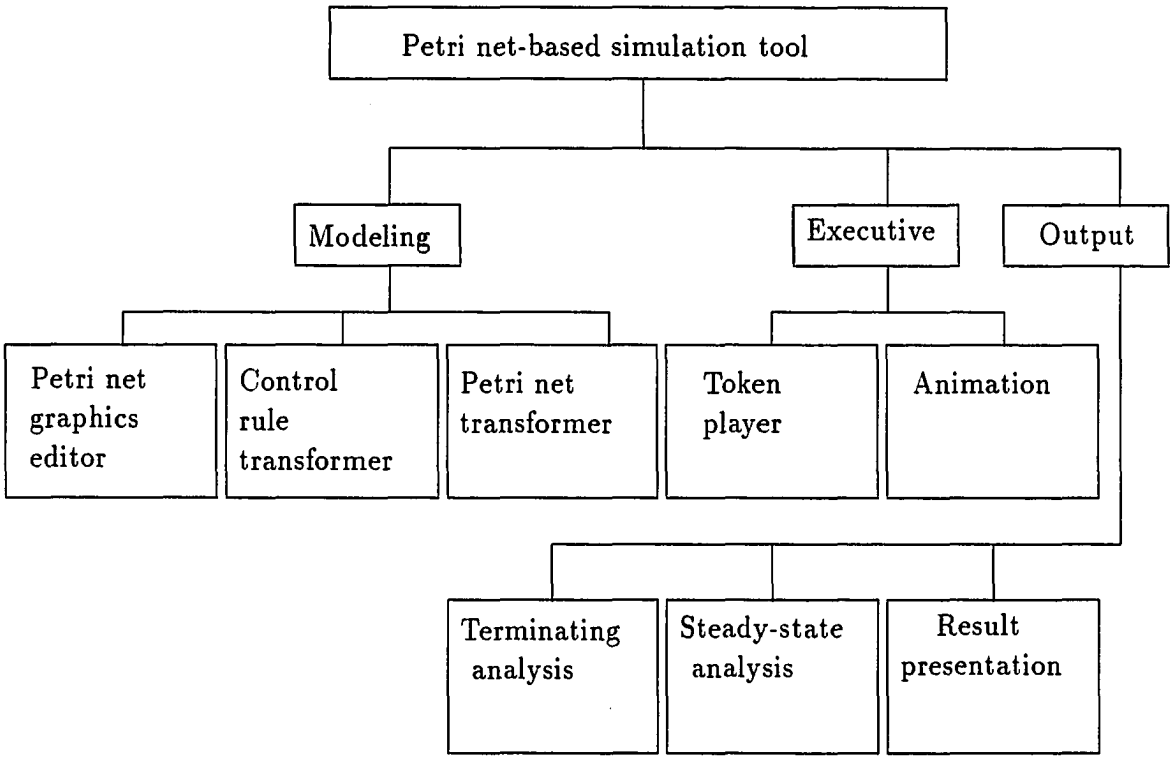


Figure 2.5: The structure of Petri net-based simulation tool

on automatic code generation in order to be compiled and linked with the token player and the animation system.

In addition to the modeling processes, this tool automatically performs output analysis according to user requirements.

Petri net graphics editor

The graphics editor is used to create and modify the Petri net model. Modelers can draw and edit a model by selecting elements of Petri nets and placing them in the desired location. Six elements are provided; abstracted places, places, transitions, directed arcs, inhibited arcs, and tokens. The abstracted place represents an abstrac-

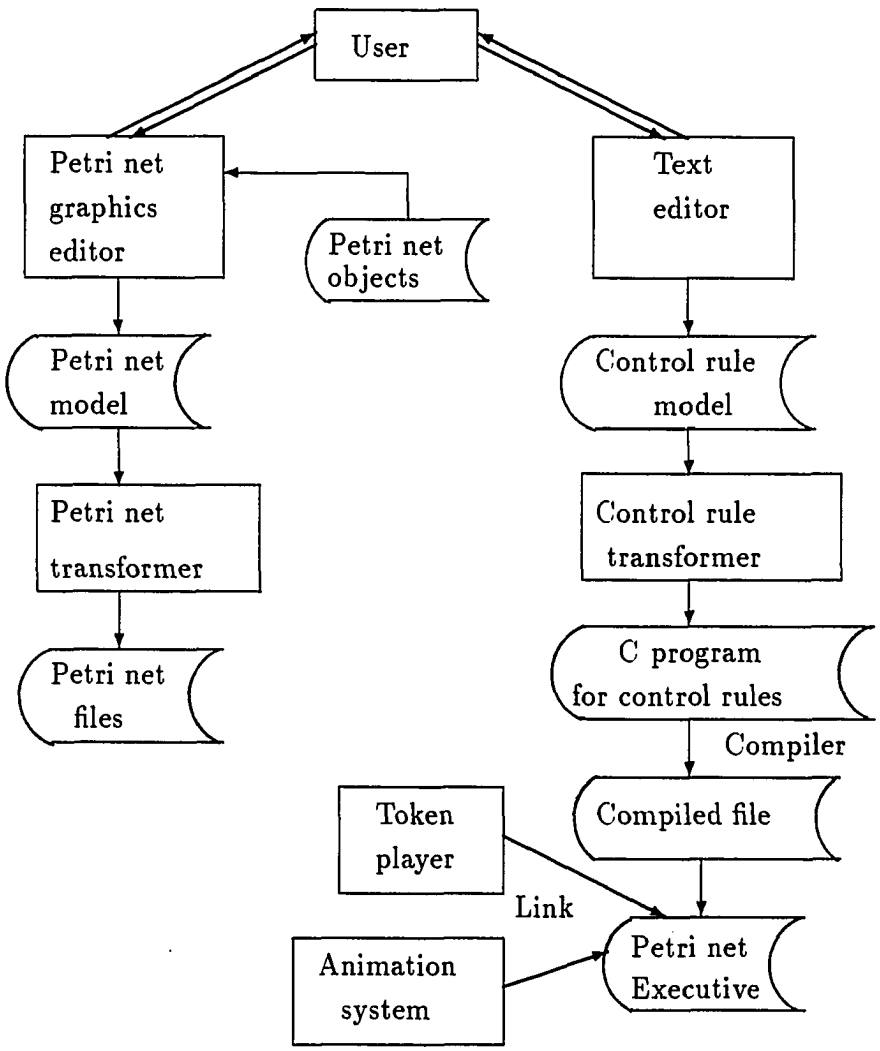


Figure 2.6: Modeling procedure

tion of a Petri net model. The abstracted place will be further completed when going down levels, thus, a top-down modeling approach is accomplished. As a result, a Petri net model consists of several submodels which are hierarchically constructed. Under the Petri net graphics editor, places, token flows, and initial tokens will be specified. In creating a place, place attributes such as place type, capacity, and token link rule can be specified. Also, in creating an arc, the possible token types through the arc are specified.

After creating a Petri net model, the Petri net transformer automatically transforms the model into the internal representation that will be fetched by a token player during simulation execution.

Control rule modeling

To model the high-level control system, a token control language was developed. It consists of predicates, functions, and control statements (Table 2.2). Originally, the language was devised to model AGV dispatching rules. Besides, it is possible to model the control rules for other controllable tokens such as robots. The control rule is modeled in order to give a command to controlled tokens. After the control rule is modeled, the control rule transformer generates a C program to be compiled and linked with the token player and Petri net animation system. In addition, the transition-attached rules such as job release rules can be specified under Petri net graphics editor.

Table 2.2: Elements of token control language

Elements	Usage
Predicate	FULL(P): If $M(P) = C(P)$, then true. EMPTY(P): If $M(P) = 0$, then true. PROCESS(P): If $M_p(P) > 0$, then true. WAIT(P): If $M_w(P) > 0$, then true. where P: place number $M(P)$: no. of tokens in P $C(P)$: capacity of P $M_p(P)$: no. of processing tokens in P $M_w(P)$: no. of waiting tokens in P
Function	PUSH($(P_{11}, P_{12}, \cdots, P_{1n}), k_1, (P_{21}, P_{21}, \cdots, P_{2m}), k_2$) PULL($(P_{11}, P_{12}, \cdots, P_{1n}), k_2, (P_{21}, P_{22}, \cdots, P_{2m}), k_1$) SELECT-TK($(tk_1, tk_2, \cdots, tk_m), k_3$) where P_{ij}: place number tk_j: the j th token name k_1: part selection rule {LWT,MROQ} k_2: process selection rule {LIT,WIQ,MRIQ} k_3: token selection rule {LIV, NV}
Conditional statement	IF (predicate 1) OP_1 (predicate 2) $OP_2 \cdots$ THEN function 1 ELSE function 2 where OP_i: logical operator {AND, OR}

Token player and animation

A token player executes a Petri net model and interfaces with a high-level control system. During the simulation, the token player executes the movement of tokens in a model, and calls the control system in case control of token movement is needed. Simulation execution can be viewed by an interactive animation system. The interactive animation system interfaces with the token player to display the animated Petri net model.

Model validation can be performed by two procedures: a syntax-oriented Petri net graphics editor and an interactive animation of the Petri net. The Petri net graphics editor interactively checks the failure of token flows, conflict by inhibited arcs, and dead-locks in a circuit during the Petri net model creation. As another way of model validation, the user can execute a Petri net model and can see the flow of tokens from an animated Petri net. During the simulation, users can select a transition to fire among the several alternative transitions. It helps to detect wrong models easily.

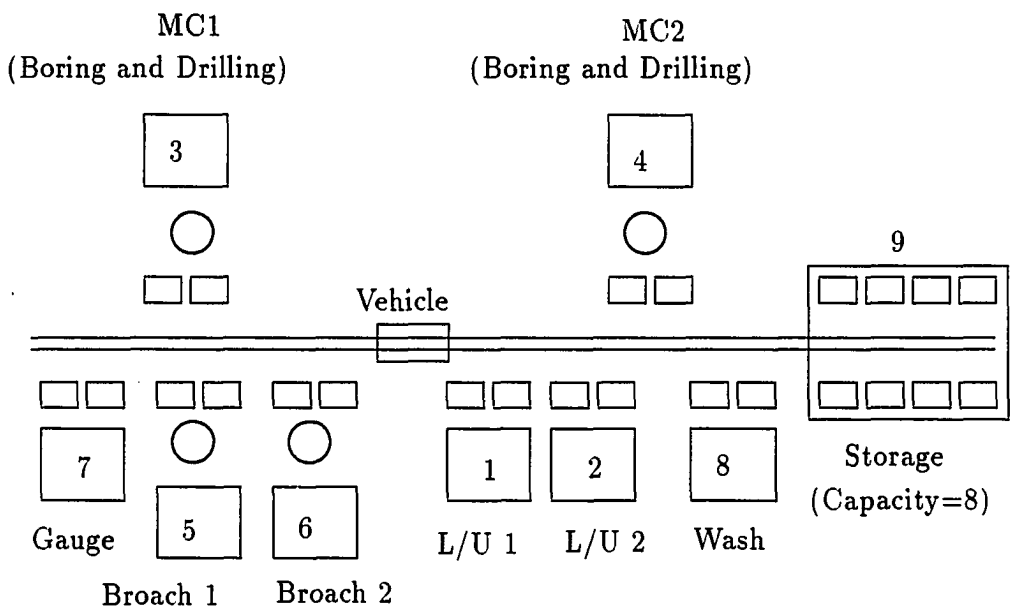
Output analysis and results presentation

From the output of a simulation execution, the output analyzer estimates the performance measures according to the characteristic of simulation–finite horizon or steady state. Several forms of simulation results are presented. The user can select the desired forms (e.g., table or graphic form) and performance measures to be displayed. With gathered place and token statistics, useful measures such as throughput rate, flow time, work in process, and make span can be obtained.

An Example

In this section, a simulation of an FMS using the Petri net-based simulation tool will be demonstrated. An example FMS used for the demonstration was adopted from the literature [17]. In this FMS, two types of jobs are produced, and the quantities of each job type are similar. As shown in Figure 2.7, there is a machine cell and a broaching machine for each job type. Gauge and wash stations are shared by both types of jobs. Raw material is manually loaded into trays at load/unload stations, each for one type of jobs. The tray is moved between work stations by a vehicle moving along a linear track. The sequence of process for each job type is included in Figure 2.7. There are buffer positions at each machine, from which the material is lifted out of the tray and placed into the machine's fixture by a robot. This system is run for three shifts each day to meet demand. There are two men in the load/unload area during the first shift, one man during the second shift, and no men during the third shift. To run the system during the third shift, the raw material should be stored in the storage, and at the same time, the finished part should be fetched from the storage to unload during the first and second shift.

The Petri net model was developed under the Petri net graphics editor (Figure 2.8, 9, and 10). In Figure 2.8, transportation including movement of a vehicle is the focus while work stations are abstracted for further modeling. Figures 2.9 and 2.10 show Petri net models of a machine, and load/unload stations. The load/unload station contains JC place with a job creation rule, SAME, to create jobs of the same type. Note that a Petri net model for load/unload stations includes the timed places, SH1, SH2, and SH3, representing each shift. They were modeled so that the personnel resource is changed according to the shift. In addition, a job release rule, FIFO



Job type	Routing	Processing time (min)
1	1,3,7,5,8,1	20,100,20,60,20,20
2	2,4,7,6,8,2	20,100,20,60,20,20

Figure 2.7: An example FMS

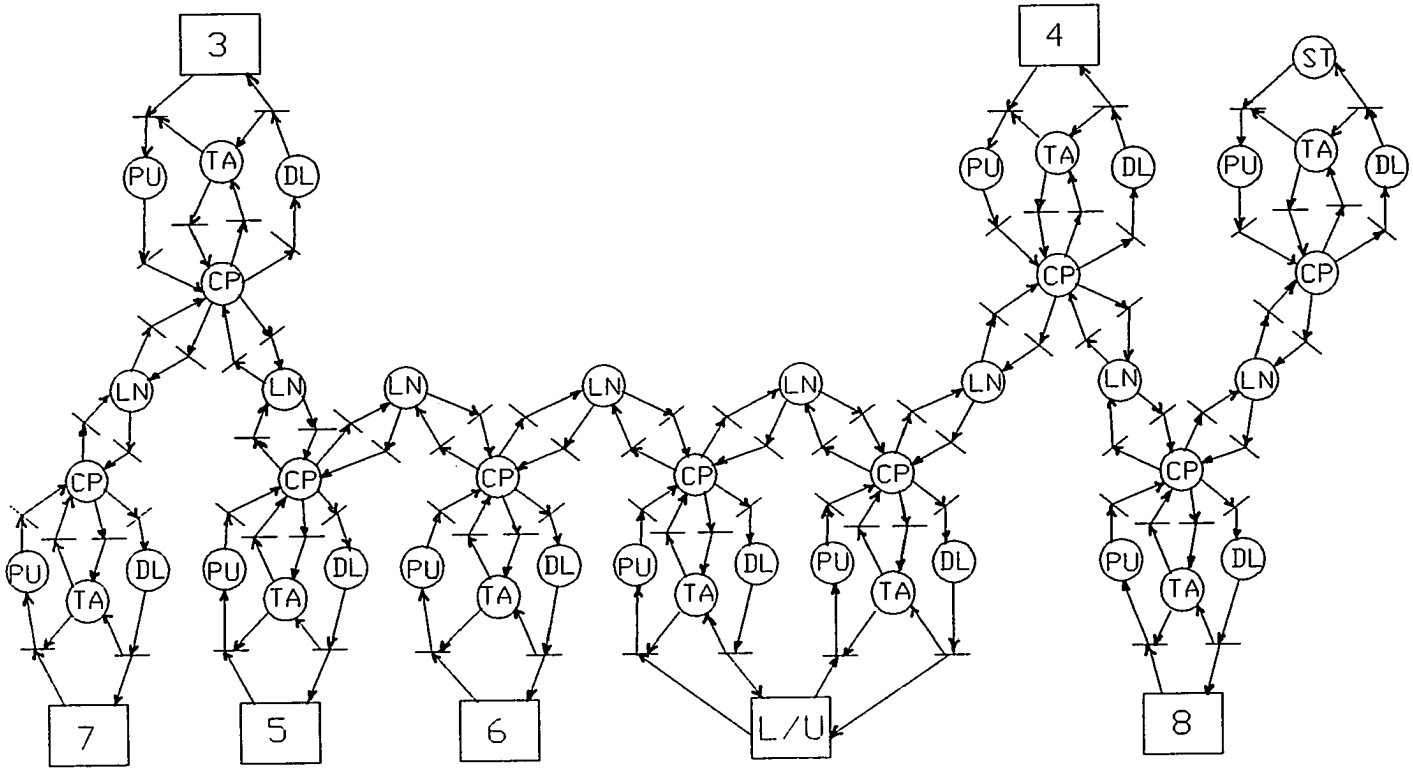


Figure 2.8: A Petri net model of an FMS at transportation level

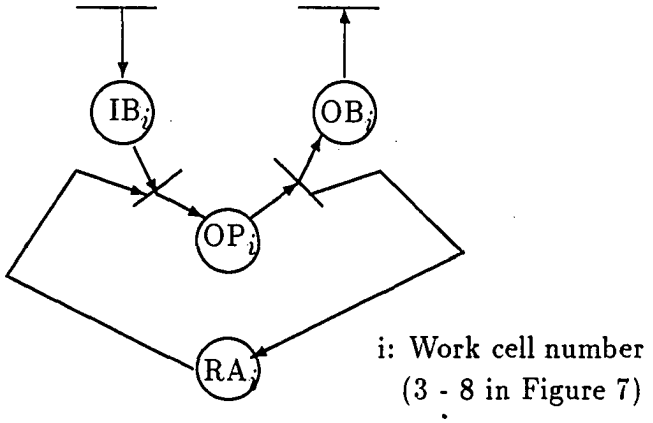


Figure 2.9: Petri net model of a machine center

(as the part selection rule) and LWP (as the pallet selection rule), is attached to the output transition of JC and PS. In addition, the initial tokens in the Petri net model were described in Table 2.3.

The vehicle dispatching rule is modeled using a control rule specification language. Two different rules are modeled to be applied at each shift. During shift 3, the finished part should not be moved to the load/unload area because there are no men in the area. Instead, the finished parts are moved to storage. During shift 1 and 2, the finished parts in storage are moved to the load/unload area, and raw material is stored in storage to be processed in shift 3. The following control rule was modeled to handle these considerations.

1. IF (FULL(SH1) OR FULL(SH2)) THEN
   (a) PUSH((OB1,OB2,OB3,OB4,OB5,OB6,OB7,OB8),LWT,(IB1,IB2, IB3,IB4,IB5,IB6,IB7,IB8),LIT)
   (b) PUSH((OB1,OB2), LWT, (ST))
   (c) PUSH((ST), LWT, (IB1,IB2), LIT)

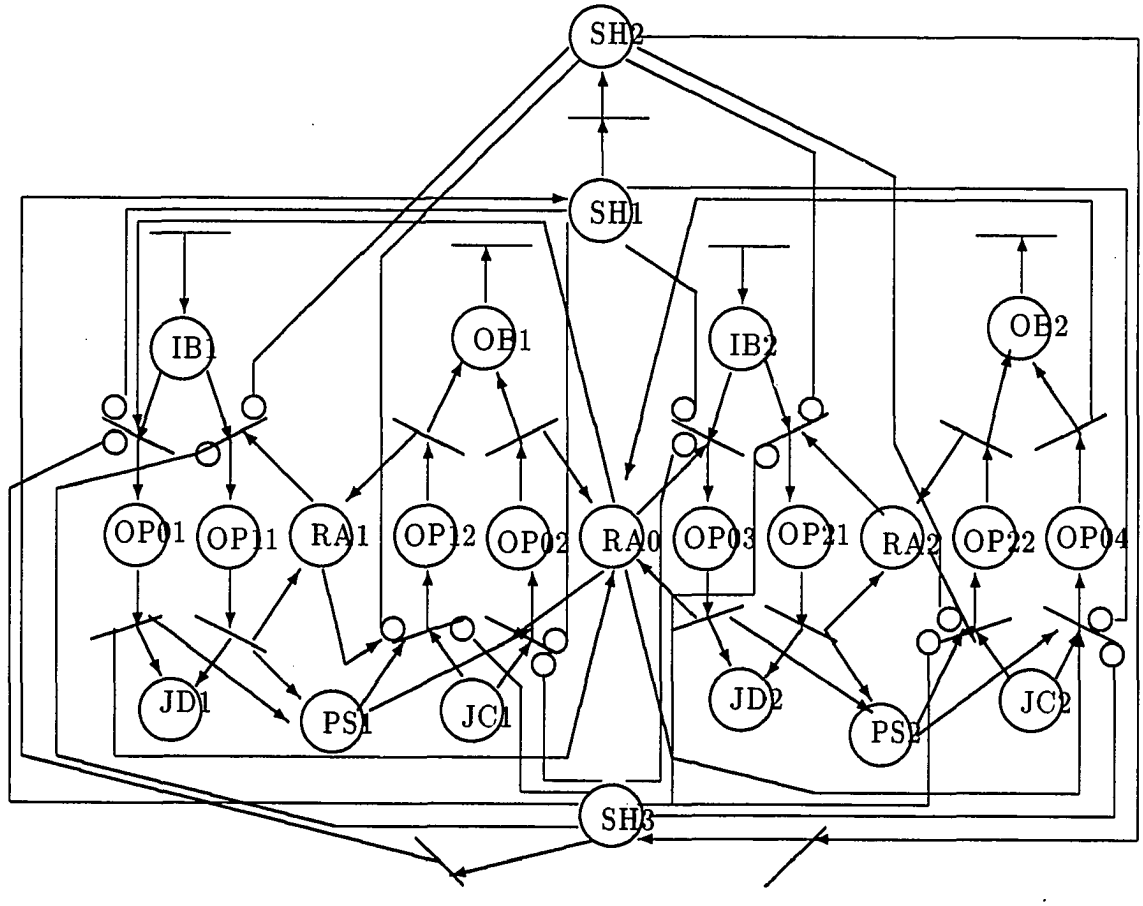


Figure 2.10: A Petri net model of load/unload stations

Table 2.3: Initial tokens

Type	Name	Places	Remark
Controllable	AGV	TA at WASH	
Autonomous	MC1	RA3	
	MC2	RA4	
	BR1	RA5	
	BR2	RA6	
	GAUGE	RA7	
	WASH	RA8	
	MAN1	RA1	
	MAN2	RA2	
	MAN3	RA0	
	SHIFT	SH1	Shift token
Passive	JOB	JC1	Type 1
		JC2	Type 2
	(JOB,PALLET)[a]	IB1,OB1,IB3,	Job
		OB3,OP3,IB5,	type 1
		OP5,OB5,IB7,	
		OB7,OB8,ST(4)[b]	4 tokens in ST
		IB2,OB2,IB4,	Job
		OP4,OB4,IB6,	type 2
		OP6,OB6,OP7,	
		IB8,OP8,ST(4)[b]	4 tokens in ST

a: Combined token of JOB and PALLET.

b: ST place contains 8 tokens.

2. IF FULL(SH3) THEN

   (a) PUSH((ST),LWT,(IB3,IB4),LIT)

   (b) PUSH((OB3,OB4,OB5,OB6,OB7,OB8),LWT,(IB3,IB4,IB5, IB6,IB7,IB8,ST),LIT)

The control rule model was translated to a C program to be compiled and linked with the token player and animation system.

Figure 2.11 shows the simulation output regarding place and token statistics.

Discussion and Future Study

In this paper, the Petri net-based simulation tool was presented. It was programmed with the C language under MS-DOS with a micro computer and EGA graphic facility.

The developed simulation software has some weaknesses which need further study. Especially, in modeling FMSs, the following aspects are required to be extended.

- Extended place objects.
- Models of high-level, real-time control systems

To give extensive modeling power to the tool, diverse place objects are required. We are considering a tool to make new place objects. According to the application, a modeler may define the required place objects under the place object definition system. Then, users could make a model using the predefined place objects.

The proposed token control language has limitations to represent more complex rules. By allowing the users to make a program for control rules with the C language,

1. Controlled Token Statistics

Token type	Token Name	Moving with part	Waiting with part	Moving empty	Waiting empty
AGV	AGV1	335.0 (28%)	0.0 (0%)	420.0 (35%)	445.0 (37%)

2. Job Token Statistics

TOTAL TIME = 1200.0
NO OF JOBS COMPLETED - TYPE 1 = 11
NO OF JOBS COMPLETED - TYPE 2 = 10

3. Place Statistics

Place No	Place Type	Cell No.	No. of Pass	Process Time	Wait Time	Queue Size	Capacity
1	OP	7	21	315.0	41.4	0.7	1
2	RA	7	21	0.0	0.0	0.0	1
3	IB	7	22	0.0	60.5	1.1	2
4	OB	7	21	0.0	51.6	0.9	1
5	OP	5	11	660.0	43.6	0.4	1
6	RA	5	11	0.0	0.0	0.0	1
7	IB	5	11	0.0	97.3	0.9	1
8	OB	5	11	0.0	98.5	0.9	1
9	OP	3	11	1100.0	0.0	0.0	1
10	RA	3	11	0.0	0.0	0.0	1
11	IB	3	10	0.0	101.0	0.8	1
12	OB	3	11	0.0	30.5	0.3	1
13	OP	6	11	660.0	43.6	0.4	1
14	RA	6	11	0.0	0.0	0.0	1
15	IB	6	10	0.0	50.6	0.4	1
16	OB	6	11	0.0	105.4	0.9	1
:	:	:	:	:	:	:	:

Figure 2.11: Simulation output for place and token statistics

or by providing interfaces with other systems (e.g., expert systems), the complex control rules may be modeled. In this case, there is a difficulty that users must know in detail about the mechanism of the token player.

References

[1] Aimo A. Törn, "Simulation Nets, A Simulation Modeling and Validation Tool," *Simulation*, Vol. 45, No. 2, 1985.

[2] Giorgio Bruno and Maurizio Morisio, "Petri Net based Simulation of Manufacturing Cells," *Proc. IEEE Int. Conf. on Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1987.

[3] P. Alanche, K. Benzakour, F. Dolle, P. Gillet, P. Rodrigues, and R. Valette, "PSI: A Petri Net Based Simulator for Flexible Manufacturing Systems," *Lecture Notes on Computer Science*, 188, Springer-Verlag, Berlin; New York, 1984.

[4] Antonio Camurri and Macercello Frixione, "Structured Representation of FMS Integrating SI-NETS and High Level Petri Nets," *Applied Artificial Intelligence*, Vol. 4, 1990.

[5] Javier Martinez, Pedro Muro, and Manuel Silva, "Modeling, Validation and Software Implementations of Production Systems Using High level Petri nets," *Proc. IEEE Int. Conf. on Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1987.

[6] D. S. Yim and T. A. Barta, "Conserved Nets for modeling and simulation of FMSs," submitted to *J. of Manufacturing Systems.*

[7] Y. Narahari and N. Vishwanahham, "A Petri net Approach to the Modeling and Analysis of Flexible Manufacturing Systems," *Annals of Operations Research*, Vol. 3, 1985.

[8] E. Best and P. S. Thiagarajan, "Some Classes of Live and Safe Petri Nets," in *Concurrency and Nets: Advances in Petri Nets* (Edited by J. Hartmann, K.V. Genrich, and G. Rozenberg), Springer-Verlag, Berlin; New York, 1987.

[9] F. Commoner and A. W. Holt, "Marked Directed Graphs," *J. of Computer and System Sciences*, No. 5, 1971.

[10] M. Hack, "Analysis of Production Schemata by Petri Nets," Technical Report 94, Project MAC, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1972.

[11] G. K. Hutchinson and A. T. Clementson, "Manufacturing Control Systems: An Approach to Reducing Software Costs," *Robotics and Computer integrated Manufacturing*, Vol. 1, No. 3/4, 1984.

[12] Meng Chu Zhou and Frank Dicesare, "A Petri Net Design Method for Automated Manufacturing Systems with Shared Resources," *Proc. IEEE Int. Conf. on Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1990.

[13] R. Suri and C. K. Whitney, "Decision Support Requirements in Flexible Manufacturing," *J. of Manufacturing Systems*, Vol. 3, No. 1, 1984.

[14] P. J. Egbelu and J. M. A. Tanchoco, "Characterization of AGV Dispatching Rules," *Int. J. of Production Research*, Vol. 22, No. 3, 1984.

[15] P. J. Egbelu, "Pull Versus Push Strategy for Automated Guided Vehicle Load Movement in a Batch Manufacturing System," *J. of Manufacturing Systems*, Vol. 6, No. 3, 1987.

[16] Roberta S. Russel and J. M. A. Tanchoco, "An Evaluation of Vehicle Dispatching Rules and Their Effect on Shop Performance," *Material Flow*, 1984.

[17] Allen Carrie, *Simulation of Manufacturing Systems*, John Wiley & Sons, New York, 1988.

Appendix: Conserved Nets

A Petri net is defined formally as the tuple $W = (P, T, A, M)$, where P is the set of places $(p_1, p_2, \cdots, p_n)$, T is the set of transitions $(t_1, t_2, \cdots, t_m)$, and A, M are functions. M is marking of P and the number of tokens in p_i is represented as $M(p_i)$. The set of $N = P \cup T$ is called a node set and an element of $n_i \in N$ is called a node. The connection relationship between node n_i and node n_j is represented by $A(n_i, n_j)$. If a directed arc connects from n_i to n_j the value of $A(n_i, n_j)$ is 1. Otherwise the value of $A(n_i, n_j)$ is zero.

In addition, the following attributes are attached to Petri net elements to increase the modeling power of a Petri net, and they can be exploited for the simulation of FMSs.

1. Each place has a capacity, $C(p_i)$, and a processing time $PT(p_i)$.
2. Each token is identified as an individual object, and belongs to a certain type.
3. Each output arc of a transition has attached to it a set of token types to flow.
4. Each output arc of a place can have attached to it a predicate for a decision on token movement.

Originally, the marking of tokens under transition firing rule is based on the deletion and creation of tokens. When a transition fires, tokens in the input places are deleted and new tokens are created in the output places of the transition. In modeling an FMS, tokens represent resources or jobs in the system. These tokens are flow objects in the system, and must be conserved in a net. Rather than being based on the creation and deletion of tokens, the transition firing rule needs to consider

the token movement such that tokens flow in a net without any transformation. Four kinds of token flows occur in a net. The possible token flows at each node are determined by examining the specification of token types attached to the output arcs of transitions. Let ${}^{\bullet}F_{n_i}$ and $F^{\bullet}_{n_i}$ be possible input token flow and output token flow at a node n_i in a net $G = (P, T, A)$.

1. Input token flows at a place.

   The possible input tokens at a place are determined by the union of token sets specified at the input arcs of the place. If a place p has n input arcs and the set of allowable token flows, a_i, is specified at the i th input arc, the possible input tokens at p is determined as

   $$ {}^{\bullet}F_p = \cup_{i=1}^{n} a_i $$

2. Output token flows at a place.

   When a token (combined or original) resides in a place p, it moves along the output arcs of the place without any transformation (note that there is at most one arc between any two nodes), i.e., $F^{\bullet}_p = {}^{\bullet}F_p$

3. Input token flows at a transition.

   The possible input tokens at a transition are determined by the product of token sets from the output token flows of input places. If a transition t has n input arcs (i.e., n input places), and possible output tokens at the i th input places is $F^{\bullet}_{p_i}$, then the possible input tokens of t is:

   $$ {}^{\bullet}F_t = F^{\bullet}_{p_1} \times F^{\bullet}_{p_2} \times \cdots \times F^{\bullet}_{p_n} $$

4. Output token flows at a transition.

   The possible output tokens at a transition are determined by the product of token sets specified at output arcs of that transition. When a transition t has n output arcs, and each arc has attached to it a set of token types a_i, then the possible output tokens at t is:

$$F_t^{\bullet} = a_1 \times a_2 \times \cdots \times a_n$$

From the above results, the possible token movements at each node can be determined. To guarantee conservativeness in a net, the input and output token flows at each node should be same. We develop the following definition of Conserved nets.

Definition: $G = (P, T, A)$ in which the specification of token flows is attached to the output arcs of transitions is called Conserved net if the following conditions hold in the net:

1. $A(n_i, n_j) = 1$ or 0, for any pair of nodes n_i and n_j.

2. When a transition has more than one input place,

$$\text{any element of } {}^{\bullet}F_{p_i} \neq \text{any element of } {}^{\bullet}F_{p_j}$$

   where p_i and p_j are any pair of input places.

3. ${}^{\bullet}F_t = F_t^{\bullet}$, for all t.

From the definition of a conserved net, the following properties are obtained.

Property 1: A conserved net can be decomposed into subnets for the flow of each

token type.

Property 2: A decomposed subnet of a token type flow is a strongly connected, closed subnet, and consists of several directed circuits.

Property 3: When two decomposed subnets of different token flows share common paths, the paths start and end with transitions.

Property 4: When two directed circuits in a subnet of a token type flow share common paths, the paths start and end with places.

At below, the liveness of Conserved Petri net system are briefly described without proof. Before we present liveness conditions, two Petri net systems will be considered.

When a place in a *r-system* has more than one output arc, the marked token in the place will move to any one of the arcs randomly whenever the connected transition meets enabling conditions. But, in a *d-system*, a marked token in a place which has more than one output arc must move along one of the output arcs according to the decision specifications attached to the arcs.

Proposition 1: An *r-system*, $W = (P, T, A, M)$, is live if and only if the number of tokens in the system, $\sum_{p_i \in G} M(p_i)$, is greater than zero and less than $\sum_{p_i \in G} C(p_i)$.

Proposition 2: A *d-system*, $W = (P, T, A, M)$, is live if the number of tokens in the system is greater than zero and less than $min\{\sum_{p_i \in G_k} C(p_i), i = 1, 2, ..., m\}$, where G_k is the k th directed circuit in $G = (P, T, A)$, and m is the number of directed

circuits in G.

Proposition 3: When two subsystems W_1 and W_2 which are live are combined sharing a common path which starts and ends with transitions, the combined system is live if and only if the following conditions are avoided:

(i) non-sharing places of a subsystem are not marked with tokens, and

(ii) all non-sharing places in the other subsystem are marked with tokens of the same number as the capacity.

PART III.

PUSH AND PULL RULES FOR DISPATCHING AUTOMATED GUIDED VEHICLES IN A FLEXIBLE MANUFACTURING SYSTEM

Push and Pull Rules for Dispatching Automated Guided Vehicles in a Flexible Manufacturing System

D. S. Yim and Richard J. Linn

Department of Industrial and Manufacturing Systems Engineering

Iowa State University, Ames, Iowa 50010, USA

Abstract

Automated Guided Vehicle (AGV) systems are widely used in flexible manufacturing systems (FMSs) for material handling purposes. Although the AGV systems have provided high flexibility, the design issue on AGV dispatching rules is still to be resolved. The AGV dispatching rules in an FMS are generally based on a push or a pull concept. A simulation study is accomplished to investigate the effect of these dispatching rules on the FMS performance. The developed simulation model consists of two modules: a Petri net model and an AGV dispatcher. Two modules are integrated so that the AGV dispatcher controls AGV tokens in the Petri net model. It was shows that there is no significant difference in output rate between push- and pull-based AGV dispatching rules in a busy FMS.

Keywords: Push and pull rules, AGVs, Petri nets, Simulation

Introduction

During the past several years, Automated Guided Vehicle (AGV) systems have received much attention by designers and engineers of automatic manufacturing systems. The AGV is a battery-powered, wire-guided vehicle, and is controlled by an on-board or a network control computer. The AGV systems have been widely used in Flexible Manufacturing Systems (FMS). Although they provide higher flexibility than conventional systems, the design issues of AGV control systems in FMSs are still to be resolved. The AGV control system dispatches idle vehicles to move pallets, parts, and tools between work centers in an FMS. The complex interaction between material flows and processes requires an efficient vehicle dispatching procedure to maximize the FMS performance.

Because of its ability to graphically and hierarchically represent systems with both asynchronous and concurrent properties, Petri nets have proved to be an efficient tool to model the complex interactions among different processes in an FMS. In this study, a Petri net-based simulation model was developed for an AGV operating in an FMS. The model was used to analyze the effect of different AGV dispatching rules on the FMS system performance.

AGV Dispatching Rules

Vehicle dispatching rules involve assigning vehicles to move loads, and concerns the relationship between the vehicle resource and the set of parts to be moved. Generally, AGV dispatching rules are classified into work-center-initiated rules and vehicle-initiated rules. When a work center has a part to be routed for the next operation,

it selects a vehicle among a set of idle vehicles according to the work-center-initiated rule. When a vehicle becomes idle, it selects a task (i.e., a part to move) to serve under the vehicle-initiated rule. The vehicle initiated rule can be further classified into a source-driven rule and a demand-driven rule. The source-driven rule operates on a push concept: an idle vehicle selects a part to move from an output queue that has the highest priority. The demand-driven rule operates on a pull concept: an idle vehicle selects the part that has the highest demand from its succeeding work stations.

The simulation has been recognized as an invaluable tool in evaluating the performance of AGV systems. A number of studies on the AGV dispatching rules were based on the simulation technique (Egbelu and Tanchoco, 1984; Egbelu, 1987; Russel and Tanchoco, 1984; Sabuncouglu and Hommertzheim, 1989; Ülgen and Kedia, 1990). Egbelu and Tanchoco (1984) compared the performance of several work-center-initiated rules and vehicle-initiated rules in a batch manufacturing system. They showed that in a busy shop the vehicle-initiated rule has a more significant effect on system performance than the work-center-initiated rule. Egbelu (1987) further compared the performance of a demand-driven rule and several source-driven rules in a batch manufacturing system. He concluded that the demand-driven rule is competitive to source-driven rules.

In FMS environment, a vehicle-initiated rule consists of a part selection function and process selection function. Very often parts have alternative routings where they may be sent to different work centers. An idle vehicle needs to select not only the part to move but also its destination (next process). The push and pull concepts of vehicle-initiated rules can be implemented on the basis of the execution order of

part selection and process selection functions. In the push rule, an idle vehicle first selects a part to move and then determines the destination of the selected part. In a pull-based rule, on the other hand, a work center with the highest need for part replenishment is selected first. Then, a part is selected among a set of parts which can move to the selected work center. Thus, the two AGV dispatching rules—push and pull—have their own characteristics. In the push-dispatching rule, the parts in the outgoing buffers of work centers are a major concern, while the incoming buffer status of each work center is a major decision factor in the pull-dispatching rule.

By pairing the part and process selection functions, numerous different push and pull rules can be generated. The following part and destination selection rules were included in the investigation. Some of these rules were adopted from the literature (Egbelu and Tanchoco, 1984; Russel and Tanchoco,1984).

1. Part selection rule

    - Longest waiting time rule (LWT): select a part with the longest waiting time.
    - Minimum remaining outgoing queue space rule with longest waiting part (MROQ): select a longest waiting part which is in the output buffer with minimum remaining queue space.

2. Process selection rule

    - Longest inter-arrival time rule (LIT): select a work center which has experienced the longest inter-arrival time of parts since the last job arrival.
    - Maximum remaining incoming queue space rule (MRIQ): select a work center with maximum remaining queue space at input buffer.

Table 3.1: Push and pull AGV dispatching rules

	Rule number	Part selection	Process selection
	1	LWT	MWQ
	2	LWT	LIT
Push	3	LWT	MRIQ
	4	MROQ	MWQ
	5	MROQ	LIT
	6	MROQ	MRIQ
	7	LWT	MWQ
	8	LWT	LIT
Pull	9	LWT	MRIQ
	10	MROQ	MWQ
	11	MROQ	LIT
	12	MROQ	MRIQ

- Minimum work-in-queue rule (MWQ): select a work center with minimum incoming queue size in terms of processing time.

Twelve different push and pull rules, as shown in Table 3.1, were included to investigate the effect of vehicle-initiated rules.

Push-based AGV dispatching procedure

Push-based procedure selects a part (source) first, then determines to where (destination) it should be moved. Once the source and destination are determined, an AGV is selected to perform the selected load movement. When selecting a part, a set of workstations (source) that are not assigned any AGV to pick up their loads is first determined. Then, a part is selected from the output buffers of this set of workstations according to the part selection rule specified. If no such a station is found, or no part is selected, the procedure is aborted.

Once a part is selected from the set of workstations, a destination for the part

will be determined according to the process selection rule specified. The input buffer of the destination must not be full. If no destination is possible for the part, another part from the source set will be picked.

When a source and a destination are determined, an idle AGV will be selected to perform the part movement. If no AGV is idle, the procedure is aborted.

Pull-based AGV dispatching procedure

Pull-based AGV dispatching procedure first selects a workstation (destination) which can receive parts according to the process selection rule specified. Then, a list of parts which can be moved to this selected workstation is identified from the output buffers of other workstations. Finally, the part selection rule is applied on this list to select a part (source).

AGV System Description

Generally, an AGV system contains four major components: (1) the transport network, (2) the vehicles, (3) the interface between the production system and AGVs, and (4) control system. There are basically three types of transport network: single line, simple loop, and network type. The network type system requires more complex control logic, especially when AGVs move along the line bidirectionally. To simplify the control over collision and blocking problems, unidirectional path is commonly used in the network type configuration. Six types of automated guided vehicles are available: unit load, towing, pallet truck, fork truck, light load, and assembly line vehicles (Miller and Walker, 1990). Among them, the unit load vehicles become more popular recently.

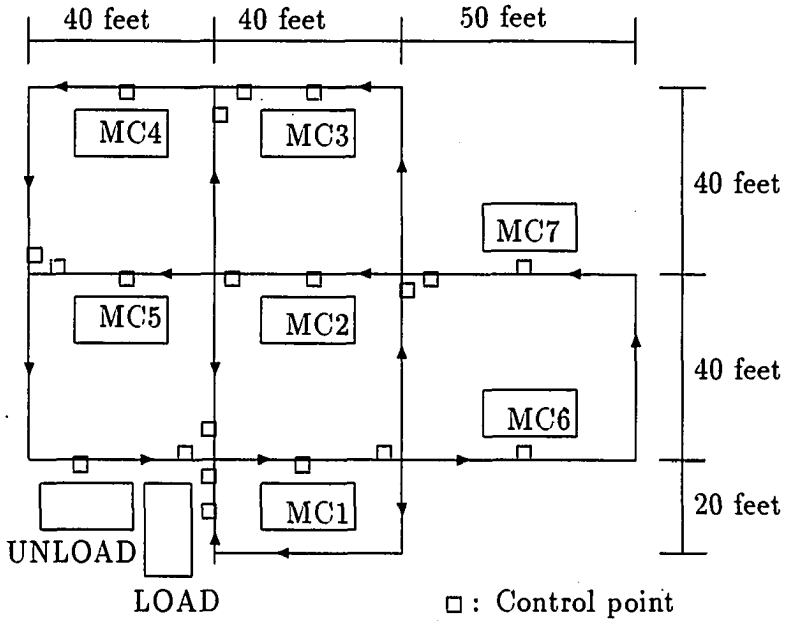


Figure 3.1: A hypothetical FMS

The FMS system considered in this simulation study is shown in Figure 3.1. The track layout is a network type, and AGVs move unidirectionally through the network. The network is subdivided into nonoverlapping zones so that no more than one vehicle is permitted within a zone at any time, prohibiting the collisions between AGVs. The zones are identified by a set of control points, at which the AGV receives the command from a control computer such as wait, move, change velocity. Twenty control points are identified in the FMS, containing pick-up, drop-off, diverging, and converging points at intersections.

There are nine workstations including seven machine centers, one load station, and one unload station. Each machine center consists of one machine, one input buffer, and one output buffer, except the machine centers MC 6 and MC 7. Each of them contains two identical machines, one input buffer, and one output buffer. The

interface between workstations and AGVs occurs at the input and output buffers of workstations. At every workstation, AGV picks up a part from the output buffer and deliver a part to the input buffer.

Petri Net Modeling of the AGV and FMS

The FMS simulator developed in this study comprises two subsystems: an AGV dispatcher and a Petri net-based FMS. The AGV dispatcher is responsible for dispatching AGVs in a Petri net model. The hardware components of FMS were modeled with Petri nets.

Petri net modeling is becoming attractive for analyzing and simulating manufacturing systems (Peterson, 1984; Kodate, et al., 1987; Cheng, 1987; Beck and Krogh, 1986; Bruno and Morisio, 1987; Vallete, 1984; Martinez, et al., 1987). For more details about Petri nets, readers are referred to Peterson (1984). Basically, Petri net is capable of modeling the multi-condition processes that has concurrency and cooperation. A Petri net consists of four parts; namely, a set of places P, a set of transition T, flow relations F, and initial marking of tokens M_0. Pictorially, places are represented by circles, transitions by bars, flows by arcs, and tokens by dots. Places in a Petri net could represent waiting conditions for program execution; transitions could represent occurrence of events in a real system; and the token would then correspond to the number of occurrence of the events. The resulting interpreted net with its marking simulates the synchronization of the events. The evolution of tokens indicates which conditions cause a transition to fire (to be enabled). The most important modeling property of Petri net is the ability to represent concurrency and conflict.

The Petri net exploited in this study allows colored tokens, inhibited arcs, and capacitated, timed places to model and simulate the AGV system. Place and token objects were classified to facilitate the modeling of FMSs. The classified places correspond well with physical elements of the FMSs (See Table 3.2). Each place object has its own attributes and allows only the specific token types. In the current Petri net modeling, token objects are classified as active tokens or passive tokens. The active tokens such as vehicles, machine resources, personnel resources, robots can move to next places voluntarily. The AGV tokens are controlled by the AGV dispatcher during the simulation. Passive tokens such as jobs, pallets, and fixtures cannot move without being combined with active tokens.

The Petri net model for the FMS is shown in Figures 3.2, 3, and 4. Figure 3.2 shows a Petri net model at a transportation level where AGV movement is the major focus, and work stations are abstracted for the detail modeling. The zone control logic is represented by using inhibited arcs which have circular head instead of arrow. Pick up and delivery processes are associated with PU and DL places. AT places represent the places where idle AGVs are waiting.

Figures 3.3 shows the detailed models for the machine center with two machines, and Figure 3.4 shows the load station and the unload station. Each machine center consists of one input and one output buffer (IB, OB), operations (OP) and machine resource available (RA) places. The job token in IB place moves to OP place being combined with a machine token whenever one of two machines is available. After processing in the OP place, the job token moves to the OB place, and machine token moves to the RA place. At load and unload stations, job tokens are created and deleted. While the load station contains a job creation (JC) place in which job

Table 3.2: Place objects for Petri net modeling of FMSs

Objects	Mnemonic name	Token types[a] through input arcs	Attributes[b]
Operation	OP	(A,J,P)	MTBF, Process time Decision arc
Input buffer	IB	J or (J,P)	Capacity Token link method
Output buffer	OB	J or (J,P)	Capacity Token link method
Resource available	RA	A	
Job creation	JC	No token	Token creation rule
Job deletion	JD	J	
Control point	CP	C or (C,J,P)	Decision arc
Line	LN	C or (C,J,P)	Process time Decision arc
Transporter available	TA	C	Decision arc
Pick-up	PU	(C,J,P)	Pick-up time
Delivery	DL	(C,J,P)	Delivery time
Job storage	JS	J	Capacity Token link method
Pallet storage	PS	P	Capacity

a. Token types.
A: Autonomous tokens (machines).
C: Controlled tokens (vehicles).
J: Job tokens.
P: Passive tokens except job tokens (pallets).
* A combined token is represented as a turple of each token type.
b. The capacity of every place is one except IB, OB, JS and PS.

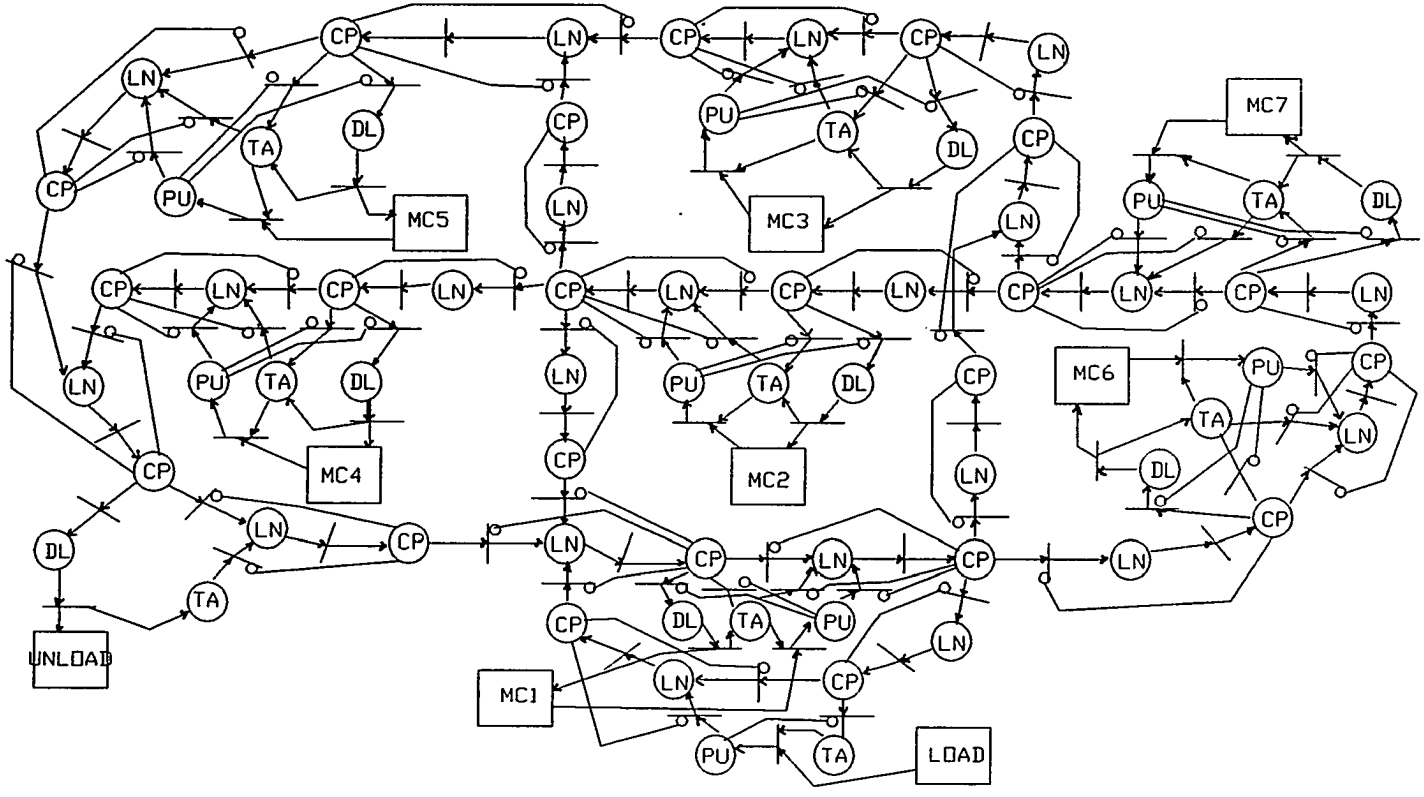


Figure 3.2: A Petri net model of an FMS at transportation level

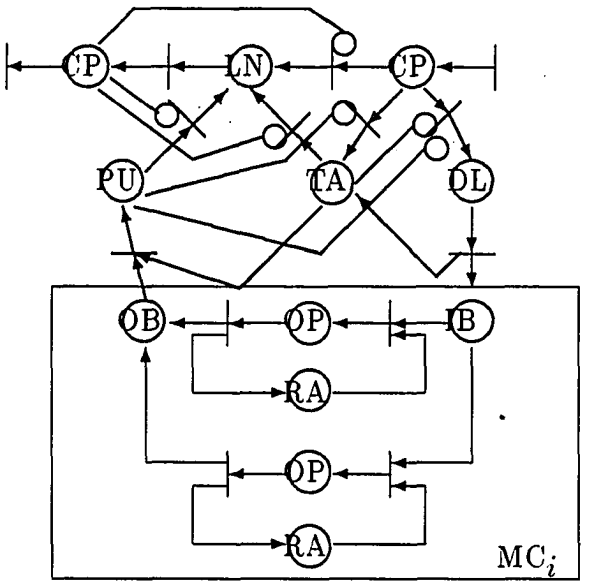


Figure 3.3: A Petri net model of a machine center with two machines

tokens are created, the unload station contains a job deletion (JD) place to delete the job tokens. When a part is removed in the unload station, the separated pallet is stored in pallet storage (PS) place. In the load station, a job in the job storage (JB) and a pallet in PS are combined according to a job release rule. The job release rule is invoked when the output transition of these places are enabled in order to select appropriate tokens.

Besides the inherent advantages of Petri net modeling, the simulation of an AGV system in FMSs using Petri net presents several advantages.

1. The synchronous and asynchronous mechanism of FMSs can be easily modeled. Also, bidirectional as well as unidirectional AGV systems are easily modeled.

2. A simulation program is easily developed by using the Petri net mechanism.

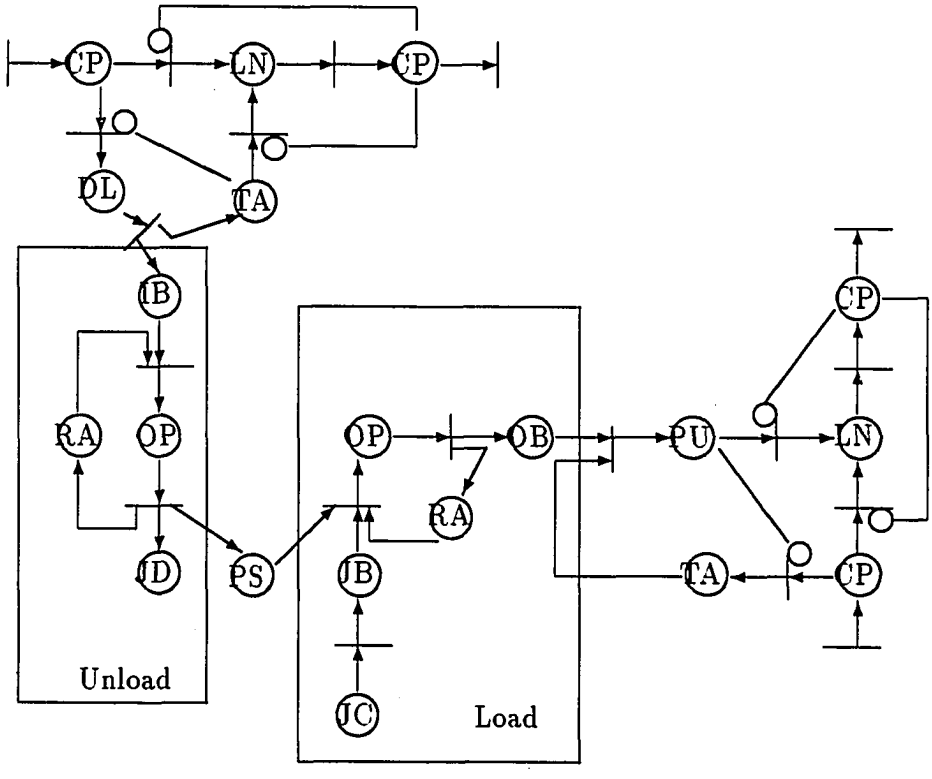


Figure 3.4: A Petri net model of load/unload stations

3. Animation of a simulated system can be accomplished by using a Petri net graph and objects that correspond with the physical elements of FMSs. The validation of simulation model is possible by an interactive animation of the Petri net graph.

The Petri net modeling has limitations in representing high-level control systems of FMSs. While low-level control systems (i.e., machine level control systems) can be well represented in Petri nets, the high-level control systems which require complex decision making process including the analysis of system status and historical data are difficult to model in Petri nets (Valette, 1984; Camurri and Frixione, 1990). That is the reason why AGV dispatcher was not implemented in Petri nets. When additional high-level control rules such as scheduling rules are needed to resolve conflicts in the Petri net model, external modules must be added.

Experimental Design and Assumptions

Assumptions

Several assumptions were made to keep the investigation in a manageable scope. The following assumptions were held throughout the investigation.

1. Eight job types are to be produced. Their processing times and routings are shown in Table 3.3.

2. A job is loaded into the system at load station when the corresponding pallet is available. When multiple jobs exist, the job type with small number of jobs launched is released whenever it is possible.

Table 3.3: Job routings for a hypothetical FMS

Job type	Routing	Process time (min)	Part mix
1	1,2,(7,8),6,9	5,16,(30,30),20,5	1/8
2	1,(3,4),(7,8),5,9	5,(16,16),(30,30),10,5	1/8
3	1,(7,8),2,(3,4),9	5,(30,30),15,(15,15),5	1/8
4	1,2,(7,8),6,(3,4),9	5,10,(20,20),10,(10,10),5	1/8
5	1,4,3,5,(7,8),9	5,16,20,10,(10,10),5	1/8
6	1,2,6,(7,8),(3,4),9	5,10,10,(30,30),10,5	1/8
7	1,(3,4),5,6,(7,8),9	5,(10,10),10,10,20,5	1/8
8	1,(7,8),(3,4),5,6,9	5,(30,30),(12,12),10,10,5	1/8

* Routing number.
1:LOAD 2:MC1 3:MC2 4:MC3 5:MC4
6:MC5 7:MC6 8:MC7 9:UNLOAD
* The routing numbers in a parenthesis means alternative routings.

3. Parts at each machine centers are processed on a First-come First-served basis.

4. Machine breakdowns are excluded in this study.

5. AGVs are unit load vehicles and they travel at 120 feet/min. Pick-up and delivery times are 0.5 minutes respectively.

6. The travel of AGVs between two locations will follow the shortest path possible.

7. Each AGV will wait in front of the work station (AT places in the Petri net model) after unloading a part on to input buffer.

8. Each pallet type has the same number of pallets.

Minimum number of AGVs and buffer capacity

In order to reduce the effect of the number of AGVs, the minimum number of AGVs which can perform target production must be determined. The detail of the

analytical procedure for obtaining the minimum number of AGVs is described in Appendix. A minimum number of 2 AGVs were determined under the assumption that a target production rate is 60 per 8 hour shift. From simulation experiments, it was found that the output rate of the system with 2 AGVs was lower than the output rate of the system with 3 AGVs. But, increasing the number of AGVs more than three did not improve the performance of the system. For the rest of simulation, therefore, three AGVs were used in the system.

Since the AGV utilization with three AGVs was about 87 %, the shop is considered busy. It was shown in the literature (Egbelu and Tanchoco, 1984) that AGV selection rule has little effects on the system performance than part and process selection rules when AGV resource is rather restricted (i.e., busy shop). When there are several AGVs waiting for mission, the least utilized AGV is selected in this simulation study.

To determine the buffer capacity, a number of initial simulation runs was accomplished. When it was assumed that infinite number of pallets is in the system, buffer capacity of three gives the best result in terms of output rate and work in process. Therefore, all buffers are assumed to have capacity of three.

Simulation Output Analysis

To evaluate the performance of AGV dispatching rules, the analysis of simulation output was based on steady-state statistics of output rate. Initially, there is no part in the system, all machines are idle, all available pallets are at loading area, and AGVs are idle (i.e., each machine token is at the corresponding RA place, pallet tokens are at PS place, and each AGV token is at an AT place in the Petri net

model). After the steady-state point was reached (i.e., average output rate shows to be stable), the simulation was executed for 4320 minutes, and steady-state statistics were estimated. To decrease the bias in the estimation of mean values, the transient state of simulation was excluded from the output collection.

The model validation is accomplished using an interactive animation of a Petri net model. The token movements in the Petri net model are graphically displayed so that false modeling can be detected. When a transition fires or a shop locks, the related information is displayed to users.

Table 3.4 summarizes the simulation results of different dispatching rules on the average output rate with different number of pallets in the system. As the number of pallets increases, the shop locking also increases. The detailed explanation about shop locking phenomenon in AGV systems can be found in the literature (Egbelu and Tanchoco, 1984). While seven rules show shop locking under infinite number of pallets, only one rule (pull rule with MRIQ+MROQ) shows shop locking when two pallets for each job type is available in the system. This confirms the fact pointed out in the literature (Sabuncouglu and Hommertzheim, 1989) that the large number of parts in the system increases the possibility of shop locking due to excessive congestion and traffic on the shop floor. It is also shown that, in general, the average output rate increases as the number of pallets increases from 2 to 3, but drops once the number of pallets goes beyond 4. Hence, the average output rate is not increased by allowing the large number of pallets in the system. Too small number of pallets in the system also decreases average output rate as well.

In order to reduce the effect of shop locking on the dispatching rule performance, further experiment was performed with the following conditions imbedded:

Table 3.4: Result of average output rate/8 hrs

	Rules	No. of pallets for each job type					
		2	3	4	5	6	∞
Push	1 LWT+MWQ	57.5	65.2	68.2	62.6	59.5	59.5
	2 LWT+LIT	57.5	63.4	66.8	64.0	64.0	64.0
	3 LWT+MRIQ	52.5	62.6	63.4	62.6	64.0	64.0
	4 MROQ+MWQ	55.9	65.3	64.4	62.6	N[a]	N
	5 MROQ+LIT	58.2	63.3	63.2	N	N	N
	6 MROQ+MRIQ	50.4	62.2	63.5	63.7	58.1	N
Pull	7 MWQ+LWT	58.6	63.6	66.2	66.2	66.2	62.8
	8 LIT+LWT	56.2	65.6	N	65.4	N	N
	9 MRIQ+LWT	52.0	62.1	65.3	59.4	46.0	50.9
	10 MWQ+MROQ	57.1	65.5	65.2	N	N	N
	11 LIT+MROQ	57.3	64.8	N	N	N	N
	12 MRIQ+MROQ	N	N	63.4	57.5	N	N
	Overall Average[b]	55.7	64.0	65.0	62.7	59.6	60.24

a: Shop locking during simulation.
b: Excluding shop locking occurrences.

(1) four pallets for each job type is available in the system,

(2) an AGV will not move the selected parts if

A. the input buffer of the first machine for the part in load station has only one remaining space, and

B. the output buffer of the part to be moved has more than one remaining space, and the input buffer of the next work station has only one remaining space.

These two conditions are included in the experiment in order to provide the maximum output rate, and to reduce the shop locking at the same time. Condition 2 aims at reducing the shop locking by not performing the load movement that has high possibility of causing shop locking.

Table 3.5: Result of average output rate under two conditions

	Rules	Output rate Mean/ 8 hrs	Std.	t
	1 LWT+MWQ	67.6	2.20	1.284
	2 LWT+LIT	66.3	0.97	1.572
Push	3 LWT+MRIQ	66.3	1.18	1.292
	4 MROQ+MWQ	66.0	1.92	0.638
	5 MROQ+LIT	63.2	1.76	-0.895
	6 MROQ+MRIQ	64.8	1.12	0.022
	7 MWQ+LWT	63.8	1.37	-0.712
	8 LIT+LWT	63.6	0.98	-1.199
Pull	9 MRIQ+LWT	64.8	1.80	0.014
	10 MWQ+MROQ	62.8	1.37	-1.442
	11 LIT+MROQ	63.3	1.20	-1.229
	12 MRIQ+MROQ	64.8	1.76	0.014
	Overall Average	64.8		

Table 3.5 shows the simulation result in average output rate under the above conditions. The shop locking was not observed at all. In addition to mean values, estimated standard deviations and t statistics are included in the table. The t statistics are calculated under the hypothesis that the average output rate of a rule is same as the overall average output rate, 64.8 per 8 hours. From the table, it is shown that there is no significant difference between the average output rate of any rule and the overall average output rate at 5% significant level since all $|t| < t(8; 0.025) = 2.306$.

Although the significant difference between the average output rate of any rule and the overall average output rate is not noticed, a pairwise comparison was done to see if any rule is particularly better. The statistical procedure is outlined as follows (Cox, 1987):

$$H_T : \mu_i - \mu_j = 0$$

$$H_0 : \mu_i - \mu_j \neq 0$$

$$t_{ij} = \frac{\overline{y}_i - \overline{y}_j}{\sqrt{s_i^2 + s_j^2}}$$

$$f_{ij} = \frac{(s_i^2 + s_j^2)^2}{s_i^2/(n_i - 1) + s_j^2/(n_j - 1)}$$

Note that t_{ij} and f_{ij} are the t statistic and the degree of freedom respectively for the hypothesis that average output rates of rule i and rule j are same. Also, $\overline{y}_i$, s_i, and n_i are the estimated mean and standard deviation of average output rate, and the number of samples from the simulation output with rule i. The statistics, as represented in the Table 3.6, show that there is no significant difference in average output rate between any two rules since $|t_{ij}| < t(f_{ij}; 0.025)$, for all i and j.

Concluding Remarks

A Petri net-based simulation model of an FMS with an AGV system was developed and used to investigate the effect of the AGV dispatching rules on the system performance. Average output rates was used to compare the performance of 12 vehicle-initiated AGV dispatching rules. Although it is difficult to develop a basic model, based on the simulation analysis, which will include all different possibilities, the following conclusions can be made:

1. The shop locking can be effectively decreased by decreasing the number of pallets in the system, thus, the system performance can be maximized. But,

Table 3.6: t statistics under the hypothesis that average output rates of a pair of rules are same

Rule	2	3	4	5	6	7	8	9	10	11	12
1	0.541 (11)	0.521 (12)	0.548 (16)	1.562 (15)	1.134 (12)	1.466 (13)	1.661 (11)	0.985 (15)	1.852 (13)	1.716 (12)	0.994 (15)
2		0.000 (15)	0.139 (12)	1.543 (12)	1.012 (16)	1.489 (14)	1.958 (16)	0.734 (12)	2.085 (14)	1.944 (15)	0.746 (12)
3			0.133 (13)	1.463 (14)	0.922 (16)	1.383 (16)	1.760 (15)	0.697 (14)	1.936 (16)	1.783 (16)	0.708 (14)
4				1.075 (16)	0.540 (13)	0.933 (14)	1.113 (12)	0.456 (16)	1.357 (14)	1.192 (13)	0.461 (16)
5					-.767 (14)	-.269 (15)	-.199 (13)	-.636 (16)	0.179 (15)	-.047 (14)	-.643 (16)
6						0.565 (15)	0.806 (16)	0.000 (13)	1.130 (15)	0.914 (16)	0.000 (14)
7							0.119 (14)	-.442 (15)	0.516 (16)	0.275 (16)	-.448 (15)
8								-.586 (12)	0.475 (14)	0.194 (15)	-.596 (13)
9									0.884 (15)	0.693 (14)	0.000 (16)
10										-.275 (16)	-.897 (15)
11											-.704 (14)

* The figure in the paranthesis represents the degree of freedom.

too small number of pallets decreases the output rate due to limited part flow in the system.

2. In a busy FMS, the vehicle-initiated rules both push-based and pull-based rules perform equally well in terms of average output rate when the shop-locking is significantly reduced by (1) restricting the number of pallets to the level that provides the maximum output rate, and (2) avoiding the load movement which has high possibility of causing shop-locking.

In this study, the performance of push-based and pull-based AGV dispatching rules were investigated when the FMS was set in a busy state. The behavior of the dispatching rules in a non-busy shop would be an interesting extension of the study.

References

[1] P. J. Egbelu and J. M. A. Tanchoco, "Characterization of AGV Dispatching Rules," *Int. J. of Production Research*, Vol. 22, No. 3, 1984.

[2] P. J. Egbelu, "Pull Versus Push Strategy for Automated Guided Vehicle Load Movement in a Batch Manufacturing System," *J. of Manufacturing Systems*, Vol. 6, No. 3, 1987.

[3] Roberta S. Russel and J. M. A. Tanchoco, "An Evaluation of Vehicle Dispatching Rules and Their Effect on Shop Performance," *Material Flow*, 1984.

[4] Ishan Sabuncouglu and Don L. Hommertzheim, "An Investigation of Machine and AGV Scheduling Rules in a FMS," *Proc. of the Third ORSA/TIMS Conference on Flexible Manufacturing Systems*, Elsevier, Amsterdam; New York, 1989.

[5] Onur M. Ülgen and Pankaj Kedia, "Using Simulation in Design of a Cellular Assembly Plant with Automatic Guided Vehicles," *Proc. of the 1990 Winter Simulation Conference*, 1990.

[6] R. K. Miller and T. C. Walker, *FMS/CIM Systems Integration Handbook*, The Fairmont Press, Inc., Lilburn, GA, 1990.

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

[8] Hidemi Kodate, Ken'ichi Fujii, and Fuzuyoshi Yamanoi, "Representation of FMS with Petri Net Graph and its Application to Simulation of System Operation," *Robotics and Computer Integrated Manufacturing*, Vol. 3, No. 3, 1987.

[9] T. C. E. Cheng, "A Simulation Study of AGV Dispatching," *Robotics and Computer Integrated Manufacturing*, Vol. 3, No. 3, 1987.

[10] Carolyn L. Beck and Bruce H. Krogh, "Models for Simulation and Discrete Control of Manufacturing System," *Proc. IEEE Int. Conf. on Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1986.

[11] Giorgio Bruno and Maurizio Morisio, "Petri Net based Simulation of Manufacturing Cells," *Proc. IEEE Int. Conf. on Robotics and Automation*, 1987.

[12] R. Valette, "Nets in Production Systems," *Lecture Notes on Computer Science*, 255, Springer-Verlag, Berlin; New York, 1984.

[13] Javier Martinez, Pedro Muro, and Manuel Silva, "Modeling, Validation and Software Implementations of Production Systems Using High level Petri nets," *Proc. IEEE Int. Conf. on Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1987.

[14] Antonio Camurri and Macercello Frixione, "Structured Representation of FMS Integrating SI-NETS and High Level Petri Nets," *Applied Artificial Intelligence*, Vol. 4, 1990.

[15] C. P. Cox, *A Handbook of Introductory Statistical Methods*, John Wiley & Sons, New York, 1987, pp 59.

[16] W. L. Maxwell and J. A. Muckstadt, "Design of Automatic Guided Vehicle Systems," *IIE Transactions*, Vol. 14, No. 2, 1982.

Appendix: The Minimum Number of AGVs

The desirable number of AGVs to accomplish the given load movements (it is assumed that target production plan and job routings are known) must be determined when the AGV dispatching rules are investigated. Because too many AGVs may create a higher possibility of collision and blocking and hence prohibit the efficient control of AGVs, the minimum number of AGVs needed to perform the assigned tasks was considered in order to minimize the effect that the number of AGVs has on the dispatching rule performance. Maxwell and Muckstadt (1982) constructed a linear programming model to obtain the minimum number of AGVs. But, they ignored random effects in the system, specifically, the location of AGVs and parts at a given time. The following section describes an extended procedure to determine the minimum number of AGVs needed considering the random effects under steady state when the idle time of the AGVs is ignored. With the information on the minimum number of AGVs determined analytically, the minimum number of AGVs considering the time-dependent effects will be determined from the experimental simulation.

Let $F(i,j)$ be the required flow matrix from i to j node (i.e., pick-up or delivery points) for the movements of parts during a specified working time, T_w. The required flow matrix is obtained from the target production rate and job routings. Let $D(i,j)$ be the transportation time from i to j node obtained by the shortest route. It is assumed that there are always parts waiting for an AGV's service. The probability that the AGV is waiting at j node in steady state, $Pw(j)$, is

$$Pw(j) = \sum_{i=1}^{n} F(i,j) / \sum_{i=1}^{n} \sum_{j=1}^{n} F(i,j)$$

Also, the probability that i-th pick-up node calls an idle AGV in steady state, $Pc(i)$,

is

$$Pc(i) = \sum_{j=1}^{n} F(i,j) / \sum_{i=1}^{n} \sum_{j=1}^{n} F(i,j)$$

Complete movement of a load includes (1) empty vehicle moving to a pick-up point, (2) pick up a part, (3) moving to a drop-off point with the loaded part, and (4) delivery.

The required average transportation time for the empty vehicle, T_v, is

$$T_v = \sum_{i=1}^{n} \sum_{j=1}^{n} Pw(j) \cdot Pc(i) \cdot D(j,i) \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} F(i,j)$$

And, the required average transportation time for loaded vehicle, T_p, is

$$T_p = \sum_{i=1}^{n} \sum_{j=1}^{n} F(i,j) \cdot D(i,j)$$

Letting l and u be the pick-up and delivery time of a part at each work station, the pick-up and delivery time for total part flow, T_l, is:

$$T_l = \sum_{i=1}^{n} \sum_{j=1}^{n} F(i,j) \cdot (l+u)$$

And the minimum number of AGVs required to accomplish the load movements during T_w can be obtained:

Minimum number of AGVs $= (T_p + T_v + T_l)/T_w$

GENERAL SUMMARY

In this dissertation, a computer-aided simulation tool was presented. It is based on Conserved nets which are a subclass of Petri nets to provide a formal and graphical modeling language. The proposed Conserved nets are shown to be a good modeling tool for the analysis and design of FMSs. They provide a simple analysis procedure for the properties of Petri net models such as conservativeness, liveness, safeness, and boundedness. Furthermore, simulation models can be easily obtained under the modeling logic of Conserved nets. To facilitate the simulation modeling process, the Petri net objects (places and tokens) are hierarchically classified to correspond to hardware components of FMSs such as machines, AGVs, robots, pallets, and buffers.

Conserved nets are not appropriate for modeling high-level control systems. These control systems are difficult to represent in Petri nets including Conserved nets. That is the reason why the real-time control rules in high-level control systems of FMSs are separately modeled. In executing a Petri net model, several conflicts may occur. To resolve the conflicts, additional procedures are required. The high-level control systems are modeled separately using a control specification language and integrated with a Petri net model so that they resolve conflicts in Petri net execution.

In this simulation tool, several systems such as Petri net modeling, control rule

modeling, token player, and output analysis have been developed in order to be extended and interfaced with other future systems. It was programmed by using the Turbo-C language under a micro-computer with MS-DOS and EGA graphics.

Generally, there are several advantages in the Conserved net-based simulation tool.

1. Flexibility in modeling hardware components of FMSs
2. Simplicity in modeling process
3. Simple development of simulation executive
4. Modularity of simulation program
5. Animation

It is uncertain how the modeling power of Conserved nets compares with general-purpose simulation languages (e.g., GPSS, SLAM, SIMAN, etc.) in creating simulation models, however, like the general-purpose simulation languages, the Conserved nets provide more flexible models than automatic code generators (Haddock, 1987; Mathewson, 1985 and 1989) and special-purpose simulation packages. Furthermore, the modeling processes will be aided by classified Petri net objects which correspond to hardware components of FMSs. The simulation executive (token player) can be easily implemented by employing transition enabling and firing rules. The Petri net objects and separated modules of high-level control rules make the simulation program modular, hence, the modification and enhancement of the program is easily performed. Animation of the simulation model is accomplished by animating Petri net graphs. As shown with several examples, Conserved net models resemble the

physical configuration of FMSs. Additional models or graphic display of the simulation model are not required.

To accomplish a successful simulation of FMSs, there are still several requirements that need further study.

1. Extended Petri net objects
2. Models of diverse control rules
3. Output analysis specially for the simulation of FMSs
4. Experiment aid (e.g., capacity planning for a given FMS prior to the simulation).
5. Optimizing design combined with the simulation

In this study, 13 place objects are classified to model hardware components of FMSs. We do not believe that any real FMS can be modeled efficiently by these place objects. More extensive and diverse-purpose place objects are required. As discussed in Part II, we are considering an additional tool, a Petri net object definition system, for creating new Petri net objects. According to the application, a modeler may define the required place objects and token objects under the system. Then users could make a model using the predefined objects.

The proposed control rule specification language has limitations to represent more complex rules. We are considering an interface mechanism with-other systems (e.g., expert system).

It is believed that the output analysis in simulation of FMSs has its own characteristics. For example, the state of a system may not be steady even if the average

number of jobs in the system is stable (even constant). This results from the fact that there is a constant number of pallets in the system. To detect the steady state, special procedures are required.

The optimal design of FMSs with simulation (Nandkeolyar and Christy, 1989; Floss and Talavage, 1988; Mellichamp and Wahab, 1987; Talavage and Hannam, 1988) is a difficult job because the simulation is an evaluative technique: it only provides estimates of performance measures. To obtain the optimal decision regarding the design of FMSs, an evaluative technique must be interfaced with a generative procedure which generates alternative sets of decisions. Usually this procedure is time-consuming because generating alternative decisions requires a large number of simulation runs. To achieve an automatic design procedure, the experiment support system, output analysis system, new alternative generating system, and automatic model modification system must be integrated with the simulation software.

BIBLIOGRAPHY

[1] Peter C. Bell and Robert M. O'Keefe, "Visual Interactive Simulation– History, Recent Developments, and Major Issues," *Simulation*, Vol. 49, No. 3, 1987.

[2] Paul Bratley, Bennett L. Fox, and Linus E. Schrage, *A Guide to Simulation*, Springer-Verlag, Berlin; New York, 1987.

[3] R. Spinelli De Carvalho and John G. Crookes, "Celluar Simulation," *O. R. Quarterly*, Vol. 27, No. 1, 1976.

[4] Springer Cox, "Interactive Graphics in GPSS/PC," *Simulation*, Vol. 49, No. 3, Sep., 1987.

[5] Georgios I. Doukidis and Ray J. Paul, "Research into Expert Systems to Aid Simulation Model Formulation," *Operational Research Society*, Vol. 36, No. 4, 1985.

[6] John B. Evans, "Discrete Event Simulation Package for Modeling Entities with many Aspects– A re-appraisal of the Activity Approach," *Proc. of the Summer Computer Simulation Conference*, 1981.

[7] Mark S. Fox, Nizwer Husain, Malcolm McRoberts and Y. V. Reddy, "Knowledge-Based Simulation: An Artificial Intelligence Approach to System Modeling and Automating the Simulation Life Cycle," *Artificial Intelligence, Simulation, and Modeling*, John Wiley & Sons, New York, 1989.

[8] F. Hank Grant, "Simulation in Designing and Scheduling Manufacturing Systems," *Design and Analysis of Integrated Manufacturing System*, National Academy Press, Washington, DC, 1988.

[9] Jorgr Haddock, "An Expert System Framework based on a Simulation Generator," *Simulation*, Vol. 48, No. 2, 1987.

[10] James O. Henriksen, "The Integrated Simulation Environment(Simulation Software of 1990s)," *O.R.*, Vol. 31, 1983.

[11] Averill M. Law and W. David Kelton, *Simulation Modeling and Analysis*, McGraw-Hill, New York, 1982.

[12] Averill M. Law, "Simulation of Manufacturing System," *Proc. of the 1988 Winter Simulation Conference*, 1988.

[13] J.E. Lenz, "MAST:A Simulation as Advanced as the FMS it Studies," *1st Proceedings of International Conference on Simulation in Manufacturing*, IFS, Bedford, England, 1985.

[14] William R. Lilidgon and Jean J. O'Reilly, "SLAM-II for Microcomputer," *Modeling and Simulation on Microcomputer*, 1985.

[15] S. C. Mathewson, "The Application of Program Generator Software and Its Extensions to Discrete Event Simulation Modeling," *AIIE Transactions*, Vol. 16, March, 1984.

[16] S. C. Mathewson, "Simulation Program Generators: Code and Animation on a P.C.," *J. Opl. Res. Soc.*, Vol. 36, No. 7, 1985.

[17] S.C. Mathewson, "Simulation Support Environments," in *Computer Modeling for Discrete Simulation (M. Pidd ed.)*, John Willey & Sons, New York, 1989.

[18] Kevin D. Reilly, Warren T. Jones, and Pradip Dey, "The Simulation Environment Concept Artificial Intelligence Perspectives," *AI and Simulation*, 1985.

[19] Robert G. Sargent, "A Tutorial on Validation and Verification of Simulation Models," *Proc. of the 1988 Winter Simulation Conference*, 1988.

[20] Lee Schruben, "Simulation Modeling with Event Graphs," *Communications of the ACM*, Vol. 26, No. 11, 1983.

[21] Robert E. Shannon, Richard Mayer, and Heimo H. Adelsberger, "Expert Systems and Simulation," *Simulation*, June, 1985.

[22] Jonathan Billington, Geoferey R. Wheeler, and Michael C. Wilbur-Ham, "PROTEAN: A High-level Petri Net Tool for the Specification and Verification of Communication Protocols," *IEEE Transactions on Software Engineering*, Vol. 14, No. 3, 1988.

[23] K. P. Brand and J. Kopainsky, "Principles and engineering of process control with petri nets," *IEEE Trans. Automatic Contr.*, Vol. 33, No. 2, 1988.

[24] Antonio Camurri and Paolo Franchi, "An Approach to the Design and Implementation of the Hierarchical Control system of FMS, Combining Structured Knowledge Representation Formalisms and High-Level Petri nets," *Proc. IEEE Int. Conf. on Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1990.

[25] D. Crockett, A. Desrochers, F. DiCesare, and T. Ward, "Implementation of a Petri Net Controller for a Machining Workstation," *Proc. IEEE Int. Conf. on Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1987.

[26] Frits Feldbrugge, "Petri net Tools," *Lecture Notes on Computer Science*, 222, Springer-Verlag, Berlin; New York, 1985.

[27] Peter J. Haas and Gerald S. Shedler, "Stochastic Petri Net Representation of Discrete Event Simulations," *IEEE Transactions on Software Engineering*, Vol 15, No. 4, 1989.

[28] Masaki Hasayuki takada, takashi Tommyo, and Hideo Matsuka, "Modeling of Exception Handling in Manufacturing Cell Control and Its Application to PLC Programming," *Proc. IEEE Int. Conf. on Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1990.

[29] Mohsen A. Jafari, "Petri bet based Shop Floor Controller and Recovery Analysis," *Proc. IEEE Int. Conf. Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1990.

[30] Kurt Jensen, "Coloured Petri Nets," *Lecture Notes on Computer Science*, 255, Springer-Verlag, Berlin; New York, 1986.

[31] K. Jensen, "How to find the Invariants of Colored Petri Nets," in *Mathematical Foundations of Computer Science, Lecture Notes on Computer Science*, 118, Springer-Verlag, Berlin; New York, 1981.

[32] Kurt Jensen, "Computer Tools for Construction, Modification and Analysis of Petri Nets," *Lecture Notes on Computer Science*, 255. Springer-Verlag, Berlin; New York, 1986.

[33] M. Kamath and N. Viswanadham, "Applications of Petri Net-Based Models in the Modeling and Analysis of FMSs," *Proc. IEEE Int. Conf. on Robotics and Automat.*, IEE Computer Society Press, Washington, DC, 1987.

[34] Y. Edmund Lien, "Termination Properties of Generalized Petri Nets," *SIAM J. Computing*, Vol. 5, No. 2, 1976.

[35] A Aziz Merabet, "Synchronization of Operations in a Flexible Manufacturing Cell: The Petri Net Approach," *Journal of Manufacturing Systems*, Vol. 5, No. 3, 1986.

[36] Jerre D. Noe, "Hierarchical Modeling with PRO-NETS," *Proc. 14th Design Automation Conference*, 1977.

[37] C. V. Ramamoorthy and Gary S. Ho, "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets," *IEEE Transactions on Software Engineering*, Vol. SE-6, No. 5, 1980.

[38] Ramarathnam Ravichandran and Amiya K. Chakravarty, "Decision Support in Flexible Manufacturing Systems Using Timed Petri Nets," *Journal of Manufacturing Systems*, Vol. 5, No. 2, 1986.

[39] A Saharoui, H. Atabakhche, M. Courvoisier, and R. Valette, "Joining Petri Nets and Knowledge-Based Systems for Monitoring Purposes," *Proc. IEEE Int. Conf. on Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1987.

[40] Josepth Sifakis, "Structural Properties of Petri Nets," *Lecture Notes on Computer Science*, 64, Springer-Verlag, Berlin; New York, 1978.

[41] Kimon P. Valavanis, "On the Hierarchical Modeling Analysis and Simulation of Flexible Manufacturing Systems with Extended Petri Nets," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 20, No. 1, 1990.

[42] R. Valette and H. Atabakhche, "Petri Nets for Sequence Constraint Propagation in Knowledge Based Approaches," in *Concurrency and Nets*, Springer-Verlag, Berlin; New York, 1987.

[43] Rudiger Valk, "On the Computational Power of Extended Petri Nets," *Lecture Notes on Computer Science*, 64, Springer-Verlag, Berlin; New York, 1978.

[44] Meng Chu Zhou and Frank Dicesare, "Adaptive Design of Petri Net Controller for Error Recovery in Automated Manufacturing Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 19, No. 5, 1989.

[45] Christopher V. Jones, "An Introduction to Graph-based Modeling Systems, Part I: Overview," *ORSA Journal on Computing*, Vol.2, No. 2, Spring, 1990.

[46] F. Azadivar and J. Talavage, "Optimization of Stochastic Simulation Models," *Mathematics and Computers in Simulation*, Vol. 22, 1980.

[47] Stephen S. Lavenberg (Chairperson), "The Initial Transient in Steady State Simulation (Panel Discussion)," *1981 Winter Simulation Conference Proceedings*, 1981.

[48] Averill M. Law, "Statistical Analysis of Simulation Output," *O. R.*, Vol. 31, No. 6, 1983.

[49] Udayan Nandkeolyar and David P. Christy, "Using Computer Simulation to Optimize Flexible Manufacturing System Design," *1989 Winter Simulation Conference proceedings*, 1989.

[50] Dennis E. Smith, "Automatic Optimum-seeking Program for Digital Simulation," *Simulation*, 1976.

[51] T. S. Chan and H. A. Pak, "Modeling of a Controller for a Flexible Manufacturing Cell," in *Simulation Applications in Manufacturing*, IFS, Bedford, England, 1986.

[52] G. Chryssolouris, K. Wright, J. Pierce, and W. Cobb, "Manufacturing Systems Operation: Dispatch Rules Versus Intelligent Control," *Robotics and Computer Integrated Manufacturing*, Vol. 4, No. 3/4, 1988.

[53] Peter Floss and Joseph J. Talavage, "A Knowledge-based Design Procedure for Flexible Manufacturing Systems," *AI and Simulation*, 1988.

[54] Donnie R. Ford and Bernard J. Schroer, "An Expert Manufacturing Simulation System," *Simulation*, May, 1987.

[55] Joseph M. Mellichamp and Ahmed F.A. Wahab, "An Expert System for FMS Design," *Simulation*, May, 1987.

[56] Paul Rogers and David J. Williams, "A Knowledge-based System Linking Simulation to Real-time Control for Manufacturing Cells," *Proc. IEEE Int. Conf. on Robotics and Automat.*, IEEE Computer Society Press, Washington, DC, 1988.

[57] Josepth Talavage and Roger G. Hannam, *Flexible Manufacturing Systems in Practice*, Marcel Dekker, New York, 1988.

ACKNOWLEDGEMENTS

I wish to thank professor Thomas Barta for his encouragement and understanding throughout preparing this dissertation. Despite of my weak communication, he patiently listened to me and reviewed my dissertation. I am very much indebted to my committee members–Dr. H. T. David, Dr. John Jackman, Dr. George Strawn, and Dr. John Wacker. Their stimulating guidance, suggestion, and critics encouraged me in carrying out this research. I also thank Dr. Linn who gave much advise in preparing Part III of this dissertation. I could not forget Dr. Kwak; he introduced me to the exciting fields of Petri nets, and gave valuable comments regarding study at ISU.

Finally, I appreciate my family for their love, trust, and support. This dissertation would not have been completed without the trust and understanding of my wife.

APPENDIX: TOKEN PLAYERS

Generally, discrete-event simulation systems are classified into three main families according to the strategy of modeling and simulation execution: event-based, activity-based, and process-interaction. For the simulation executive (token player) of Petri nets, these approaches can be implemented. Simply, a Petri net model can be considered to consist of sets of events, activities, and relationships between events and activities. The event (transition) can be executed (fired) only when it meets the transition enabling conditions. Therefore, a Petri net is executed simply by scanning all transitions at each cycle until no more transitions can be fired. This approach is actually based on the activity-scanning strategy. It is not surprising that some Petri net simulation software adopted the activity-based approach. To implement this approach, some considerations are necessary to reduce the computation time in scanning all of the transitions at a time beat. This was done by scanning the transitions that have at least one input place marked with tokens (Alanche, et al., 1984: refer to Part II), or by extending three-phased approach (Evans, 1981).

Event-based approach can also be implemented. Only the transitions which satisfy the enabling conditions are stored in a current event list. When a transition in the current event list is fired, the current event and future event lists are updated by checking the neighborhood transitions connected by the places involved in the

transition firing. The time scan is carried by fetching a record in the future event list.

The process-interaction approach views the process as the temporary entity flow (e.g., parts in FMSs). In the application of token player of our Petri net exploited in this thesis, the process is viewed for the active tokens. Therefore, another approach is possible by manipulating future event list which stores places containing active tokens in processing state, and current event list containing places which have active tokens in waiting state. But, like activity-scanning approach, the transitions connected with places in the current event list should be scanned at a time beat.

The developed simulation tool provides two token players: Transition-based token player and Place-scanning token player. The Transition-based token player is based on the event-based approach, and Place-scanning approach combines the activity-scanning and process-interaction approaches.

Transition-based token player

For this token player, two-doubly linked lists are constructed: Firing Transition List (FTL) and Processing Place List (PPL). FTL contains the records for those transitions which meet the transition enabling conditions. The transition records are linked in the decreasing order of waiting time of tokens in the input places of that transition. PPL contains the records for those places in which active tokens in processing state are marked. The place record includes two attributes; place number and scheduled finishing time of process imposed to an active token in that place. The records are linked in the order of these times. The token player is executed with the following procedures.

Phase 1: Transition firing

1.1 Fetch and remove the transition at the head of FTL.

1.2 Perform arrival and departure processes for the tokens involved in the firing of the fetched transition. Update FTL and PPL.

1.3 Repeat 1.1 and 1.2 until there is no record in FTL.

Phase 2: Time advance

2.1 Fetch and remove the record at the head of PPL. Advance simulation clock to the scheduled finishing time of the record. Update FTL.

2.2 Repeat 2.1 while simulation clock equals the scheduled finishing tome of the fetched record.

2.3 Check the termination of simulation. If the current status meets the termination condition, then stop. Otherwise, go to Phase 1.

Place-scanning token player

For the place-scanning token player, two doubly-linked lists are constructed: Waiting Place List (WPL) and Processing Place List (PPL). WPL contains the records for those places in which active tokens in waiting state are marked. It has two main attributes: place number and the time switched to a waiting state in the place. The records in WPL are linked in decreasing order of this time.

Phase 1: Place scanning

1.1 Fetch and remove one by one from the head of WPL, and check if any output transition of the fetched place can be enabled. If the transition meets the enabling conditions, perform departure and arrival procedures according to the transition firing rules.

1.2 Repeat 1.1 until there is no place in WPL whose output transitions can be enabled.

Phase 2: Time advance

2.1 Fetch and remove the record at the head of PPL. Advance simulation clock to the scheduled finishing time of the record. Update FTL.

2.2 Repeat 2.1 while simulation clock equals the scheduled finishing tome of the fetched record.

2.3 Check the termination of simulation. If the current status meets the termination condition, then stop. Otherwise, go to Phase 1.

Comparison of complexity

Two token players presented are different each other. To compare the complexity, the following notations are used.

k: the number of tokens in a net

n_1: the average number of output arcs of a place

n_2: the average number of input arcs of a place

n_3: the average number of output arcs of a transition.

n_4: the average number of input arcs of a transition

m: the average number of transitions to fire at a time beat

For a performance measure, the maximum number of places to be checked in order to fire m transitions in a time beat is considered.

1. Place-scanning token player

$$\begin{aligned} \text{Complexity} &= kn_1(n_3 + n_4)(m + 1) \\ &= 2kn_3(m + 1), \text{ if } n_1 = n_2, n_3 = n_4. \end{aligned}$$

2. Transition-based token player

$$\begin{aligned} \text{Complexity} &= (n_3 + n_4)^2(n_1 + n_2)m \\ &= 8n_1n_3^2m, \text{ if } n_1 = n_2, n_3 = n_4. \end{aligned}$$

Usually, a Petri net model has same number of input arcs and output arcs at each node (i.e., $n_1 = n_2$ and $n_3 = n_4$). In this case, the transition-based token player is better when $k(m + 1) > 4n_3m$.

Under the place-scanning token player, all marked active tokens in waiting state have to be evaluated at each cycle. This gives in efficiency in computation time as in the case of activity-scanning approach. It may scan unnecessary places. Rather than scanning all places or marked places, only the active tokens in waiting state are evaluated at each cycle (note that passive tokens cannot move voluntarily without combination with active tokens). This reduction in evaluation provides efficient execution in case of a busy shop in FMS because very few active tokens are in waiting state at a time.

The transition-based token player can provide efficient computation time when small number of arcs is connected with transitions. To update the FTL at each transition firing, however, it is necessary to check the transition enabling conditions for the neighborhood transitions.