1994

Petri net approaches for modeling, controlling, and validating flexible manufacturing systems

Bong Wan Choi
Iowa State University

Petri net approaches for modeling, controlling, and validating flexible manufacturing systems

Choi, Bong Wan, Ph.D.

Iowa State University, 1994

Petri net approaches for modeling, controlling, and validating flexible manufacturing systems

by

Bong Wan Choi

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
1994

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I would like to express thank to my God first.

There are a number of people who provided assistance and support my doctoral studies. Without their help, the successful execution of my study would have been most difficult.

Dr. Way Kuo deserves the first acknowledgment. I am especially grateful to Dr. Kuo for his understanding, eagerness, patience and encouragement throughout my doctoral studies.

I also express my appreciation to Dr. Doug Gemmill, Dr. Young W. Park, Dr. Richard Linn, Dr. Doug Jacobson and Dr. Irvin Hentzel for helpful advice and serving as research committee.

I also thank the Korean Navy for supporting me during my doctoral studies.

Finally, I would like to say, the honor of this dissertation should go to my wife shin-ju, Lee and my daughter Ji-hyun (Karan) and Rebecca because of their assistances and sacrifices.

GENERAL INTRODUCTION

Background

The doctoral dissertation of Carl Adam Petri in 1962 introduced a initial theoretic concepts of Petri nets and discussed the basis for a theory of communication between synchronous components of a computer system. Petri was particularly concerned with describing the causal relationships between events that can be occured in a computer system. This work thus began the development of Petri nets into the large body of research and development existing today. Initially, many researchers studied Petri nets with respect to theories and applications. The use and study of the Petri nets have spread, and many research projects and conferences on Petri nets (e.g., Information System Theory Project at Rome Air Development Center, Griffiss Air Force Base, New York and Project MAC at the Massachusetts Institute of Technology) have come to our attention.

Further, the research areas of Petri nets have spread and have expanded to depth in theory and width in application on the basis of the mathematical as well as the graphical tool. Recently, many authors who have entered the research of Petri nets have provided not only suitable platforms in the areas of modeling and design of concurrent systems, information systems, manufacturing systems, and performance analysis, but also their own experience and understanding of various extended models and applications. Also an excellent society has been established for sharing new research results with other researchers who are concerned about the field of Petri nets. These environments continuously made me to have strong attention, and gave motivations and backgrounds to study the areas of Petri nets as a Ph.D. dissertation.

Motivations and Objectives

The basis of Petri nets is to model graphically and test analytically the discrete events of concurrent operations within a computer system. Of course, the fundamental constructs of Petri nets are useful to model and analyze manufacturing systems. Since 1962, many researchers have studied Petri nets and published reports and papers in many diverse areas, and their contributions also have been growing impressively in a versertile scientific and engineering activities. However, many researcher's works have only partial representation Petri net theories and applications with respect to background, sequential studies, and application areas, and also their works have been difficult to access easily. Therefore, we present four papers that describe Petri nets studies in order of studies, in details, and in well-organize with accomplishments of three objectives: (1) study the fundamental constructs of Petri nets that can be visualized, analyzed, and validated for a discrete system, production system, manufacturing system, or as controller in a flexible manufacturing system, (2) suggest some extensions that help make Petri nets useful for modeling and analyzing discrete event systems and manufacturing systems models (3) Validation methods are presented for these models, and results of a performance analysis from a deterministic and atochastic model are used to reorganize and re-evaluate a manufacturing system in order to increase its flexibility.

Organization of the Dissertation

The four papers included in this dissertation introduce the fundamental ideas and constructs of Petri net models, extend these models based on the context of a versatile manufacturing system, and apply extended Petri nets models to several manufacturing systems such an assembly cell, an Automated Palletized Conveyor System, and a tooling machine to show increased modeling power and efficient analysis methods.

In **the first paper**, the fundamental constructs of the Petri nets (ordinary, timed, colored, stochastic, control, and neural) are reviewed, and suggested extensions that help make Petri nets useful for modeling and analyzing discrete event systems and manufacturing systems models are introduced. We then present some studies that emphasize Petri nets theories and applications as extended research fields that provide suitable platforms in modeling, controlling, validating, and evaluating concurrent systems, information systems, and a versatile dynamic system and and manufacturing systems. Finally, we introduce methods for reducing Petri net models.

In **the second paper**, we introduce the fundamental constructs of Petri net models. We then extend these models and apply them to manufacturing systems. Validation methods are presented for these models. In addition, results of a performance analysis from a deterministic model are used to reorganize and re-evaluate a manufacturing system in order to increase its flexibility. In **the third paper**, we present an approach to modeling, analyzing and evaluating an Automated Palletized Conveyor System (APCS) using extended Petri net models. We first examine the

APCS and extend the fundamental constructs of Petri net models. We then build a Petri net model of the given APCS, analyze important qualitative aspects of APCS behaviors, and finally evaluate performances of the APCS.

A modified deterministic and stochastic algorithm is developed to describe and evaluate the Petri net model of the given APCS. The input and control mechanisms of the Petri net model are varied, implemented, and evaluated to produce results that can be used to redesign the APCS and also can be directly applied to the design and analysis of the full-scale material handling operation.

In **the fourth paper**, we have studied the tool changing problem that arises in flexible manufacturing environments. We introduce and review this problem as an overall model that can be formulated as a linear and non-linear integer problem. We then extend this model on the basis of two more constraints: (1)jobs that require more than C tools, with C representing the magazine capacity of the machine, and (2)the increased processing time that is required for tuning the tool offset after a tool in slot #1 is changed. Since this model increases computational complexity, we propose a heuristic approach for job sequencing. This approach is locally optimized to minimize the number of tool changes.

Next, we introduce the fundamental constructs of the Petri net models to describe sequence control specifications for a flexible manufacturing machine. We then examine a flexible manufacturing cell that has two automated guided vehicles and a milling machine (DM4400) with an automatic tool changer. Finally, we build a Petri net model as the interpretation schema and implementation model with respect to the local optimal job sequence, the tool changing-procedure, and the machining job of the milling machine.

PAPER I.

PETRI NETS: A STATE-OF-THE ART REVIEW

ABSTRACT

Petri nets have been used successfully to model, control, and analyze discrete event dynamic systems that are characterized by: concurrency or parallelism; asynchronous processes; distributed, deterministic and/or stochastic, deadlocks, conflicts, and event driven-processes. Petri nets are also particularly valuable when modeling and analyzing versatile manufacturing systems because they provide accurate models and efficient analysis methods based on they (1) capture interactions of concurrent and sequential events, (2) are logical models derived from the knowledge of how systems work, (3) give concise models for conflicts and buffer sizes, and (4) can be used to implement real-time analysis.

In this paper, the fundamental constructs of the Petri nets (ordinary, timed, colored, stochastic, control, and neural) are reviewed as they developed, and suggested extensions that help make Petri nets useful for modeling and analyzing discrete event systems and manufacturing systems are introduced. We then present some studies that emphasize Petri nets theories and applications as extended research fields that provide suitable platforms in modeling, controlling, validating, and evaluating concurrent systems, information systems, and manufacturing systems. Finally, we introduce methods for reducing Petri net models.

INTRODUCTION

History

The doctoral dissertation of Carl Adam Petri in 1962 [1] discussed the basis for a theory of communication between synchronous components of a computer system. Petri was particularly concerned with describing the causal relationships between events [2]. This work thus began the development of Petri nets into the large body of research and development existing today. Initially, many researchers studied Petri nets with respect to theories and applications. The use and study of the Petri nets have spread, and many research projects and conferences on Petri nets (e.g., Project MAC at the Massachusetts Institute of Technology) have come to our attention [2].

Further, the research areas of Petri nets have spread and have expanded to depth in theory and width in application on the basis of the mathematical as well as the graphical tool. Recently, many authors who have entered the research of Petri nets have provided not only suitable platforms in the areas of modeling and design of concurrent systems, information systems, manufacturing systems, and performance analysis, but also their own experience and understanding of various extended models and applications. Also an excellent society has been established for sharing new research results with other researchers who are concerned about the field of Petri nets.

Objective

The basis of Petri nets is to model graphically and test analytically the discrete events of concurrent operations within a system. Of course, the fundamental constructs of Petri nets are useful to model and analyze manufacturing systems. The objectives of this paper are (1) to study the fundamental constructs of Petri nets that can be visualized, analyzed, and validated for a discrete system, production system, manufacturing system, or as controller in a flexible manufacturing system, and (2) to introduce a state-of-the-art review of Petri nets and their applications in depth and width of their useful fields of Petri nets.

Advantages and Disadvantages of Petri Nets

Kamath and Viswanadham [3] have introduced several positive aspects of Petri nets for general dynamic systems:

- they describe the modeled system graphically and hence enable an easy visualization of complex systems,
- Petri nets can model a system hierarchically (systems can be represented in a top-down fashion at various levels of abstraction and detail),
- a well-developed Petri net analysis techniques provide a systematic and complete qualitative analysis of the system
- the existence of well-formulated schemes for Petri net synthesis facilitates system design and synthesis, and timed Petri nets can evaluate system performance.

Peterson [2] wrote further advantages of Petri nets as a model of parallel computation. In large part, Petri nets are receiving increased attention because of their simplicity coupled with a careful balance of modeling power and decision power. The modeling power of Petri nets is quite good, as witnessed by the wide variety of systems that can be modeled by Petri nets. The decision power is also good, since the reachability problem is decidable, and most problems can be converted into reachability problems. Agerwala [4] further states that Petri nets include the ability to model at every level of the system, something which most other design languages cannot do.

More recently, Al-Jaar and Desrochers [5,6] introduced the concept that Petri nets are useful for the modeling , performance evaluation, and control of manufacturing systems with the following characteristics:

- Concurrency or parallelism

  In a manufacturing system, many operations that are enabled and do not interact take place at the same time.

- Asynchronous and synchronous

  Machines need variable amounts of time to complete their operations, and this variability must maintain the partial ordering of the occurrence of operations.

- Deadlock

  A state can be reached where none of the processes can continue. This situation can occur when two jobs share two resources. More specifically, when one job takes the first resource and the other job takes the second resource, then both jobs cannot go further because two resources are occupied. The order in which two resources are used and released for two jobs needs to be arranged.

- Conflict

  This may occur when two or more jobs require a common resource at the same time. For example, two workstations might share a common transport system or might simultaneously want access to the same database.

- Event driven

  The manufacturing system can be viewed as a sequence of discrete events. Since operations occur concurrently, the order of occurrence of events is not necessarily unique; it is one of many allowed by the system structure.

- Real-time control

  Petri net models can also be used to implement real-time control systems for an automated manufacturing system. They can sequence and coordinate the subsystems like a programmable logic controller does.

- Mathematical foundation

  Petri net models have a well-developed mathematical foundation that allows a qualitative and quantitative analysis of the system.

In addition, the Petri nets offer convenient ways of expressing system behavior that are both asynchronous and distributed, compared with other common graph models of dynamic behavior (such as the state transition diagram of a finite-state machine, or the PERT/CPM network). Petri nets also provide a solid platform for the precedence constraints among operations and relaxed couplings associated with shared resources, as well as the repetition or sequences of certain operations. Finally, the Petri nets can be analyzed in a formal way to obtain information about the dynamic behavior of the modeled system [7-9].

In addition to advantages of Petri nets, Peterson [2] summarized some of the disadvantages as follows:

- The concurrency of operations has become more and more common. This has generally improved utilization and throughput, but consequently increases the complexity.
- Subclasses of Petri nets increase the decision power, but at a cost of being unable to model a large number of systems. Extended Petri net models increase the modeling power, but in all known cases at the expense of decision power, since most analysis questions become undecidable.
- Petri net models have limitations in their inability to test for exactly a specific marking in an unbounded place and to take action on the outcome of the test.

PETRI NETS

Ordinary Petri Nets

- Structure of a Petri net

A Petri net (PN) is a four-tuple, **PN** = (P,T,I,O), where P = { p_1, p_2, ..., p_n, } is a set of places, T = { t_1, t_2, ..., t_n, } is a set of transitions, I is an input function, and O is an output function. The set of places and the set of transitions are disjointed as, $P \cap T = \emptyset$, $I \subseteq \{ P * T \}$, and $O \subseteq \{ T * P \}$ are sets of directed arcs.

A place p_i is an input place of a transition t_j if $p_i \in I(t_j)$; p_i is an output place if $p_i \in O(t_j)$. The structure of a Petri net is defined by its places, transitions, input function, and output function, as shown in Figure 1.

- Petri net graph

More theoretical work on Petri nets is based on the formal definition of Petri net structures. However, a graphical representation of a Petri net structure is much more useful for illustrating the concepts of the Petri net.

A Petri net graph uses circles to represent places (states) and bars to represent transitions (events). Input-output relationships are represented by directed arcs between places and transitions. An arc directed from a place p_i to a transition t_j defines the place to be an input of the transition. Multiple inputs to a transition are indicated by multiple arcs from the input places to the transition. An output place is indicated by an arc from the transition to the place. Again, multiple outputs are represented by multiple arcs. A Petri net is a multigraph, since it allows multiple arcs from one node of the graph to another. In addition, since the arcs are directed, the Petri net is a directed multigraph. Since the nodes of the graph can be partitioned into two sets (places and transitions), such that each arc is directed from an element of one set (place or transition) to an element of the other set (transition or place), the Petri net is a bipartite directed multigraph [2]. We refer to it simply as a Petri net graph.

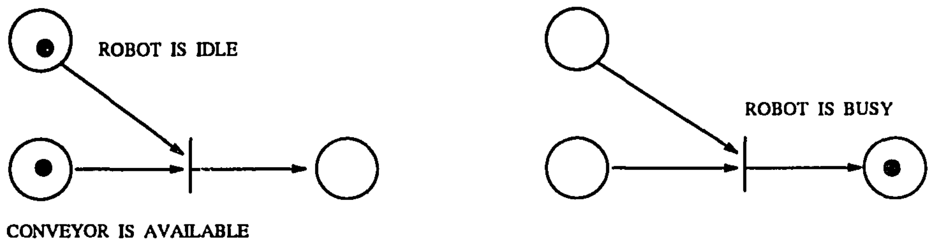
- Marked Petri net (= **M**)

  A Petri net **M** containing a marking μ is a marked Petri net, **M** = (P,T,I,O,μ). Marking μ of a Petri net **PN** is a function from the set P to a set of non-negative integers **N**, μ: P $\rightarrow$ **N**, where μ sets tokens to every place, $\mu_i = \mu(p_i \in$ N indicates the number of tokens in place p_i). We denote a marked Petri net (= **M**) by (**PN**, μ). We generally associate an initial marking μ_0 with a given **M**.

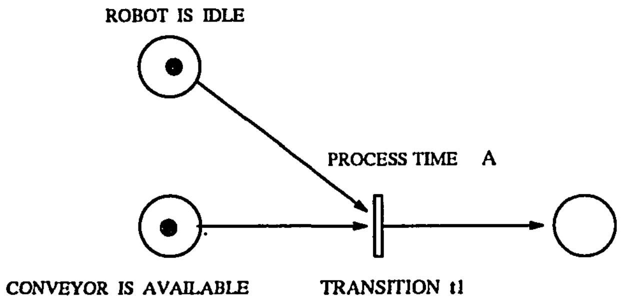
  Tokens reside at a place when it is active. Tokens flow through the net depending on the present marking of the net. The marking of a Petri net is contained in a vector of dimension n, where n is the number of places and each value of the vector corresponds to the number of tokens in the corresponding place. Figure 1(a) shows an example of marking for PN where dots represent tokens. For the example of Figure 1(a), μ_0 = (1,1,0) and μ_1 = (0,0,1).

- Dynamic behavior

  When there is a token in each of the input places of a transition, that transition is enabled to fire. If the weights on each of the arcs between places and transitions are equal to one, then the transition fires by removing a token from each of its input places and by placing a token in each of its output places.



(a) Petri net example



(b) Timed Petri net example

Figure 1: Ordinary and Marked Petri net example

– State (= **S**) of the Petri net

Marking $\mu = (\ \mu_1, \mu_2, \ldots, \mu_n\ )$ is also called the state of a Petri net. Let state **S** be

$$\mathbf{S}=(\ s_1, s_2, \ldots, s_n)$$

where

$$s_i = \begin{cases} 1, & \text{if } \mu_i = \mu(p_i) > 0, \\ 0, & \text{if } \mu_i=\mu(p_i) = 0 \end{cases}$$

The state **S** shows whether a place has tokens or not.

– Incident matrix and firing rules

Let **INS** be the (m,n) $\rightarrow$ { 0,1, . . ., i } function that defines the multiplicity of an input place of the transition. Also let **OUTS** be the (m,n) $\rightarrow$ { 0,1, . . ., o } function that defines the multiplicity of an output place of the transition. If the **M** has no self-loops, then the m $\times$ n incident matrix **D** defined by **D** = **OUTS** - **INS** characterizes the relationship between places and transitions. Therefore, the functions of INS and OUTS of Petri net **M** are represented by two matrices D^- and D^+. Each matrix has m rows for each place and n columns for each transition. The incident matrix **D** is defined by **D** = $D^+[j,i]$ (= # (p_i, I(t_j)) - $D^-[j,i]$ (= # (p_i, O(t_j)) [2].

Now a transition t_j is enabled in marking μ if

$$\mu \geq e[j] \times D^-$$

where e[j] is the unit m-vector, which is zero everywhere except in the jth component.

The result of firing transition t_j in marking μ, if it is enabled, is

$$\mu - e[j] \times D^- + e[j] \times D^+,$$
$$= \mu + e[j] \times (-D^- + D^+),$$
$$= \mu + e[j] \times D.$$

Figure 1(b) also shows a Petri net example for accessing a robot. The tokens, places, and transitions must be assigned a meaning for proper interpretation of the model. In a manufacturing system, places usually represent resources (e.g., machine, part, and data). A token in a place indicates that the resource is available; otherwise it is unavailable. A place can also be used to imply that a particular condition holds. Transitions are generally used to represent the initiation or termination of an event.

Timed Petri Nets

A strength of the Petri net formalism is that it provides a set of simple constructs that can model a wide variety of systems. However, a major weakness of ordinary Petri nets is that they provide no way to represent the passage of time. Indeed, assumptions are made regarding the amount of time it takes to complete the different processes. Tokens move in a manufacturing system according to the transition firings, which have a given processing time.

Ramchandani [10] and Sifakis [11] introduced the notion of a timed Petri net (TPN). Ramchandani described a timed Petri net as a pair (PN,η), where PN is a Petri net and η is a vector of processing time functions that assigns a positive rational number to each transition of the net. In a timed Petri net model, each transition t_i (after being enabled) has a time delay of $\eta(t_i)$ before firing. The firing times must be rational so that one can discretize the processing times in units of time and precisely describe the state of the process at each instant of time. The rule of operation of a TPN is similar to an ordinary PN. Once a transition is enabled, the tokens are removed from the input places and are held for a time, $\eta(t_i)$, after which the tokens are sent to all the output places. Transitions in TPN can be viewed as a list of events where multiple sets of tokens can be at different stages of the time delay. The firing and termination occur during the processing time $\eta(t_i)$ and at the end of the execution, respectively.

A transition associated with time $\eta(t_i)$ is graphically represented using a bar [], which indicates that a token stays in that transition for a processing time $\eta(t_i)$. Figure 4(a) shows a timed Petri net example using a robot. In the figure, transition t_1 has an associated process time A. The entire model is controlled through the use of a global clock to time events. The transition t_1 has a time delay of A before firing. If multiple transitions become enabled, they fire simultaneously. In Figure 4(a), when the job and the robot are both available, the firing execution of the transition t_1 occurs during process time A. Process time A represents the time it takes for the conveyor to move a part to the robot. Figure 4(b) shows the TPN equivalence for the firing time of the transition t_1 that can be associated with the place p(a) in the following manner: When transition t_1 is enabled to fire, t(a) fires, a token is removed from each input place of t(a), and a token is deposited on place p(a). This token stays in p(a) for the process time A. At the end of this interval, transition t(b) fires, corresponding to the termination of the transition t_1.

The TPN studied by Ramchandani has deterministic processing times with transitions and approximate bounds on the transition firing rates for more general TPN. That study was generalized by Sifakis, who considered deterministic processing time and obtained the same results as Ramchandani, but Sifakis handled more general Petri nets with multiple arcs between nodes. Sifakis showed that the distinction between associating processing times with transitions or with places was not important, since one type of TPN can be converted into the other. However, Sifakis' work is effective for the case of the deterministic processing time, but not for the random processing time. Ramamoothy and Ho [12] showed how to obtain the same results given by Sifakis and Ramchandani. Cohen et al. [13] showed that decision-free TPN with deterministic processing times can be analyzed by using (max, +) algebra results.

Another aspect of TPN was introduced and demonstrated by Merlin [14] and Menasche [15], respectively. Merlin states the fixed duration d of an event can be simply modeled as $[d,d]$. The basic PN formalism can be represented as a special class where the bounds on all events are $[0, \infty]$.

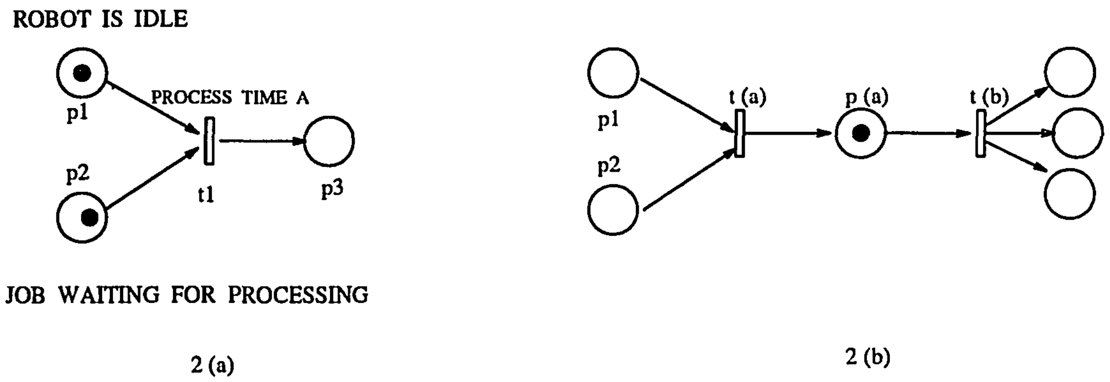


Figure 2: Timed Petri net example: (a)timed Petri net example using a robot, (b)timed Petri net equivalence

Merlin's analysis of TPN begins with the enumeration of the token machine of the nontimed PN. Then by using the time information in the TPN, deleted selective parts of the token machine. The result was inspected to verify certain properties that characterize a well-behaved telecommunication protocol.

But this sort of analysis is unsuitable when the systems exhibit much concurrency. For example, consider two transitions t_1, t_2 with firing delay [0,4],[2,5] that were enabled together. If we call that time instant 0, the t_1 may fire immediately, while t_2 must wait at least until time instant 2 to fire ; however, both transitions might fire together at the time instants 2,3,4. Therefore, the number of possible combinations of firing for even these two transitions is large. But the transitions may fire in parallel, implying that their firings are not related.

Some of the works discussed above can be extended to analyze TPN with random processing times by replacing these times by their expected values. However, the results obtained in this way provide only very loose approximations of the average firing rate. Several researchers have tried to remedy this situation by converting the TPN into an equivalent Markov chain and then analyzing the resulting Markov chain. Zuberek [16] was the first researcher to perform this transformation and was able to analyze a stochastic timed Petri Net (STPN), which only allowed very simple decision rules based on independent probabilities. Razouk and Phelps [17] extended Zuberek's work to STPNs that can model time-out situations where the completion of one activity may disable others, and to slightly more complex decision situations. The decision rules are still based on independent probabilities. Both of these articles, however, fail to show that the resulting Markov chain has a well-defined steady-state probability distribution, and their procedures are applicable to only very small problems.

Molly [18] solved somewhat larger problems by associating exponential processing times with transitions and by specifying a decision rule that stated that the transition whose processing time terminates first would fire. Marson et al. [19] extended Molly's results to manage transitions with zero processing times.

As indicated before, the main weakness of all works mentioned above is that they need to construct an equivalent Markov chain modeling the evolution of the marking of the net to find the performance measures of interest.

Colored Petri Net

Colored Petri net (CPN) is an extension to ordinary Petri nets in which colors are associated with tokens, and transitions fire according to a set of rules that match the appropriate colors. A colored token is analogous to a subscripted variable. The advantage of colored Petri nets is that they provide compact models of large systems.

Jensen has introduced and defined the CPN [20-23], whose main ideas are the relation between an occurrence color and token colors (which were involved in the occurrence of the transition). The relation is defined by functions attached to the arcs [21]. In addition, the CPN attaches a set of possible token colors to each place and a set of possible occurrence colors to each transition. The CPN can be defined in the following ways.

A CPN is a six-tuple, CPN = (P,T,C, I_-,I_+,M_0), where C is the color function that can be defined from P $\cup$ T into nonempty sets, and color function can be attached to each place and to each transition as mentioned before. Also I_- and I_+ are the negative and positive incidence-functions defined on P $\times$ T, such that I_-(p,t), I_+(p,t) $\in$ (a set of possible occurrence colors to each transition $\rightarrow$ a set of possible token colors to each place). The initial marking M_0 is a function defined on P, such that M_0(p) $\in$ a possible set of token-colors in each place.

A CPN graph can be drawen with two disjointed sets of nodes (places and transitions). Any pair of a place and a transition may be connected with a set of directed arcs. There exists a set of typed variables that has a name and a type. Moreover, each arc has attached to it an arc expression, containing a set of variables. Each place p has attached to it a nonempty set of token colors and initial markings, and each transition has attached to it a predicate circumspect that can only contain those variables which are already in the immediate surrounding arc expressions.

The CPN allows the modeler of systems with repetitive processes to view a smaller network in which tokens have changed color to indicate process steps, assign attributes, or differentiate between tokens. The primary function of CPN is data management. The structure of the Petri net systems are not affected nor are the reachability trees or analysis questions. The color of tokens is just another data item carried in the markings. The colors represent levels of activity or the number of times the part has moved through the process. This model concept is also useful when several parts must be processed through the same system. An example occurs in electronic chip manufacture where wafers being fabricated pass through five basic processes many times adding layers of new material onto existing layers. The processes are represented by states, the pass number is represented by the color of the tokens, and the transition represents the movement between processes.

Viswanadham and Narahari [24] give two detailed examples on the use of CPNS in automated manufacturing. The first is concerned with the modeling and analysis of two machines that process two part-types. The machines and three robots process the two part-types by using a limited number of shared tools.

Alla and Ladet [25] illustrated the use of CPN as a modeling, validation, and simulation tool by using a flexible manufacturing line with first-in first-out queues.

Martinez et al. [26] turned their attention to the level above the local control: the coordination (cell) level. Monitoring and real-time scheduling was the third level. By using the Renault flexible manufacturing system (FMS) layout, a CPN model for the co-ordinator was derived and analyzed. To solve any conflicts in the local control level model, the researchers suggested the use of an expert system, especially since production rules can be modeled by using the CPN. Also, fault (or error) detection and recovery can be similarly modeled, either at the same or higher levels.

Gentina and Corbeel [27] proposed the use of structured adaptive and structured colored adaptive Petri nets to model FMS and their control systems, at the two lowest levels. The third level was modeled as a rule-based (declarative) expert system. The method of analysis and validation is illustrated by using an FMS.

Using a similar approach of the CPN, Choi and Kuo [28] represented token color as token shape, indicating the identity of a product, part, and resource, and the like, in manufacturing systems. For example, a place can have one or more token shapes that represent a set of different resources. These sets can be used to model a system with n different resources that provide different capabilities. A transition can fire with respect to each of these shapes. Transition firing follows the rules of ordinary Petri nets except that token shapes must match. In other manufacturing environments, there is a many-to-many mapping between product type and resource. This mapping is a direct result of capability requirements of products and available capabilities of resources. If different products (representing different token shapes) require one resource, then the transition cannot fire because the token shape is not matched. We then assign letters to the tokens (a, b, c, ... , x, y, z representing different capabilities). In this case, the transition firing rule will be changed as follows: (1) basically, a transition can fire with respect to the same token; (2) when a different token shape is matched, transition firings follow the rules of ordinary Petri nets except that the letter much be checked. If the letter is matched with the different token shapes, then the transition can be fired.

Stochastic Petri Nets

In stochastic Petri nets (SPNs), places represent resources and transitions represent operations that are associated with random-operation time variables. The stochastic behavior must be completely defined by a set of rules associated with the choice of the next transition to be fired in a given marking [29-31]. The SPN has been defined by many authors who have attempted to extend the modeling power of the SPN by assigning different versions of variable times with the transitions, for example, zero time and exponentially distributed random variables. These occur in Generalized Stochastic Petri nets, in random variables with different distributions allowing inhibitor arcs, in probabilistic arcs (extended SPNs) [32] and in stochastic activity networks [33]. In this paper, extended SPN so-called generalized stochastic Petri nets (GSPN), is introduced in the following ways: A GSPN has eight-tuple, **GSPN** = (P,T,Pr,I,O,Inh,Sc, M_0), where the priority function Pr represents the priority level of any transitions $t_i \in T$, the inhibition function Inh is represented by circle-headed arcs connecting every place p_i to the transition t_i [32], and Sc is the stochastic function of the GSPN and Sc_i (i.e., random variables assigned for any transition $t_i \in T$). The dynamic operation of a GSPN is equivalent to the behavior of a continuous-time stochastic process [34].

The stochastic Petri net was initially defined by Molloy [35] in following two classes of firing distributions: (1) exponential for the continuous time systems and (2) geometric for discrete time systems. These distributions are memory-less in the following sense. Consider a marking in which several transitions are enabled. If the firing delay of each such transition is modeled in an exponential and geometric way, then when one transition fires, the distribution of the time delays associated with other enabled transitions remains unchanged. Molloy then demonstrated that such stochastic PNs are isomorphic to continuous-time Markov processes and can therefore by analyzed using classical Markovian techniques. In particular, a Markov chain can be generated to describe the possible markings of the nets (i.e., the reachability set), and the probabilities associated with moving from one marking to another. In this way, the steady-state probability distribution of the markings can be computed using Markovian techniques. Now using the marking probabilities and the number of tokens in each place for each marking, we can obtain the token probability density function that is the steady-state probability distribution of tokens in each place.

Molloy also has proposed a discrete-time stochastic PN [36] with a system-wide clock to advance time in a discrete way. Since multiple transitions may fire at any time step, the probabilities for each possible combination are determined. For each such enabled transition, a conditional probability must be computed that it will fire at the next time step (whether or not the other enabled transitions fire). Thus, before the normal Markovian analysis can begin, the conditional probabilities must be deconditioned.

Finally, let us consider the GSPN compared with queueing networks (QN) and simulation. Simulation allows us to represent precisely a real time system but it is a resource-consuming tool and the model validation is quite difficult. For example, performance optimization of FMS with respect to a number of decision variables requires a large number of simulation runs. Also multiple simulation and output analysis are required to reduce the simulation error because of the randomness involved in system operations (such as failure rates), and severing, arrival, and departure rate of the queue. Thus the simulation method could be computationally very demanding [24].

However, GSPNs are graphical models and provide a compact framework for modeling and validating, a convenient way to express system behaviors, and a suitable mathematical and/or statistical analysis.

Another analytical model of systems is a QN that takes into account system dynamics, interactions, and uncertainties inherent in versartile systems. Also efficient computational algorithms are available for solving QN models. However, QN only allows modeling purely parallel behaviors. In other word, complex qualitative behaviors (synchronization schemes) cannot be described in the basic QN model, compared with SPN or GSPN that consider much more complex behavior [37].

Control Petri Nets

Petri nets can represent only two statuses corresponding to the token's existence or nonexistence, while machine actions usually have plural status depending on the results of their execution. To avoid this problem and to apply the Petri net model for describing sequence control specifications, control nets will be introduced.

Control Petri nets (CPN) are a modified form of the ordinary Petri net(P,T,I,O) where I and O represent input and output arcs by Murata, et al [38,39]. Before these studies, Valette, et al. [40] proposed Petri net-based sequence description languages and executed them directly on a microcomputer-based controller without applying for real process control and examination to evaluate their response time in real- time control.

CPN model Control Petri net (CPN) models are introduced based on initial works [15-18] as defined by the tuples CPN = (P,T,I,O, δ, φ, η, θ, ϑ, ι, U,V,M), where

U, V (system status functions) represent execution status at places and transitions, and δ, φ, η, θ, ϑ, and ι (input-output process functions) represent process status. The system status functions allow supervision of the execution status and management of the transition and place statuses, and input-output process functions are used to allow an operator direct control of token movement in the system. This is an example of modeling enhancements quickly limiting the decision and analysis attributes of Petri net models. In order to define a corresponding place and transition in a CPN and the controllable and observable process in a FMS, several functions are needed as follows.

- Definitions

  Let C be a set of control signals (c_i) and O be a set of observable signals (o_{ij}); similarly let CH be a set of checking signals (ch_i) and J be a set of judgment signals (j_i). Input-output process functions δ: $\mathbf{T} \rightarrow \mathbf{C}$ φ: $\mathbf{T} \rightarrow \mathbf{O}$ θ: $\mathbf{P} \rightarrow \mathbf{CH}$ ϑ: $\mathbf{P} \rightarrow \mathbf{J}$ η: $\mathbf{T} \rightarrow \mathbf{O}$ ι: $\mathbf{P} \rightarrow \mathbf{J}$ are defined as follows:

$$\delta(t_i) = c_i, (c_i \in \mathbf{C}, t_i \in \mathbf{T}) \quad (1)$$

$$\varphi(t_i) = o_{i1}, o_{i2}, \ldots, o_{in}, (o_{ij} \in \mathbf{O}, t_i \in \mathbf{T}) \quad (2)$$

$$\eta(t_i) = o_{ij}, (o_{ij}, \in \mathbf{O}, t_i \in \mathbf{T}) \quad (3)$$

$$\theta(p_i) = ch_i, (ch_i \in \mathbf{CH}, p_i \in \mathbf{P}) \quad (4)$$

$$\vartheta(p_i) = j_{i1}, j_{i2}, \cdots, j_{im}, (j_{ij} \in \mathbf{J}, p_i \in \mathbf{P}) \quad (5)$$

$$\iota(p_i) = j_{ij}, (j_{ij} \in \mathbf{J}, p_i \in \mathbf{P}) \quad (6)$$

- Input process function

  When a token enters into a transition t_i, a control signal c_i defined by $\delta(t_i)$ triggers a machining action. Then the token waits to fire in a transition until one of the input signals o_{ij} defined by $\varphi(t_i)$ is shown for completion of a machining action. Input signal o_{ij} defined by $\eta(t_i)$ is used for firing a transition. After detecting input signal o_{ij}, the transition can fire and the token moves to its output places.

- Output process function

  The checking signal ch_i is defined by $\theta(p_i)$ that corresponds to plural statuses on the basis of results of the machining actions in output places. By using the checking signal ch_i , the checking operation is started. Also the token waits to fire in a place until one of the input judgment signals shows completion of a checking action like the input process function. The signals j_{ij} are defined by $\vartheta(p_i)$ that corresponds to its completion of a judgment, including quality specifications. Input signal j_{ij}, defined by $\iota(p_i)$, is used for firing a place. After detecting an input signal j_{ij}, a place can fire and the token move to its output transitions.

- Process status functions

  In order to define the execution status at a transition and a place, and in order to manage the transition and place the open and close statuses and the process status functions [15] The parameters U:**P** $\in$ **L**(**L** =0,1, ..., m), V:**P** *in* **N**(N =0,1) are introduced as follows:

$$U(t_i) = \begin{cases} \textit{in} & \text{action associated with } t_i \text{ is executing now} \\ \textit{out} & \text{action associated with } t_i \text{ is completed with return code } o_{in} \end{cases} \tag{7}$$

$$U(p_i) = \begin{cases} \textit{in} & \text{checking associated with } (p_i) \text{ is executing now} \\ \textit{out} & \text{checking associated with } (p_i) \text{ is completed} \end{cases} \tag{8}$$

$$V(t_i) = \begin{cases} \textit{close} & t_i \text{ is closed} \\ \textit{open} & t_i \text{ is opened} \end{cases} \tag{9}$$

$$V(p_i) = \begin{cases} \textit{close} & p_i \text{ is closed} \\ \textit{open} & p_i \text{ is opened} \end{cases} \tag{10}$$

When an output signal c_i defined by $\delta(t_i)$ has been put out in the transition, U(t_i) is set at *in*. When one of the input signals o_{ij} defined by $\varphi(t_i)$ is detected, U(t_i) is set at the value of *out*. If an input signal o_{ij} defined by $\eta(t_i)$ has not been detected, the value of V(t_i) is set at *0*; otherwise, V(t_i) is set at *open*. Similarly, after the token in transition is moved into its output places, if an output signal ch_i defined by $\delta(p_i)$ has been checked, then U(p_i) is set at *in*, otherwise U(p_i) is set at *out*. If an output signal j_i defined by ι_i has not yet been detected, the value of V(t_i) is set at *0*; otherwise, V(t_i) is set at *open*.

By introducing these functions, execution statues or transition operation modes can be supervised and controlled at a place and transition. In this paper, places and transitions are called CPN-transitions and CPN- places (represented by the fat box and the fat circle) since the process input-output functions and process status functions can be defined at places and transitions.

- The token firing rule in CPN-transitions and CPN-places

  A token in all input CPN-places p_i of the transition $t_i \in$ T can be enabled at each marking M(p_i)=1, if and only if,

$$\begin{aligned} V(p_i) &= \mathit{open}, \text{ and} \\ U(p_i) &= \mathit{out} \end{aligned} \tag{11}$$

  A token in CPN-transition $t_i \in$ T can be enabled if and only if,

$$\begin{aligned} V(t_i) &= \mathit{open}, \text{ and} \\ U(t_i) &= \mathit{out} \end{aligned} \tag{12}$$

- Other functions

  We have more complicated sequence control specifications for the machining processes such as conditional branches based on the result of a machining action and timing control. A CPN place can have several output transitions and the output transition to be fired is selected according to the result of machining actions. In addition, a time value can be assigned to the token and can be used to evaluate time factors such as production time and rate.

Neural Petri Nets

Another extension of Petri nets is from the biological brain, for example, sequential, parallel, interconnected, and self-organizing systems. More specifically, we extend the Petri nets to recognize aspects of brain architecture and its processing. The basic brain-processing unit is the nerve cell, called the neuron, which has the basic components such as the axon, synapse, cell body, dendrites; the cell's functions and dynamics processing have been examined to developed the model representing the brain system [41].

The Petri net has been chosen as the basis for the model since the type of structure and behavior it represents corresponds to the functioning of the fundamental neural elements in the human brain. The basic neural Petri net (NPN) has been defined by Zargham and Tyman [42] to accommodate all the elements of the neural system as follows:

A neural Petri net is a system **N = (P,T,A,S,F,q,n,g,h,c)**
where:

- **P** is a finite nonempty set of distinctly labeled places (p_1, p_2, ..., p_n).
- **T** is a definite nonempty set of distinctly labeled transitions (t_1, t_2, ..., t_n).
- **A** is a relation which corresponds to a set of arcs where each arc is either from a place to a transition or from a transition to a place A.
- **S** is a finite, nonempty set of starting (or initial) places that is a subset of p.
- **F** is a finite, nonempty set of final (or output) places that is a subset of p.

- **q** is a real number that indicates the lifetime of a token in a place.
- **n** is an integer number that indicates the minimum number of tokens required to fire a transition.
- **g** is a function that calculates the total token value in each place at every unit of time.
- **h** is a function that associates a color with each output arc of a transition.
- **c** is a set of two colors c^1 and c^2 to represent two classes of tokens.

The body of a neuron is represented by a place. Each neuron has its own membrane potential; the membrane potential is represented by a transition that has only one input place. The output arcs, from a transition to its output places, represent the axons of the neuron represented by input place of the transition. To represent the fact that an axon transmits either an excitatory postsynaptic potential and an inhibitory postsynaptic potential at the synapse, an output arc will be assigned a color. Each axon synapse has only one neuron, but a neuron may have numerous axons. Therefore, a transition may have numerous output places [41,42].

The firing of a transition corresponds to the generation of an action potential in the associated input place. As a consequence of the firing of a transition, a token appears in each output place of the transition. The token has the same color as the output arc of the transition on which it was transmitted [42].

Using the NPN, researchers can model the physiology and the structural constitution of the brain functions and apply this to computing systems and manufacturing systems. However, the NPN that was developed and introduced in [42] is a beginning stage for simplification of certain neural processes.

ANALYSIS OF PETRI NETS

A major reason for using Petri nets versus other modeling systems is the ability to test and validate a system. Liveness, boundedness, safeness, invariants, and reachability are measures of effectiveness for the Petri nets.

Definition 1: A PN is live with respect to an initial marking M_0 if from any marking in M_0, there exists for each transition a firing sequence leading to a marking in which that transition is enabled.

Definition 2: A PN is safe or 1-bounded if the token count of every place is always 0 or 1.

More specifically, if the number of tokens in all places is always bounded by some finite value k, then the PN is called k - bounded. The boundedness in PN is determined by whether or not a given PN is bounded for a given initial marking.

Definition 3: A PN is conservative if there exists an n non-negative integer vector x such that

$$x^T M = x^T M_0 \qquad (13)$$

for any initial marking M_0 and a reachable marking M $\in$ R(M_0). This indicates that the sum of the tokens weighted by x is constant.

Definition 4: A PN is reversible if for every M $\in$ R(M_0); then M_0 $\in$ R(M). This indicates that the initial marking is reachable from all reachable markings.

Invariants

Other important properties of the logical structure of a PN are invariants that characterize in some way all possible firing behaviors. There are two kind of invariants, so-called *p*-invariant and *t*-invariant. P-invariants are associated with places and represent unchanging truths about sets of conditions, such as mutual exclusion. Conversely, *t*-invariants are associated with transitions and represent collections of transitions that leave the marking of the PN unchanged after firing of transitions. These invariants can be analyzed based on the incidence matrix **A**. For a PN is described in Section 2.1.1, the incidence matrix **A** $(=a_{ij}^{+} - a_{ij}^{-})$ is an $m \times n$ matrix of integers defined as A = a_{ij}, i = 1,2.....m(transitions) and j = 1,2.....n(places). The invariants of a PN may be obtained as the integer solutions to the following equations:

$$\mathbf{A}^T W = 0, \text{ for } p\text{-invariants} \qquad (14)$$

$$\mathbf{A} W = 0, \text{ for } t\text{-invariants} \qquad (15)$$

where W be a n x 1 column vector that indicates the weighted sum of token in a PN.

Deadlock occurs when a transition cannot fire and no sequence of transition firing will take the net to a marking that allows the transition to fire. A Petri net is live if there is no deadlock [1,2].

Reachability Tree

A reachability tree is generated from an initial marking by firing enabled transitions. Reisig [43] and Peterson [2] discuss this in detail. Essentially, if the reachability tree shows no infinite markings (places containing or having the potential to contain an infinite number of tokens) then the tree is bounded and safe. The reachability tree is a finite representation of the usually infinite reachability set from an initial marking of the Petri net.

The reachability problem deals with the ability to reach a marking from an initial marking. Although the reachability problem is extremely difficult to solve, recent results seem to indicate that it is solvable. Thus, although the problem can be solved in general, it may take too much time and money to be worthwhile. Other problems, such as the equality of the reachability sets of two Petri nets(useful for considering equivalence and optimization), are known to be unsolvable.

APPLICATIONS OF PETRI NETS

Petri Net Models for Production Systems

Applications of Petri nets in the manufacturing are varied. The usefulness of PN logic in system design, control, testing, simulation, and analysis is demonstrated using PN systems.

Petri's 1962 dissertation [1] in 1972 came to the attention of project MAC at MIT; numerous reports and dissertations resulted. Hack [44] brought together the many facets of the production environment and provided a broad approach to manufacturing systems overview and design in Petri nets. Most importantly, the work describes the use of free choice Petri nets and their definitions. This work puts together the nuts and bolts of production schemata and is the seminal work relating Petri nets directly to a manufacturing environment. Since that time, work has branched into several different areas such as manufacturing systems, flexible manufacturing systems or cells, and controllers.

The next particular interest is two subclasses of the PN. The PN is defined as a *marked or event graph* when each place has just one input and output transition. The marked graph may model a deterministic system because each condition can only become true in one way and can have one consequence. The marked graphs are useful for modeling because the firing of a single transition may change the truth of multiple conditions.

Conversely, the PN is defined as a *state machine* when each transition has just one input and output place. The state machine is useful for modeling limited forms of conflicts, since a given condition may enable multiple events.

Robotics and flexible manufacturing systems The need for formal models in robotics was recently discussed [45]. The simplest model is the condition event (C/E) PN in which the robotic operations to be executed are presented as events whose interdependencies are expressed as conditions. Events are then associated with transitions and conditions with places in the PN. The presence of a token in a place indicates that the corresponding condition is true. The marking describes the state of the PN in term of conditions that are true or not.

In addition, tokens might represent the number of input parts waiting at the head of an assembly line or the number of spaces available in a storage. In this case, subtracting a token from a place by firing a transition and adding to a place does not necessarily make a condition true or false.

However, when the system is composed of many operating elements as in an FMS, the large numbers of conditions and events required for modeling prefer colored PN [21,22,46]. Tokens are now interpreted as variables representing different classes of objects whose individuals are represented by different colors. Some examples of FMSs working with colored PN are described in [47-50].

To date, most of the published PN-related research has been concerned with the routing of parts and tool magazines between work cells within an FMS with deterministic events.

Petri Net Based Controllers

Design and implementation Houldsworth and Brearly [51] state that programmable logic controller (PLC) functions can be enhanced using a Petri net-like programming system and a structured method of programming. The authors maintain that good methods and a graphical approach to programming are used effectively with a personal computer to provide good documentation of the system.

Many other researchers have studies the implementation of PN. Brand and Kopainsky [52] and Bruno and Marchetto [53] have used Petri nets as the basis for process control design. Gentina and Corbeel [27] proposed a modified net system for the synthesis of FMS control. Devanathan et al. [54] developed computer-aided design of relay ladder logic (RLL) using state diagrams. Krogh and Beck [55] investigated the use of nets for simulation of manufacturing systems and Krogh, et al [56] proposed using Petri net and microcomputers to generate programming for PLC functions keeping a data-base of used and available data points. The work of Ref 56 allows the retention of RLL but keeps the programming level at a higher level language. Lloyd [57] discusses GRAFACT, a program which uses Petri nets as a basis for PLC programming. These researchers promote a function block approach to states and transitions. The elemental functionality of PNs is not considered or demonstrated; only their higher order modeling is used. The function blocks and transitions are evaluated or fired based on RLL written at a lower level. Martinez, et al [58-60] developed a language for describing concurrent systems such as FMSs and a package for computer design of concurrent logic systems. They discussed using Petri nets to specify of FMS. Matsuzaki et al. [61] used a GRAFCET-like Petri-structured programming system similar to BASIC and reported increases in software productivity of 50% to 100% over RLL, as did Murata, et al [38] who reported a 50% steff-hour reduction in software development time from RLL by using their PN-oriented language.

Controllers Chocron and Cerny [62] have provided the first example of implement of a PN- based controller. Their example is paralleled by the work of Courvoisier et al. [63,64] in developing a PN based-controller on a Z80-based computer. Naturally, this implementation is slow due to the CPU speed and memory size. Advances in semiconductors is expected to improve the processing time. Meanwhile, Murata et al. [38,39,65] have presented work on a PN-based controller implementation on a large CPU controlling a FMS cell. Their findings indicate good control and quick comprehension of the graphics used in the control monitor allowing rapid fault detection and repair with increases in software productivity of 50% over the RLL systems. Crockett et al. [66] expanded upon all of the previous work to implement a PN-based controller at a higher level of control, generating software control commands used to interact with hardware signals.

Performance Analysis

The Petri net model of manufacturing systems is not sufficient to answer performance-related questions. TPN can be used to analyze performance of computer systems and manufacturing systems. In TPN, the firing of a transition takes a certain amount of time that is deterministic. The same properties of the ordinary Petri nets can applied to TPN.

The stochastic Petri nets, which have the transition with randomly distributed firing rates, also can be used to analyze performance of the computer system and of the automated manufacturing system. In addition, STPN has modeling power as well as the powerful ability to build analogies based on the initial marking; thus, the reachability tree can be generated and the equivalent Markov chain can be obtained and analyzed.

Chiola [67] has developed software called Great SPN for the construction and analysis of complex, generalized SPN models. This software accepts deterministic delays or exponentially distributed firing rates. It also computes the transient and steady-state solutions to the Markov chains. Dugan et al. [68] have developed the Duke extended SPN evaluation package (DEEP) for performance analysis of SPN models which in turn led to a more recent version [69]. Holliday and Vernon [70] have developed the Great TPN analyzer for the performance evaluation of generalized timed Petri net models.

REDUCTION OF PETRI NETS

In some cases, modeling a system with Petri nets can lead to a large number of places, transitions, and arcs. To obtain insight into the operation of the original system, it is desirable to find a net with fewer places, transitions, and arcs that retains the liveness and boundedness properties of the original Petri nets model. By studying and analyzing the reduced Petri net, it is possible to make conclusions about the token flow and structural properties of the original Petri net. Several authors [71-75] proposed reduction methods to accomplish this for generalized Petri nets. The goal of the reduction method is to establish a set of rules for combining places, transitions, and arcs that preserves the number and direction of flow of tokens in the original Petri net. The flow of tokens into and out of a reduced Petri net must be the same as in the portion of the original Petri net that has been replaced.

In Refs. [71-75], researchers developed generalized Petri net reduction methods for real applications of the Petri net models that will become large. Consequently, model reduction methods are needed to help avoid the the reachable state explosion problem. The authors also present an application to a complex FMS that was originally modeled with 92 places, 59 transitions, and 174 arcs. Their method was used to find 12 transitions and 20 places of the reduced Petri net, which allowed them to analyze the original system. The results of the reduction suggested five subsystems for the flexible manufacturing system.

BIBLIOGRAPHY

[1] Petri, C.A., "Kommunikation mit Automaten," Ph.D. dissertation, University of Bonn, Bonn, West Germany, 1962.

[2] Peterson, J.L., Petri Net Theory and the Modeling of Systems, Prentice Hall, Englewood Cliffs, N.J., 1981.

[3] Kamath, M., and Viswanadham, N., "Applications of Petri Net Based Models in the Modeling and Analysis of Flexible Manufacturing Systems," *Proceedings of the 1986 International Conference on Robotics and Automation*, IEEE Computer Society Press 1, 312-317, New York, 1986.

[4] Agerwala, T., "Some Applications of Petri Nets," *Proceedings of the National Electronics Conference XXXII*, Tranter, W.H., Ed, National Electronic Consortium, Inc, Vol 32, 149-154, Chicago, Ill, 1978.

[5] Al-Jaar, R.Y., and Desrochers, A.A., "Petri Nets in Automation and Manufacturing," in Advances in Automation and Robotics(ed. G.N. Saridis), JAI press, Greenwich, Conn., Vol 2, 1989.

[6] Al-Jaar, R.Y., and Desrochers, A.A., "A Survey of Petri Nets in Flexible Manufacturing Systems," *Proceedings of the 1988 IMACS Conference*, Paris, France, July 1988.

[7] Barad, M., and Sipper, D., "Flexibility in Manufacturing Systems:Definitions and Petri Net Modeling," *Int, J. Production Res.* Vol 26(5), 237-248,1988.

[8] Beck, C., and Krogh, B., "Models for Simulation and Discrete Control of Manufacturing Systems," *Proc. IEEE Int. Conf. Robotics Automat*, 305-310, 1986.

[9] Kodate, H., Fujii, F., and Yamanoi, K., "Representation of FMS with Petri Net Graph and its Application to Simulation of System Operation," *Robotics Computer Integrated Manufact.*, Vol3(3), 275-283, 1987.

[10] Ramchandani, C., "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets," Technical Report No.120, Laboratory for Computer Science, M.I.T., Cambridge, Mass, 1974.

[11] Sifakis, J., "Performance Evaluation of Systems using Nets," *Net Theory and Applications* (Lecture Notes in Computer Science), Springer-Verlag, Berlin, Germany, 307-319, 1978.

[12] Ramamoothy, C.V., and Ho, G.S., "Performance Evaluation of Asynchronous Concurrent Systems using Petri Nets," *IEEE Transactions on Software Engineering*, SE-6(5), 440-449, 1980.

[13] Cohen, G., Moller, P., Quadrat, J.P., and Viot, M., "Linear System Theory for Discrete Event Systems," *Proceedings of 23rd IEEE Conference on Decision and Control*, Vol 1, 539-544, 1984.

[14] Merlin, P., "A Methodology for the Design and Implementation of Communication Protocols," *IEEE Trans. Commun.*, 614-621,1976.

[15] Menasche, M., "Parede:An Automated Tool for the Analysis of Time(d) Petri Nets," *Proceeding IEEE Int. Workshop Timed Petri Nets*, Torino, Italy, 162-169, July 1985.

[16] Zuberek, W.M., "Timed Petri Nets and Preliminary Performance Evaluation," *Computer Architecture News*, Vol 8(3), 88-96, 1980.

[17] Razouk, R.R., and Phelps, C.V., "Performance Analysis using Timed Petri Nets," Protocol Specification, Testing, and Verification, eds. Y. Yemini, R. Storm and S. Yemini, 561-576. North-holland: Elsevier Science Publishers, 1982.

[18] Molly, P. M., "Performance Analysis using Stochastic Petri nets," *IEEE Transactions on Computers*, C-31(9), 913-917, 1982.

[19] Marson, M. A., Balbo, G., and Conte, G., "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor System," *ACM Transactions On Command Systems*, Vol 2(2), 93-122, 1984.

[20] Huber, P., Jensen, A.M., Jepsen, L. O., and Jensen, K., "Reachability Tree for High-Level Petri Nets," *Theoretical Computer Science*, 45, 1986.

[21] Jensen, K., "How to Find Invariants for Coloured Petri Nets," *Lecture Notes in Computer Science*, Vol 118, Springer-Verlag, 327-338, 1981.

[22] Jensen, K., "Colored Petri Nets," *Net Theory and Applications* (Lecture Notes in Computer Science), No.254, Springer-Verlag, Bad Honnef, Germany, 248-299, 1987.

[23] Jensen, K., "Colored Petri Nets and the Invariants-Method," *Theoretical Computer Science* 14, 317-336, 1981.

[24] Viswanadham, N., and Narahari, Y., "Colored Petri Net Models for Automated Manufacturing Systems," *Proceedings of the 1987 IEEE Internation Conference on Robotics and Automation, Raleigh, N.C.*, 1985-1990, April 1987.

[25] Alla, H., and Ladet, P., "Colored Petri Nets:A Tool for Modeling, Validation, and Simulation of FMS," *in Flexible Manufacturing Systems: Methods and Studies*, 271-281, Elsevier, New York, 1986.

[26] Martinez, J., Muro, P., and Silva, M., "Modeling, Validation and Software Implementation of Production Systems Using High Level Petri Nets," *Proceedings of the 1987 IEEE International Conference on Robotics and Automation Raleigh, N.C.*, 1180-1185, April 1987.

[27] Gentina, J.C., and Corbeel, D., "Colored Adaptive Structured Petri Net: A Tool for the Automatic Synthesis of Hierarchical Control of Flexible Manufacturing Systems," *Proceedings of the 1987 IEEE International Conference on Robotics and Automation Raleigh, N.C.*, 1166-1173, April 1987.

[28] Choi, B.W., and Kuo, Way., "Petri Net Extensions for Modeling and Validating Manufacturing Systems," Int. J. Prod. Res., (accepted for the publication)

[29] Beyaert, B., Florin, G., Lonc, P., and Natkin, S., "Evaluation of Computer System Dependability using Stochastic Petri Nets," *Proceeding of the 11th International Symposium on Fault Tolerant Computing(FTCS-11)*, Portland, 24-26, June 1981.

[30] Shapiro, S.D., "A Stochastic Petri Net with Application to modeling Occupancy Times for Concurrent Task Systems," *Networks*, Vol 9, 1979.

[31] Symons, F.J.W., "The Description and Definition of Queuing Systems by Numerical Petri Nets," *Australian Telecommunication Research*, 13, 20-31, 1980.

[32] Dugan, J.B., Trivedi, K.S., Geist, R.M., and Nocola, V.F., "Extended Stochastic Petri Nets," *Applications and Analysis, Performance 84*, Paris, France, 507-519, December 1984.

[33] Meyer, J.F., Movaghar, A., and Sanders, W.H., "Stochastic Activity Networks; Structure, behavior, and Applications," *International Workshop on Timed Petri Nets*, Torino, Italy, 106-115, July 1985.

[34] Ajmone Marsan, M. et al., "Generalized Stochastic Petri Nets", *Microelectron Reliability*, Vol 31(4), 698-723, 1991.

[35] Molloy, M., "Performance Analysis using Stochastic Petri Nets," *IEEE Trans. Comput.*, Vol C-31(9), 913-917, Sept 1982.

[36] Molloy, M., "Discrete Time Stochastic Petri Nets," *IEEE Trans. Software Eng.*, Vol SE-4(4), 417-423, Apr 1985.

[37] Florin, G., Fraize, C., Natkin, S., "Stochastic Petri Nets: Applications and Tools," *Microelectron Reliability*, Vol 31(4), 669-697, 1991.

[38] Murata, T., Komoda, N., and Matsumoto, K., "A Petri Net Based Factory Automation Controller for Flexible and Maintainable Control Specifications," *Proceedings of Conference on Industrial Electronics, Control and Instrumentation, IEEE Press*, Vol 1, 362-366, New York, 1984.

[39] Murata, T., Komoda, N., and Matsumoto, K., "A Petri Net-Based Controller for Flexible and Maintainable Sequence Control and its Applications in Factory Automation," *IEEE Transactions on Industrial Electronics*, Vol IE-33(1), 1-8, Feb 1986.

[40] Valett et al., "Putting Petri Nets to Work for Controlling Flexible Manufacturing Systems," *Proceedings of ISCAS'85*, 929-932, June 1985.

[41] Kent, E.W., "The Brains of Men and Machines," McGraw-Hill, New York, 1981.

[42] Zargham, M.R., and Tyman, M., "Neural Petri Nets" (Unpublished manuscript)

[43] Reisig, W., Petri Nets: an Introduction, Springer Verlag, Berlin, West Germany, 1985.

[44] Hack, M.H.T., "Analysis of Production Schemata by Petri Nets," *Technical Report 94 Project MAC*, Massachusetts Institute of Technology, Cambridge, Massachusetts, 110, 1972.

[45] Lyons, D., and Arbib, M., "A Formal Model of Computation for Sensory-Based Robotics," *IEEE Trans. Robotics Automat.*, Vol 5(3), 280-293, June 1989.

[46] Genrich, H., and Lautenbach, K., "System Modeling with High Level Petri Nets," *Theoretical Comput. Sci.*, Vol 13, 109-136, 1981.

[47] Alla, H., Ladet, P., Martinez, J., and Silva, M., "Modeling and Validation of Complex Systems by Colored Petri Nets: Application to a FMS," *Lecture Notes in Computer Science No. 188: Advances in Petri Nets, Proc. 5th European Workshop on the Application and Theory of Petri Nets*, Aarhus, Denmark, New York, Springer-Verlag, 15-31, 1984.

[48] Corbeel, D., Gentina, J. C., and Vercauter, C., "Application of an Extension of Petri Nets to Modernization of Control and Production Processes," *Proc. 6th European Workshop Application and Theory of Petri Nets*, Espoo, Finland, June 1985.

[49] Gentina, J., and Corbeel, D., "Coloured Adaptive Structured Petri Net: A Tool for the Automatic Synthesis of Hierarchical Control of Flexible Manufacturing Systems," *Proc. IEEE Int. Conf. Robotics Automat.*, 1987.

[50] Kasturia, E., Dicesare, F., and desrochers, A., "Real time Control of Multi-level Manufacturing Systems using Colored Petri Nets," *Proc. IEEE Int. Conf. Robotics Automat.*, 1988.

[51] Houldsworth, P.A., and Brearly, D., "Programmable Controller Functions are Enhanced by Structured Programming and Graphic Sequence Control Together with Good PC Documentation," *Proceedings of the Conference on Programmable Controllers*, Peter Lawrenson Editor, GAMBICA Programmable Controllers Committee, London England, 129-134, 1985.

[52] Brand, K.P., and Kopainsky, J., "Principles and Engineering of Process Control with Petri Nets," *IEEE Transactions on Automatic Control*, AC33(2), 138-149, 1988.

[53] Bruno, G., and Marchetto, G., "Process-translatable Petri Nets for the Rapid Prototyping of Process Control Systems," *IEEE Transactions on Software Engineering*, SE12(2), 346-357, 1986.

[54] Devanathan, R., Kuan, F.Y., Chang, C.J., and Choo, S.A., "Computer Aided Designing of Relay Ladder Logic via State Transition Digram," *IEEE International Conference on Industrial Electronics, Control and Instrumentation*, New York, Vol 2, 764-772, 1987.

[55] Krogh, B.H., and Beck, C.I., "Synthesis of Place/Transition Nets for Simulation and Control of Manufacturing Systems," *IFAC/IFORS Fourth Symposium on Large Scale Systems: Theory and Applications*, Zurich Switzerland, Vol 2, 583-589, August 1986.

[56] Krogh, B.H., Willson, R., and Pathak, D., "Automatic Generation of Control Programs for Discrete Manufacturing Processes," *The Robotics Institute Annual Research Review*, Carnegie Mellon University, 21-31, 1987.

[57] Lloyd, M., "GRAFCET - Graphical Function Chart Program," *Proceedings of the Conference on Programmable Controllers* Peter Lawrenson Editor, GAMBICA Programmable Controllers Committee, London England, 51-56, 1985.

[58] Martinez, J., and Silva, M., "A Package for Computer Design of Concurrent Logic Systems," *Proceedings of the Third IFAC/IFIP Symposium on Software for Computer Control*, Pergamon Press, Oxford England, 243-248, 1982.

[59] Martinez, J., and Silva, M., "A Language for the Description of Concurrent Systems Modelled by Coloured Petri Nets: Applications to the Control of Flexible Manufacturing Systems," *Language for Automation*, Plenum Press, New York, Chapter 8, 369-388, 1985.

[60] Martinez, J., Alla, H., and Silva, M., "Petri Nets for the Specification of Flexible Manufacturing Systems," *Modeling and Design of Flexible Manufacturing Systems*, Elsevior Science Publishers, Amsterdam, Chapter 8, 389-406, 1986.

[61] Matsuzaki, K., Hata, S., Junichi, H., Kurashima, Y., and Torii, M., "Petri-Net Structured Sequence-Control Language with GRAFCET-like Graphical Expression for Programmable Controller," *Proceeding of International Conference on Industrial Electronics*(1), New York, 1985.

[62] Chocron, D., and Cerny, E.A., "Petri Net Based Industrial Sequencer," *IECI Proceedings of Conference on Applications of Mini and Microcomputers*, IEEE Press, New York, Vol 1, 18-22, 1980.

[63] Courvoisier, M., Valette, R., Bigou, JM., and Esteban, P., "A Petri Net Based Programmable Logic Controller," *Proceedings of the IFIP Conference on Computer Applications in Production and Engineering* North-Holland, New York, 103-116, 1983.

[64] Valette, R., Courvoisier, M., Bigou, J.M., and Albukerque, J.A., "Petri Net based Programmable Logic Controller," *Computer Applications in Production and Engineering (CAPE 83)*, North-Holland Publishing, Amsterdam, The Netherlands, 103-116, 12983.

[65] Komoda, N., Murata, T., and Matsumoto, K., "Petri-Net Based Controller: Scr and its Applications in Factory Automation," *IEEE International Symposium on Circuits and Systems*, New York, Vol 2, 937-940, 1985.

[66] Crocket, D.H., Desrochers, A.A., DiCesare,F., and Ward, T., "Implementation of a Petri Net Controller for a Machining Workstation," *Proceedings of a Conference on Robotics and Automation 3*, IEE Society Press, New York, 1861-1867, 1987.

[67] Chiola, G., "A Graphical Petri Net Tool for Performance Analysis," *Proceedings of the 3rd International Workshop on Modeling Techiniques and Performance Evaluation*, AFCET, Paris, France, March 1987.

[68] Dugan, J. B., Bobbio, A., Ciardo, G., and Triverdi, K., "The Design of a Unified Package for the Solution of Stochastic Petri Net Model," *Proceedings of IEEE International Workshop on Timed Petri Nets*, Torino, Italy, 6-13, July 1985.

[69] Ciardo, G., "Manual for the SPNP Package," Duck University, Durham, N.C., July 1988.

[70] Holliday, M.A., and Vernon, M.K., "Generalized Timed Petri Net Model for Performance Analysis," *Proceedings of the IEEE International Workshop on Timed Petri Nets*, Torino, Italy, 181-190, July 1985.

[71] Murata, T., and Koh, J. Y., "Reduction and Expansion of Live and Safe Morked-Graphs," *IEEE Transactions on Circuits and Systems*, Vol CAS-27(1), 68-70, Jan 1980.

[72] Johnsonbaugh, R., and Murata, T., "Additional Method for Reduction and Expansion of Marked Graphs," *IEEE Transactions on Circuits and Systems*, Vol CAS-28(10), 1009-1014, Oct 1981.

[73] Suzuki, I., and Murata, T., "A Method for Stepwise Refinement and Abstraction Of Petri Nets," *Journal of Computer and System Sciences*, Vol 27, 51-76, 1983.

[74] Lee, K. H., and Favrel, J., "Hierarchical Reduction Method for Analysis and Decomposion of Petri Nets," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol SMC-15(2), 272-280, 1985.

[75] Lee, K. H., Favrel, J., and Baptiste, P., "Generalized Petri Net Reduction Method," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol SMC-17(2), 297-303, March/April 1987.

PAPER II.

PETRI NET EXTENSIONS FOR MODELING AND VALIDATING MANUFACTURING SYSTEMS

ABSTRACT

In this paper, we begin with the fundamental constructs of Petri net models. We then suggest extensions that help make Petri nets useful for modeling manufacturing systems. We also show how validation methods can be used to examine these systems for potential problems. Examples are presented to show how one might use this approach to determine the performance and validate the logic of a manufacturing system.

Keywords:

Extended Petri Net Models, Validation Method, Performance Analysis

INTRODUCTION

A manufacturing system can be viewed as a large set of different entities interacting in a complex manner. When we observe such a system, these entities exhibit both deterministic and stochastic behaviors. The complex nature of this behavior makes it difficult to model and evaluate, because one must consider factors such as capacity, resources, machine failure rate, and repair rate, in order to accurately determine various performance measures such as production rate or work in process.

A variety of approaches exist for modeling manufacturing systems, such as those proposed by Cohen [1], Nevins and Whitney [2], Hanssmann [3], and Malone and Smith [4]. While these models provided insight into system behavior, they introduced many restrictive assumptions and tended to be computationally complex, making it difficult to model and evaluate manufacturing systems in dynamic situations.

Petri nets have been used successfully to model, control, and analyze discrete event dynamic systems that are characterized by concurrency or parallelism, asynchronous processes, deadlocks, conflicts, and event-driven processes. Petri nets also provide accurate models and efficient analysis methods because they (1) capture interactions of concurrent and sequential events, (2) can be derived from the knowledge of how systems work, (3) give concise models for conflicts and buffer sizes, and (4) allow implementation of real time analysis [5,6].

Theories and applications of Petri nets have been studied by Peterson [6], Jensen [7-8], Viswanadham and Narahari [9], Murata [10], Memmi and Roucairol [11], Genrich and Lautenbach [12], and Chretienne and Carlier [13]. These studies provided only partial representations of a manufacturing system, leaving out such characteristics as multiple products, capacity, resource availability, failure rate, and priority.

A distinctive advantage of Petri nets is the ability to test and validate the model. This testing and validation process includes both determining if the model performs correctly (e.g., no deadlocks or boundedness) as well as determining if the Petri net accurately models the actual system.

A number of methods have been proposed for the validation of Petri net models. For instance, Peterson [8] introduced the reachability tree and matrix equation methods; Jensen [7-8] describes the invariant-method; and Memmi [11] introduces the algebraic meaning of the invariants. However, they considered relatively simple models. Therefore, we need to examine these methods to see if they are applicable to actual manufacturing systems.

Performance analysis of manufacturing systems provides a means of determining system characteristics. Performance analysis of Petri net models has been studied by Magott [14], Choi and Kuo [15], Hillion [16], Ramchandani [17], Sifakis [18], and Arbel and Seidmann [19]. These studies show how measures such as the maximum computation rate (minimum cycle time) and dynamic response time (including firing schedules) can be determined.

Using previous studies as a starting point, we extend Petri net models by adding elements for time, resource availability (number of resources, types of resources), type of processes, multiple products (with and without priority), capacity (buffer size or storage capacity limit), and failure rate (part defect or equipment breakdown). We then show how these models(with extensions) can be applied to manufacturing systems. A validation procedure is presented along with examples. In addition, results of a performance analysis using a Petri net model of a representative manufacturing system are used to evaluate the system in order to identify areas for improvement.

MODELING METHODOLOGY

Ordinary Petri Nets

A Petri net graph uses circles to represent places (states) and bars to represent transitions (events). Input-output relationships are represented by directed arcs between places and transitions. Tokens reside at a place when it is active. Tokens flow through the net depending on the present marking of the net. The marking of a Petri net is contained in a vector of dimension n, where n is the number of places and each value of the vector corresponds to the number of tokens in the corresponding place. When there is a token in each of the input places of a transition, that transition is enabled to fire. If the weights on each of the arcs between places and transitions are equal to one, then the transition fires by removing a token from each of its input places and by placing a token in each of its output places.

Figure 1(a) shows a Petri net example for accessing a robot. The tokens, places, and transitions correspond to the various elements found in manufacturing systems. Places usually represent resources (e.g., machines, parts, and data). A token in a place indicates that the resource is available; otherwise it is unavailable. A place can also be used to imply that a logical condition holds. Transitions are generally used to represent the initiation or termination of an event.

Processing Time

With a set of simple constructs, Petri nets can model a wide variety of discrete event dynamic systems. However, ordinary Petri nets do not account for the passage of time. In most systems, timing is a critical factor for evaluating performance and validating control logic. For manufacturing systems, this is especially true because time is an essential element in functions such as production scheduling and control.

Ramchandani [17] and Sifakis [18] introduced the notion of a Timed Petri net (TPN). Ramchandani described a TPN as a pair (PN,μ), where PN is a Petri net and μ is a vector of processing time functions that assigns a positive rational number to each transition of the net. In a TPN model, each transition t_i (after being enabled) has a time delay of $\mu(t_i)$ before firing. The firing times must be rational so that we can discretize the processing times in units of time and precisely describe the state of the process at each instant of time. The rule of operation of a TPN is similar to an ordinary PN. Once a transition is enabled, the tokens are removed from the input places and are held for time $\mu(t_i)$, after which the tokens are sent to all the output places. Transitions in TPNs can be viewed as a list of events in that multiple sets of tokens can be at different stages of the time delay. The execution of the model would be controlled through the use of a global clock to time events.

A transition associated with time $\mu(t_i)$ is graphically represented using a bar [], which indicates that a token stays in that transition for a processing time $\mu(t_i)$. Figure 1(b) shows a timed Petri net example using a robot. Transition t_1 has an associated delay time of A. When the conveyor and the robot are both available (i.e., a token is present in each place), the processing time for transition t_1 begins. The time delay A represents the material handling time for the conveyor to move a part to the robot.

Resources

Capacity In manufacturing systems, one finds a number of limited resources, such as machines and robots, that have the same process structure and behavior. Each resource has a fixed number of tokens representing the total capacity. The number of resource tokens in a place indicates the state of those resources (e.g., idle, down, or busy).

Capability Resources are differentiated by their set of capabilities to perform required functions. For example, a milling machine can be used to perform a family of metal removal processes. A Petri net extension that is useful for modeling different types of resources is called Colored Petri nets [7-8,12]. Using a similar approach, we represent token color as a token shape, indicating the identity of a resource. A place can have one or more token shapes that represent a set of different resources. These sets can be used to model a system with n different resources that provide different capabilities. A transition can fire with respect to each of these shapes. Transition firing follow rules of ordinary Petri nets except that token shapes must match.

In other manufacturing environments, there is a many-to-many mapping between product type and resource. This mapping is a direct result of capability requirements of products and available capabilities of resources. If different products (representing different token shapes) require one resource, then the transition cannot fire because the token shape is not matched. We then assign Greek letters to the tokens (α, β, γ, ... , χ, ψ, ω representing different capabilities). In this case, the transition firing rule will be changed as follows: (1) basically, a transition can fire with respect to the same token shape; (2)when a different token shape is matched, transition firings follow the rules of ordinary Petri nets except that the Greek letter must be checked. If the Greek letter is matched with the different token shapes, then the transition can be fired.

Failure rate Resource failure is a common stochastic element of behavior. We add a useful extension to Petri nets by allowing a failure rate (percentage of time that a given resource is down) to be assigned to each token. The resource failure rate **FR** specified by

$$\mathbf{FR} = \frac{\text{Repair time}}{\text{Operation time} + \text{Repair time}}$$

is calculated under the assumption that operation and repair time follow the exponential or weibull distribution. When a transition with a non-zero failure rate fires, a random number is generated with a probability **FR** of needing repair. This probability is used to determine which arc should receive a token. If a failure does occur, then the resource token moves along the failure arc to a place where the resource failure can be modeled. The resource token is not available until the resource has been repaired. If the resource fails due to the random variable after the transition fires in Figure 2, the resource token in place p_1 is moved to place p_r; and it takes a repair time R to make the token available in place p_1 again.

In this paper, we allowed that the model structure can be modified based on resource and system capability during repair time. For example, if one resource has problem, then other resources that have the same capability(representing by same token shape) can takes over its tasks. In this case, one resource can be shared for two processes and model structure may be changed dynamically corresponding to the resource and system capability.

Products Structure

Types of product A number of products, each having different process plans, must be considered. Again, token shapes (based on Colored Petri nets) are used to distinguish the products. Each unique shape has a corresponding process plan, and each process plan has a procedure associated with different resources.

Process plan In order to complete a process plan, a part uses a number of resources with different capabilities and potentially competes with the other parts for the same resources. The process plan for a part can be represented by a unique subnet. The processing times for each step are incorporated in the transitions as described earlier.

Priority Manufacturing systems can have a large number of multiple products, with demand varying from high-volume products (that are continuously produced) to low-volume products (that are produced intermittently). Conflicts arise when work orders compete for a single resource. Since a single resource cannot process these work orders simultaneously, there must be a set of rules to determine the order in which the work orders are processed. In situations like this, several questions are raised.

1. Which product should be produced first?
2. Which operation should take place first?
3. How many products should be produced according to the inventory and limited capacity?

In order to answer these questions, transitions and tokens are assigned priorities. If two or more transitions are enabled by one or more of the same places, we assign a priority to the transitions on the basis of which transition should fire first [6,10]. This is done by assigning a different priority number(1, 2, ... , n) to each transition, with 1 being the highest priority and n being the lowest priority. In Figure 3(a), two transitions (t_1,t_2) attempt to execute at the same time, but transition t_1 fires because it has a higher priority.

Priorities can also be assigned to token shapes as shown in Figure 3(b). The circle token has higher priority (priority 1) than the square token(priority 2), so the circle token always fires first. In addition, a higher priority also can be assigned to the token that has arrived earlier than the other token to use same resource in the particular queue of manufacturing systems.

Defective parts After processing, a product is either within specifications or outside of specifications (defective). We model this behavior as independent Bernoulli experiments with a probability p of being outside of specifications. A uniform random number between 0 and 1 is generated when the transition is fired to determine which edge should receive a token. If a failure does occur, the token (i.e., defective part) can be directed to a failure arc that leads to a place where rework is performed; or the token leaves the system. If both the resource and the part fail, then the procedure for resource failure (as described in Section 2.3.3) is performed simultaneously.

In Figure 4(a), we have two possible outcomes for the assembly process of a robot, namely, a failure or success, along with their probabilities p and $1-p$, respectively. If the defective part occurs after firing the transition (which takes process time T), the token is sent either to the place representing rework or it is disposed of, depending on the quality requirements of the product. With probability $1 - p$, the part enters the next process.

Storage

Buffers and storage areas are used throughout manufacturing systems. The size or capacity of these areas is an important consideration in the design of such systems. Capacity-designated Petri nets, as introduced in the literature [20,21], allow for the representation of limited storage space. For instance, a buffer could be represented by a capacity-designated Petri net. We use a modified-capacity Petri net, in which a capacity-designated place has a number representing storage capacity limit and another number representing inventory. The difference between these numbers is the space available for tokens. A capacity-designated place is graphically represented by using a large empty square, with the number inside the square indicating inventory (i.e., number of tokens) and the number outside the square indicating the storage capacity limit. Capacity-designated places will prevent input transitions from firing (i.e., blocking) if the inside number is equal to the outside number. Figure 4(b) shows a modified-capacity Petri net.

MODELING A MANUFACTURING SYSTEM

Product Structure and Resources

Consider a Flexible Manufacturing Cell(FMC) used to produce two different products (P^1 and P^2) shown in Figure 5. P^1 consists of three parts, let us call them P_1^1, P_2^1, and P_3^1. Three robots (R1, R2, and R3) and three conveyors (A, B, and C) are used to perform drilling and assembly tasks for P^1. These tasks are performed at three stations S1, S2, and S3. Conveyor A transfers P_1^1 to station S1 for drilling. Similarly, Conveyors B and C transfer P_2^1 and P_3^1 to stations S2 and S3, respectively. Once drilling operations are completed, P_1^1, and P_3^1 are sent to station S2, where robot R2 performs the final assembly for P_1^1, P_2^1, and P_3^1.

Product P^2 consists of two parts, P_1^2 and P_2^2. Robots R2 and R3 along with conveyors B and C are used to perform drilling and assembly tasks for P^2. These tasks are performed at stations S2 and S3. Conveyors B and C transfer P_1^2 and P_2^2 to stations S2 and S3, respectively, for drilling and assembly tasks. Once drilling operations for P_1^2 and P_2^2 are completed, P_2^2 is sent to station S2, where robot R2 assembles the part P_2^2 with P_1^2 to get the final product P^2. Upon completion of their respective assemblies, both product P^1 and P^2 are sent to a storage area.

The acquisition of a robot occurs when it is idle, and the release of a robot occurs when the assembly task is completed. We assume that the process times can be represented deterministically, which is not without precedent for automated systems [22,23]. Deterministic processing times are assigned to each process and are represented by transitions.

Modeling

Starting with ordinary Petri net constructs, we can formulate the following models to partially represent the system in Figure 6.

- Single interacting model 1

  To carry out the drilling operation for P_1^1 of product P^1, conveyor A and its left robot R1 are needed. The process occurs in three parts: t_0 (distribute parts for each robot), t_1 (acquire its robot), and t_2 (drill, release robot, and send completed part to conveyor B). The circle token in place p_1 indicates that robot R1 is available.

- Single interacting model 2

  This model uses conveyor C and robot R3 for P_3^1 of product P^1 and is similar to the previous model.

- Single interacting model 3

  P_2^1 uses conveyor B and robot R2 that carry out a drilling operation and then an assembly operation. This process occurs in four parts: t_0 (distribute parts for each robot), t_5 (acquire its robot and drill), t_6 (assemble with parts from conveyor A), and t_7 (assemble with parts from conveyor C, release robot, and send completed products to storage areas.) The circle token in place p_9 indicates that robot R2 is available. Figure 6 shows the integrated Petri net model that combines models 1, 2, and 3.

In order to complete the Petri net model of the flexible manufacturing cell, we add our extensions previously described in the following manner.

- Extension 1

In Figure 7, two different products (P^1 and P^2), each having different process plans, are modeled. Also, two different robots (represented by circle and square tokens) can be assigned in place p_5 and p_9 for different process plans. In addition, the number of these two robots can be controlled by increasing or decreasing the tokens in places p_1, p_5, and p_9. The drilling and assembly processes for P^1 and P^2 are as follows: the circle token passes through transitions t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7, and the square token passes through transitions t_0, t_3, t_4, t_5, t_6, t_7. Note that the square token does not use the input transition of p_2, and the output transition of p_4. In this situation, the circle and square token in place p_{13} have unique process plans, where their steps are the same but the times and resources are different.

- Extension 2

We introduce demand from the distribution system as well as stock and capacity limits. In Figure 7, the place p_0 represents the source of demands that are predicted on the basis of orders from customers. The tokens in place p_{13} represent empty fixtures used for each part. The places p_4 and p_8 represent buffer areas, and the place p_{13} represents initial capability of the overall assembly system. The number of tokens initially in p_{13} bounds the number of inputs (demands) that the FMC can process at the same time. Each time that a new input is processed, i.e., whenever t_0 fires, a token is removed from p_{13}. The tokens return to p_{13} once the processing is completed, i.e., when t_7 fires.

- Extension 3

Priority and failure are introduced in the model in the following way. Robots R1, R2, and R3 are given failure rates. If two robots fail using exponential distributions, then the system goes down. However, if only a single robot fails, then the remaining robots can perform only their tasks and the robot that has a problem stops its tasks. For example, in Figure 7 if robot R1 has mechanical problems modeled by the random variable after the firing of transitions t_1 or t_2, then the resource token in place p_1 moves to place p_r and the token delays a deterministic repair time in transition t_r before returning to place p_1.

An assigned priority for tokens determines which token can fire first. For instance, if square and circle tokens in p_5, p_6, p_9, and p_{10} are available at the same time, the circle token fires first because the circle token has a higher priority.

In addition, we changed the Petri net model structure of FMC to show a conflict case based on the resource capability in p_9 in the following ways: add arcs from p_9 to t_1 and t_2 to p_9 instead of input-output arcs of place p_1 as shown in Figure 8. Then robot R2 takes over assembly tasks for t_1 because robot R2 has the same capability as robot R1. In this case, robot R2 must serve two assembly processes(t_1 and t_5) and priorities for transitions t_1 and t_5 are assigned. If a conflict arises between these transitions, then the priorities for t_1 and t_5 are determined by which transition fires first.

VALIDATION OF PETRI NET MODEL

After modeling the elements of the system with Petri nets, we consider several properties of the Petri net model for validation, namely, reachability, liveness, boundedness, and conservativeness [6]. This analysis can lead to a better understanding of the system's behavior. Two main analysis methods - the reachability tree method and the invariant method [6,7] - can be used to analyze the properties of Petri nets.

Invariant Method

To show the invariant method for a Petri net model [6,11] of the FMC, we consider the conservation problem - showing that tokens are neither created or destroyed. Stated another way, a weighted sum of the number of tokens in each place at any instant in time should be constant(i.e., invariant). The weighted sum of tokens may be viewed as invariants about the behaviour of the FMC and can be determined using matrix equations. We have two kinds of invariants [11], the *p*-invariant (associated with places and represent unchanging sets of conditions of Petri net models) and the *t*-invariant (associated with transitions and represented transitions that leave a weighted sum of the number of tokens in each place of the Petri net model unchanged after fired). Therefore, *t*-invariants will be considered for validating Petri net model in this paper.

We can also represent a Petri net by using a matrix with rows and columns representing transitions and places, respectively. Input transitions to place j are indicated by -1 in the corresponding row and column j. In a similar manner, $+1$ indicates an output transition. Let us define **A** ($=a_{ij}^{+}$ - a_{ij}^{-}) as an $m \times n$ incident matrix, where a_{ij}, i = 1, 2, ..., m(transitions), j = 1, 2, ..., n(places) signifies a directed arc, a_{ij}^{+} = number of arcs from the transition j to output place i, and a_{ij}^{-} = number of arcs from input place i to transition j. Let W be the n x 1 column vector of weights for each place in the Petri net model.

The t-invariants can be obtained as the integer solutions to the following equation (based on the definition in which vectors W are called t-invariants iff $\mathbf{A} \cdot W = 0$) [6,11]:

$$\mathbf{A} \cdot W = 0$$

The set of linear equations for the FMC corresponding to the matrix equation are as follows:

$$\begin{aligned}
-W_1 + W_3 + W_7 + W_{11} - W_{14} &= 0 \\
-W_2 - W_3 + W_4 &= 0 \\
W_2 - W_4 + W_5 &= 0 \\
-W_6 - W_7 + W_8 &= 0 \\
W_6 - W_8 + W_9 &= 0 \\
-W_{10} - W_{11} + W_{12} &= 0 \\
-W_5 - W_{12} + W_{13} &= 0 \\
-W_9 + W_{10} - W_{13} + W_{14} + W_{15} &= 0
\end{aligned}$$

To find t-invariants that characterize all possible firing behaviors and validate the Petri net model of FMC, we obtain the set of positive t-invariants that are vectors (W_i). Since there are 15 unknowns and only 8 equations, there are multiple solutions. We introduce a simple heuristic method to obtain one solution for W_i because one solution that is a minimum integer is sufficient to get the t-invariants. We begin by choosing a solution to the first equation. This is done by assigning a value for the each W_i in the equation such that there is a balance between the number of input transitions and number of output transitions. For example, in the first equation, W_3, W_7, and W_{11} represent output transitions (because they have a positive value). Thus, we assign each of these transitions a value of 1. Then we solve the remaining transitions (W_1 and W_{14}) by assigning a value to each such that the total equation sums to 0. We arbitrarily assign one a value of 1 and the other a value of 2. The second step in the heuristic is to substitute the values from the first equation into the second and choose values for any remaining unknowns in the second equation. We continue this process until all equations have been solved. The result represents one possible solution to the set of equations. Using the heuristic, we find a vector W_i for which the weighted sum over all reachable markings from initial markings of the Petri net model are constant. These results are useful for validating some important properties of Petri net models such as conservativeness, boundedness, liveness, and properness.

Reachability Tree Method

A reachability tree can be used to determine if all states are reachable and the existence of deadlocks. The reachability tree is a directed graph that shows all possible sequences of transition firings. Each firing results in a unique marking of the Petri net. A deadlock is indicated by a marking that has no possible transitions. The Petri nets model in Figure 7 is validated by using a reachability tree from the arbitrary initial marking M_0 =(1,0,0,3,2,0,0,4,2,0,0,0,5). First, the model must be shown to be deadlock-free by determining if all the transitions can be fired along with the FMC operations. In Figure 9, a reachability tree that has 12 branches can be generated from the Petri net model of FMC. Transitions t_1, t_3, and t_5 can be fired at the same time and transitions t_2, and t_4 can be followed sequentially. But FMC operations depend upon the firing of transition t_0, t_5, t_6, and t_7. After all transitions are fired in any sequence of branches, the marking is returned to the initial marking M_0. Figure 9 also shows that the sequence for these transitions can be changed when several transitions are enabled at the same time. Finally, since the Petri net model is deadlock free, the reachability tree shows that all places have one, two, or no tokens with all possible sequences except places p_4 and p_8, which have a deterministic number of tokens representing storage capacity. This indicates that the Petri nets model is bounded.

From these results, the formal properties of the Petri nets model (boundedness, conservativeness, liveness, and deadlock free) can be easily analyzed and validated.

PERFORMANCE ANALYSIS

A simple Petri net model of the logical and causal dependencies in a manufacturing system is not a sufficient tool to analyze system characteristics such as time, resource availability, capacity constraints, and demand.

Two types of constraints affect the performance of a manufacturing system. The first type is the internal structure that defines how the processes work in the system. Most manufacturing systems have both sequential and concurrent processing activities. The second type of constraint is magnitude of time, resources, capacity, priority, and demand. The system has a limited amount of productive time because of the constrained resources. In addition, resource availability and capacity limits are used to represent the productive capacity of the system. Priority is used to resolve conflict situations that arise when work orders compete for a single resource. Customer demand will be given in place P_0 in our model.

Maximum Production Rate of the System

If demand is low, the manufacturing system will satisfy it. In this case, the rate at which products are being processed will precisely correspond to the demand rate. However, beyond a certain demand rate, products will compete for resources, creating a backlog of work. This bound determines the maximum production rate of the system and is a function of time, resources, capacity, and priority.

The Production Schedule

The production schedule specifies the time at which work orders are released into the system. This type of schedule represents a push production system. Assume that the processing starts at time $t = 0$ and that a large demand forces processing to occur at the maximum production rate. In this section, we want to determine the schedule for the various processes (represented by transitions) for various demand rates. From the flow time (the time interval between the moment the demand was received and the moment a product was made), we can determine the tardiness of the work order.

By changing the control variables in the model (e.g., resources and time) and executing the Petri net model of FMC, we can determine execution schedules for each process and a production schedule for scheduling purposes. Therefore, the production schedule of the FMC will be changed on the basis of process schedules for each process and the characteristic dynamic behaviors of the FMC. Starting from the initial stage, the process can be continued repetitively until demand is satisfied. Thus, the best performance of the manufacturing system will be obtained with respect to system characteristics such as time, resource availability, capacity, and demand.

Numerical Results

Using our extended Petri net models, we have implemented a deterministic and stochastic algorithm to analyze the FMC system in Figures 6, 7, and 8. We create an input file (including transitions, places, input-output relations of transitions, tokens in each place, process time for each transition, capacity of storage place, priority, and failure rate) in order to model and analyze the assembly system. By changing different variables, we create output files with the following results:

- The maximum production rate of the FMC [17]

  From the incident matrix, we can determine all the circuits (possible flows of tokens), the maximum circuit time (longest circuit flow time), the throughput of the Petri net model[24].

- Availability(percentage of idle time) of resources in the FMC

  From the state variable (**A**) defined as follows:

$$A = \begin{cases} \mathit{0} & \text{if resource is not available} \\ \mathit{1} & \text{if resource is available} \end{cases}$$

  we know the availability and utilization for resources in place p_1, p_5, and p_9 of the Petri net model using state variables.

- Processes and production schedules

  Schedules for the various processes (represented by transitions) and production schedules for various demand rates can be evaluated.

- Statistical analysis

  Confidence intervals for maximum production rates are based on resource failure rates and deterministic repair times.

We investigated six cases in which we perturbed the FMC system to identify areas for improvement. The perturbations included processing time for each transition (case 1), resource availability (case 2), number of resources (case 3), storage capacity (case 4), priority and failure rate (case 5), and sensitivity analysis (case 6) as follows:

- Case 1

  We initially assume deterministic time ($t_0 = 3$ units, $t_1 = 2$ units, $t_2 = 3$ units, $t_3 = 4$ units, $t_4 = 5$ units, $t_5 = 3$ units, $t_6 = 5$ units, and $t_7 = 5$ units) and vary these times to increase the maximum production rate.

On the basis of the incidence matrix, six circuits (six possible flows of tokens) are found in the Petri net model [16,17]. Table 1a shows six circuits, with the longest one being the critical circuit that determines the maximum flowtime of the assembly system [16]. The critical circuit of this Petri net model is associated with processes between conveyor A (transitions t_0, t_1, t_2) and conveyor B (transitions t_6, t_7). However, in general, the maximum production rate is governed by the bottleneck. Because of the configuration of the FMC, the maximum production rate (=1/minimum flowtime) is used as a measure of system performance.

To increase the maximum production rate, we must reduce the processing time for transitions on the critical circuit. For example, if the process time for transition t_6 is reduced from five to four, then there are two critical circuits (Table 1b), which in turn increases the maximum production rate. In addition, if process time for transition t_7 is reduced from five to three, then there are two critical circuits (Table 1c), but the maximum production rate is increased.

- Case 2

  The processing times are fixed and the number of moving fixtures in place p_{13} are varied to determine the number of fixtures that maximize resource utilization.

Table 2a shows dynamic changes of the resource availability in the Petri nets model. Robot R1 is not working from time 8 to 18 although it is available. To increase robot utilization, we need to increase the number of fixtures in place p_{13}. The maximum production rate will also be increased. Table 2b shows dynamic changes of the resource utility in the Petri nets model with two moving fixtures in place p_{13}.

- Case 3

  In this case, the processing time (same as used in Case 2), and number of moving fixtures(2) are fixed, but the number of robots R1, R2, and R3 is varied. Table 3(a) shows that case 3(e) and case 3(g) have the largest maximum production rate.

- Case 4

  With fixed deterministic processing time and a number of moving fixtures (same as used in Case 2), and a fixed number of robots R1(= 1), R2(= 2), R3(= 2), but stock in place p_4, p_8 varied by five. In this case, the maximum production rate is not changed, but the production schedule will change on the basis of the stock.

- Case 5

  A failure rate (0.0125, 0.025, 0.0375, and 0.05) and a deterministic repair time(100 unit times) has been given to robot R1 on the basis of the exponential distribution with $\lambda = 0.0025$ and $\mu = 0.1975$, $\lambda = 0.005$ and $\mu = 0.195$, $\lambda = 0.0075$ and $\mu = 0.1925$, and $\lambda = 0.01$ and $\mu = 0.19$ per 1000 hours, respectively. If R1 fails, then the remaining robots continue to perform their tasks.

The confidence interval for the maximum production rate that can be obtained by using the stochastic algorithm was run 10,000 times on the basis of case 4 and different failure rates. Table 3(b) shows a confidence interval of 95 % for the maximum production rate for different failure rates.

If two robots fail, the system goes down. If only a single robot fails, then the remaining robots may be used to perform the failed robot's tasks (see extension 3 in Section 3.2). For example, in Figure 8 robot R1 has failed from a generated random variable, so robot R2 will perform R1's tasks. Transitions t_1 and t_5 have become enabled to fire simultaneously. However, transition t_1 has a higher priority so it fires first. In this case, the maximum production rate is 0.22 based on deterministic time (case 2), number of moving fixtures (= 2), and number of robots R1(= 1), R2(= 2), R3(= 2) are fixed.

- Case 6

We assume process time, number of moving fixtures, and number of robots can vary from configurations of the FMC as follows.

1. process time : t_0 = 2-3 units, t_1 = 2 units, t_2 =2-3 units, t_3 =3-4 units, t_4 =4-5 units, t_5 =2-3 units, t_6 = 4-5 units, and t_7 =4-5 units.
2. number of moving fixtures are varied by ten.
3. number of robots R1, R2, and R3 are varied by two.

Using the deterministic algorithm, we simulate an analysis according to the process time, the number of moving fixtures, and the number of robots (which are varied). Finally, we suggest the greatest maximum production rate that occurs given the process time, number of moving fixtures, and number of robots.

CONCLUSION

We have shown how ordinary Petri nets can be extended to model time, resource availability, multiple products, different processes, capacity, priority, and failure rate. These extensions allow a wide variety of manufacturing systems to be modeled. Validation methods in the context of these extensions can identify potential problems in system operations. Several examples have shown how the Petri net models with some extensions can be effective in modeling and analyzing manufacturing systems. The results of the performance analysis from a deterministic or stochastic model are used to reorganize and re-evaluate manufacturing systems so they may respond flexibly.

BIBLIOGRAPHY

[1] Cohen, H.L., 1988, Strategic Analysis of Integrated Production-Distribution Systems: Models and Methods. *Operations Research*, 36(2), 216-228.

[2] Nevins, J.L., and Whitney, D.E., 1989, *Concurrent Design of Products and Processes*(McGraw-Hill, New York.).

[3] Hanssmann, F., 1959, Optimal Inventory Location and Control in Production and Distribution Networks. *Operations Research*, 7, 187-193.

[4] Malone, T.W., and Smith, S.A., 1987, Modeling the Performance of Organization Structures. *Operations Research*, 36(3), 198-206.

[5] Al-Jaar, R.Y., and Desrochers, A.A., 1989, Petri Nets in Automation and Manufacturing. *in Advances in Automation and Robotics*, Vol 2, (JAI Press, Greenwich, Conn.).

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

[7] Jensen, K., 1981, Colored Petri Nets and the Invariants-Method. *Theoretical Computer Science*, 14, 317-336.

[8] Jensen, K., 1981, High-Level Petri Nets Applications and Theory of Petri Nets. *Informatica-Fachberichte* 66 (Springer-Verlag, Hamburg) 166-180.

[9] Viswanadham, and Narahari, Y., 1987, Coloured Petri Net Models for Automated Manufacturing Systems. *Proceedings of the IEEE International Conference on Robots and Automation*, Raleigh, N.C., 1985-1990.

[10] Murata, T., 1987, Petri Nets : Properties Analysis and Applications. *Proceedings of the IEEE*, 77(4), 541-580.

[11] Memmi, G., and Roucairol, G., 1979, Linear Algebra in Net Theory. *Net Theory and Applications* (Lecture Notes in Computer Science), 84 (Springer-Verlag, Hamburg, Germany) 213-223.

[12] Genrich, H.J., and Lautenbach, K., 1981, System Modeling with High-Level Petri Nets. *Theoretical Computer Science*, 13, 109-136.

[13] Chretienne, P., and Carlier, J., 1984, Modeling Scheduling Problems with Timed Petri Nets. *Advances in Petri nets* (Lecture Notes in Computer Science), 188 (Springer-Verlag, Berlin, Germany) 62-82.

[14] Magott, J., 1987-88, Performance Evaluation of Concurrent Systems using Conflict-Free and Persistent Petri Nets. *Information Processing Letters*, 26, 77-80.

[15] Choi, B.W., and Kuo, W., 1988, Performance Analysis of a Generic Naval C2 Battle Group System by Use of Timed Petri Nets. *Proceedings of the Winter Simulation Conference*, 765-774.

[16] Hillion, H.P., 1983, Performance Evaluation of Decision Marking Organization using Timed Petri Nets. *M.S. thesis*, (M.I.T., Cambridge, Mass.).

[17] Ramchandani, C., 1974, Analysis of Asynchronous Concurrent Systems by Timed Petri Nets. *Technical Report No.120, Laboratory for Computer Science*, (M.I.T., Cambridge, Mass.).

[18] Sifakis, J., 1978, Performance Evaluation of Systems using Nets. *Net Theory and Applications* (Lecture Notes in Computer Science), (Springer-Verlag, Berlin, Germany) 307-319.

[19] Arbel, A., and Seidmann, A., 1984, Performance Evaluation of Flexible Manufacturing Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 1(5), 606-617.

[20] Andre, C., Armand, P., and Boeri, F., 1979, Synchronic Relations and Applications in Parallel Computation. *Digital Processes*, 5, 99-113.

[21] Murata, T., and Komoda, N., 1987, Liveness Analysis of Sequence Control Specifications Described in Capacity-Designated Petri Nets using Reduction. *Proceeding of the IEEE International Conference on Robotics and Automation*, Raleigh, N.C., 1985-1990.

[22] Liu, C., 1988, Stochastic Design Optimization of Asynchronous Flexible Assembly System. *Annals of Operations Research*, 11, 131-154.

[23] Bulgak, A,A., 1992, Impact of Quality Improvement on Optimal Buffer Designs and Productivity in Automatic Assembly Systems. *Journal of Manufacturing Systems*, 11(2), 129-143.

[24] Alaiwan, H., and Toudic, J.M., 1985, Recherche des Semi-Flots, des verrous et des Trappes dan les Reseaux de Petri. *Technique et Science Informatiques*, 4(1), Dunod, France, 103-112.

(a)

CYCLE	CIRCUIT TIME (UNITS)	TRANSITION - PLACE SEQUENCES
1	16.0	T0 P10 T5 P11 T6 P12 T7 P13
2	17.0	T0 P6 T3 P7 T4 P8 T7 P13
3	18.0	T0 P2 T1 P3 T2 P4 T6 P12 T7 P13
4	5.0	T1 P3 T2 P1
5	9.0	T3 P7 T4 P5
6	13.0	T5 P11 T6 P12 T7 P9

(b)

CYCLE	CIRCUIT TIME (UNITS)	TRANSITION - PLACE SEQUENCES
1	15.0	T0 P10 T5 P11 T6 P12 T7 P13
2	17.0	T0 P6 T3 P7 T4 P8 T7 P13
3	17.0	T0 P2 T1 P3 T2 P4 T6 P12 T7 P13
4	5.0	T1 P3 T2 P1
5	9.0	T3 P7 T4 P5
6	12.0	T5 P11 T6 P12 T7 P9

(c)

CYCLE	CIRCUIT TIME (UNITS)	TRANSITION - PLACE SEQUENCES
1	13.0	T0 P10 T5 P11 T6 P12 T7 P13
2	15.0	T0 P6 T3 P7 T4 P8 T7 P13
3	15.0	T0 P2 T1 P3 T2 P4 T6 P12 T7 P13
4	5.0	T1 P3 T2 P1
5	9.0	T3 P7 T4 P5
6	10.0	T5 P11 T6 P12 T7 P9

Table 1: Critical circuit and maximum production time of the FMC.

(a)

TIME	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14
0.0	10	1	0	0	0	1	0	0	0	1	0	0	0	1	0
3.0	9	1	1	0	0	1	1	0	0	1	1	0	0	0	0
6.0	9	0	0	1	0	1	1	0	0	0	0	1	0	0	0
8.0	9	1	0	0	1	0	0	1	0	0	0	1	0	0	0
12.0	9	1	0	0	0	1	0	0	1	0	0	0	1	0	0
15.0	9	1	0	0	0	1	0	0	0	1	0	0	0	1	1
18.0	8	1	1	0	0	1	1	0	0	1	1	0	0	1	1

(b)

TIME	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14
0.0	10	1	0	0	0	1	0	0	0	1	0	0	0	2	0
3.0	9	1	1	0	0	1	1	0	0	1	1	0	0	1	0
6.0	8	0	1	1	0	1	1	0	0	0	1	1	0	0	0
8.0	8	1	1	0	1	0	1	1	0	0	1	1	0	0	0
12.0	8	0	0	1	0	1	1	1	1	0	1	0	1	0	0
15.0	8	1	0	0	1	0	0	1	0	1	1	0	0	1	1
18.0	7	1	1	0	1	0	1	0	0	0	1	0	0	1	1

Table 2: Dynamic changes of the Petri net model.

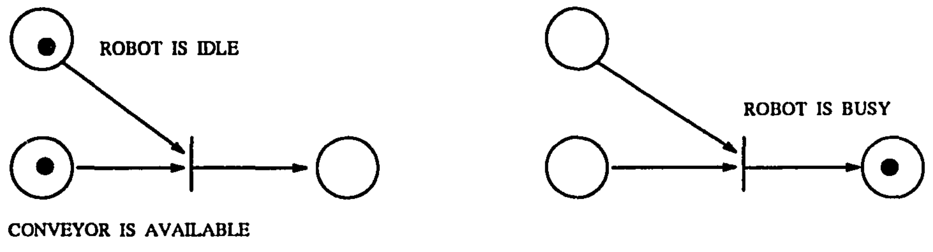
	ROBOT A	ROBOT B	ROBOT C	MAXIMUM PRODUCTION RATE
CASE 3(a)	2	1	1	0.1
CASE 3(b)	1	2	1	0.1
CASE 3(c)	1	1	2	0.11
CASE 3(d)	2	1	2	0.11
CASE 3(e)	1	2	2	0.2
CASE 3(f)	2	2	1	0.1
CASE 3(g)	2	2	2	0.2

(a) Maximum production rate from deterministic algorithm

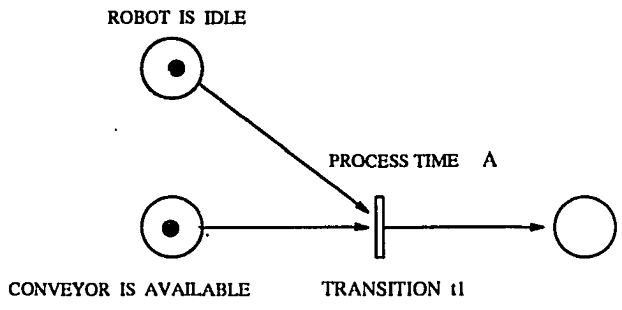
FAILURE RATE	CONFIDENCE INTERVAL FOR THE MAXIMUM PRODUCTION RATE
0.0125	0.195 - 0.205
0.025	0.193 - 0.207
0.0375	0.19 - 0.21
0.025	0.189 - 0.211

(b) Maximum production rate from stochastic algorithm

Table 3: Maximum production rate



(a) Petri net example



(b) Timed Petri net example

Figure 1: Ordinary and Timed Petri net example

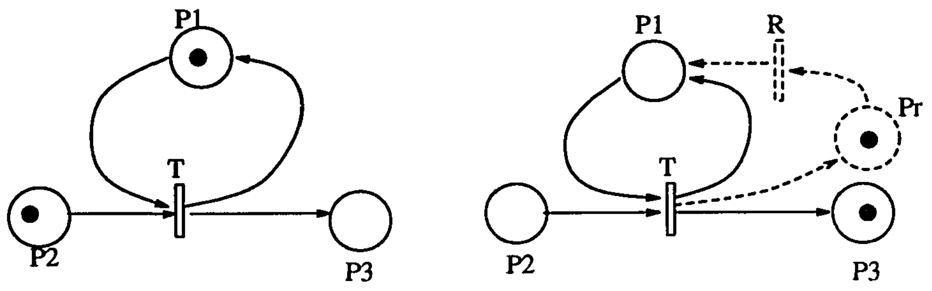
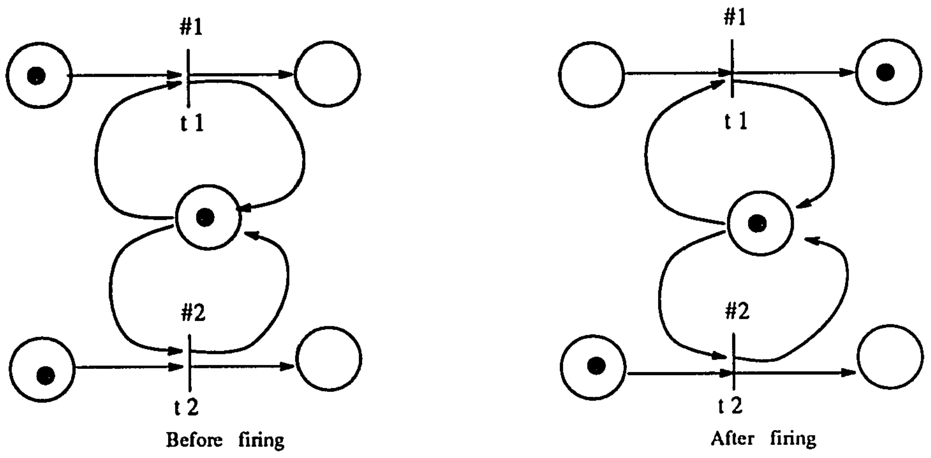
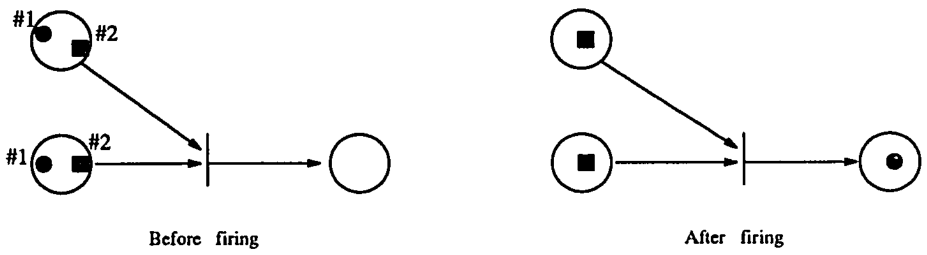


Figure 2: Modified Petri net example with failure rate

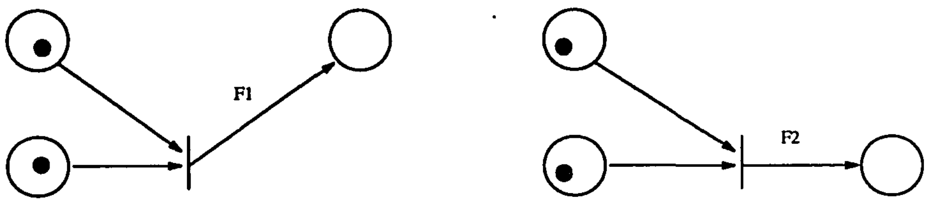


(a) Transition with priority

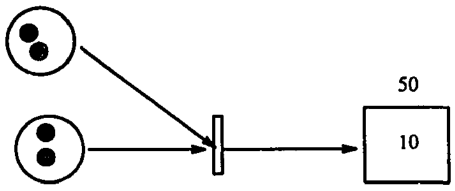


(b) Token with priority

Figure 3: Modified Petri net examples with priority



(a) Failure rate for the defective part



(b) Modified Petri net example with storage

Figure 4: Modified Petri net examples with product

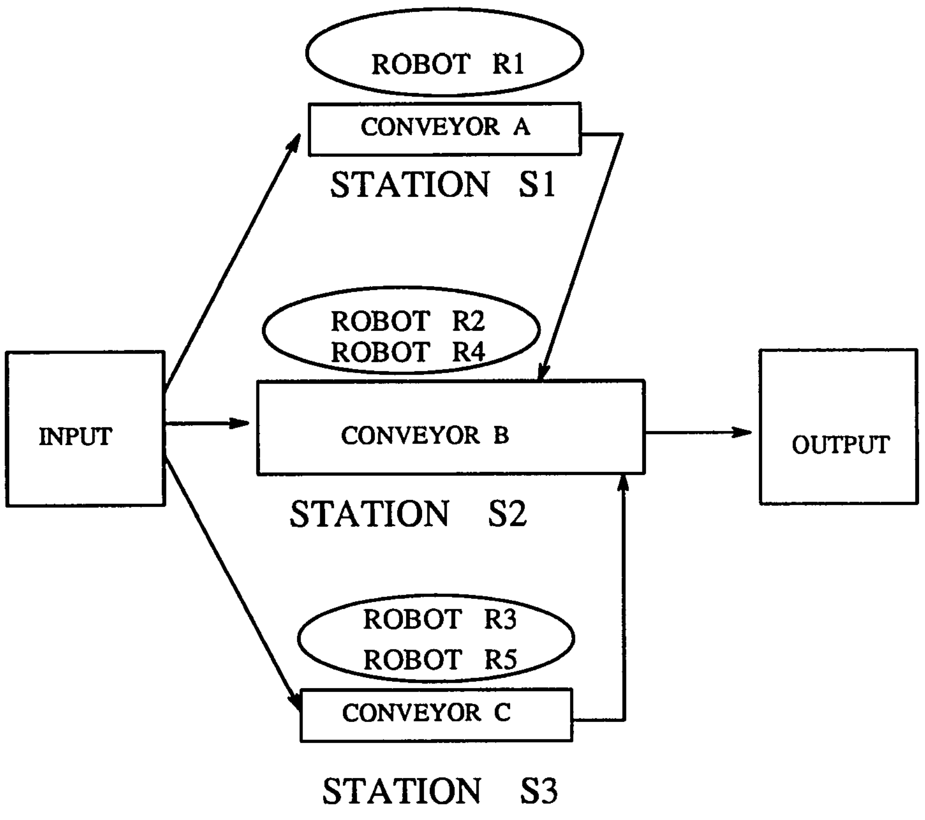


Figure 5: Flexible manufacturing cell

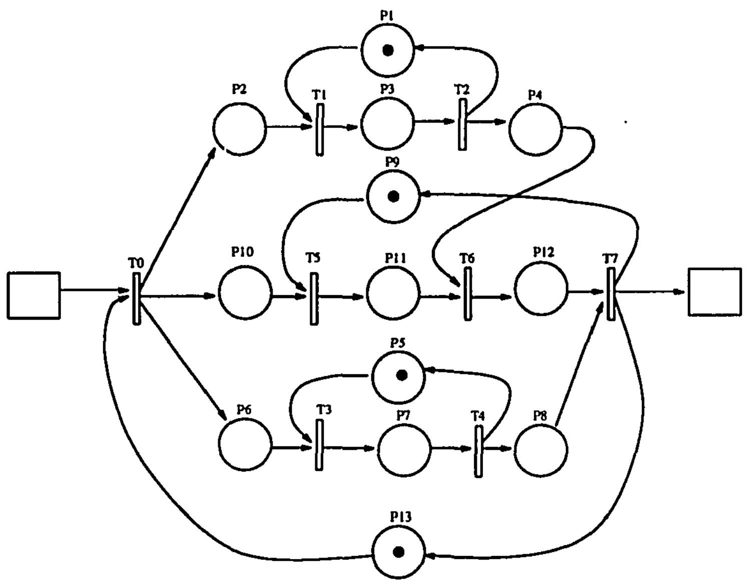


Figure 6: Petri net model of the FMC

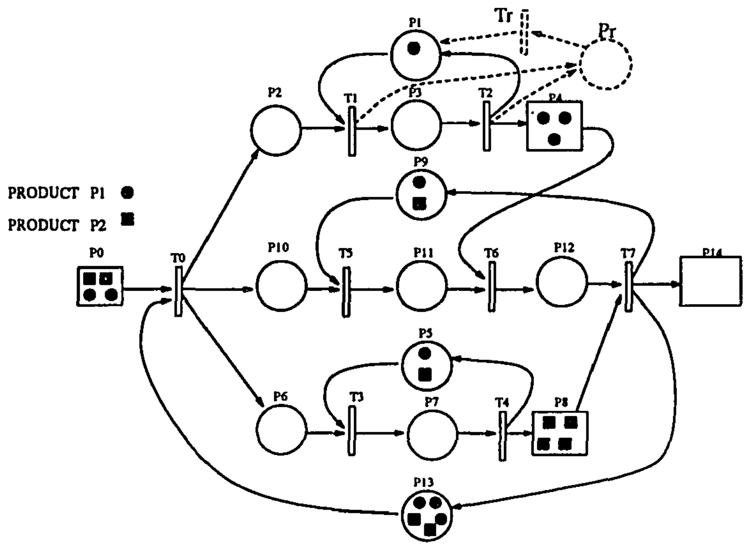


Figure 7: Extended Petri net model of the FMC

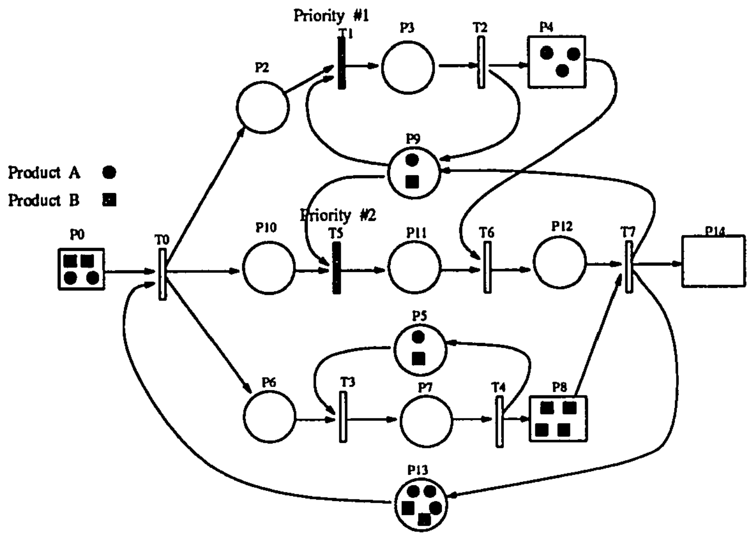


Figure 8: Extended Petri net model of the FMC with failure rate and priority

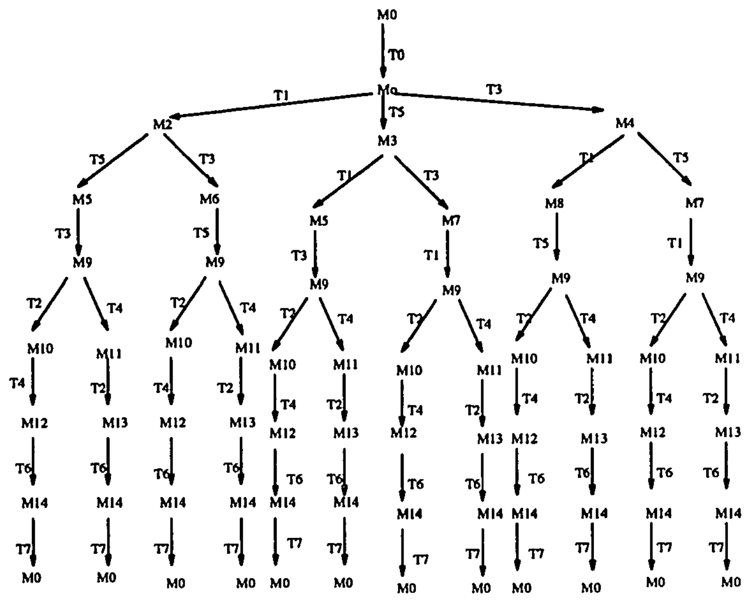


Figure 9: Reachability tree

PAPER III.

A PETRI NET APPROACH TO MODELING, ANALYZING, AND EVALUATING AN AUTOMATED PALLETIZED CONVEYOR SYSTEM

ABSTRACT

In this paper, we present an approach to modeling, analyzing and evaluating an Automated Palletized Conveyor System (APCS) using extended Petri net models. We first examine the APCS and extend the fundamental constructs of Petri net models. We then build a Petri net model of the given APCS, analyze important qualitative aspects of APCS behaviors and finally evaluate performances of APCS.

A modified deterministic and stochastic algorithm is developed to describe and evaluate the Petri net model of the given APCS. The input and control mechanisms of the Petri net model are varied, implemented, and evaluated to produce results that can be used to redesign the APCS and also can be directly applied to the design and analysis of the full-scale material handling operation.

INTRODUCTION

Material handling facilities are considered to be the key factor in modern manufacturing systems because they are widely used to transport and store material between production stages. Although many aspects of the material handling facilities have been considered as subjects of research, we still require methodologies and control mechanisms to experiment with the material handling facility in a readily controlled environment.

The APCS consists of two conveyors, pallets, three different parts, one robot for part placement, several pneumatic actuators, several sensors, sixteen cylinders, two conveyor circuits with four motors and one programmable logic controller. This system provides a real-time physical model for a modern integrated manufacturing facility.

More specifically, the APCS is a double closed loop system (feed loop and machining loop) controlled by a Programmable Logic Controller(PLC), programmed using relay ladder logic. The belts of the double conveyor move continuously at a constant speed. The feed loop shows how to control assembly materials represented by different shapes, and the machining loop controls the number of pallets and represents the machining processes. With the use of photoelectric sensors, features such as sorting, accumulation of dissimilar parts, and automated transfer between conveyor circuits provide simple solutions for material handling problems. By using this system, we can have several useful experimental situations to analyze and design a material handling system.

However, most PLC's that control the APCS, are based on Boolean languages or relay ladder logic. These are complex control diagrams that have not been used or defined in order to describe high level specification concepts. Also, the APCS has several changeable variables and components such as conveyor speed, sensor, different parts, number of parts, number of pallets and cylinder positions. Therefore, we need to exploit a high level specification mechanism to represent changeable variables and components of the APCS, and to implement operational functions of the APCS from the system flow to the individual equipment.

The Petri nets are useful specification tools for modeling, analyzing and evaluating discrete event logic of material handling systems, specifically for the given APCS, because concurrency or parallelism, asynchronous processes, deadlock, conflict, and event driven processes can all be considered.

Theories and applications of the Petri nets have been studied by Jensen [9-11], Peterson [19], Viswandahham and Narahari [17,22], Grenrich and Lautenbach [6], Murata [15-16], Choi and Kuo [4]. More specifically, theories and applications of the Petri net model for manufacturing systems with robots have been studied by Crockett, Desrochers, DiCesare, and Ward [5], Paul [18], Weixiong [23], and Martinez, Muro, and Silva [13]. However, these studies provided partial representations of the given APCS, leaving out such characteristics as resource availability, type of processes, multiple products, capacity, priority, and failure rate.

A distinctive advantage of Petri nets is the ability to analyze the model. This analyzing process includes both determining if the model performs correctly (e.g., no deadlocks or boundedness) as well as determining if the Petri net accurately models the actual system. A number of methods have been proposed for the analysis of the Petri net models. For instance, Peterson [19] introduced the reachability tree and matrix equation methods; Jensen [10] describes the invariant-method; Memmi

[14] introduces the algebrac meaning of the invariants; Narahari [17] also reviews the important concept of Petri net invariants and describes a knowledge of the invariants of a Petri net model in the Flexible Manufacturing System context.

Performance analysis of manufacturing systems provides a means of determining system characteristics. Performance analysis of Petri net models has been studied by Magott [12], Choi and Kuo [4], Hillion [7], Ramchandani [20], Sifakis [21], and Arbel and Seidmann [2]. These studies show how measures such as the maximum computation rate and dynamic response time can be determined.

Using previous studies as a starting point, we first examine the APCS and extend the fundamental constructs of Petri net models by adding elements for time, resource availability (number of resources, type of resources), types of processes, multiple products (with and without priority), capacity (buffer size or storage capacity limit), and failure rate (part defect or equipment breakdown). We then build a Petri net model of the given APCS, analyze important qualitative aspects of APCS behaviors and finally evaluate performances of APCS.

A modefied deterministic stochastic algorithm is developed to describe and evaluate the Petri net model of the given APCS. The approach is based on the Petri net graph structure, firing rules, the state of the Petri nets model, and extended properties.

Using this algorithm the input and control mechanisms of the Petri net model are varied, implemented, and evaluated to produce results of performance analysis. Finally, these results can be used to redesign the APCS and directly applied to the design and analysis of the full-scale material handling operation.

AN AUTOMATED PALLETIZED CONVEYOR SYSTEM

Figure 1 shows an APCS that includes pallets, three different parts, one photo-eye, six sensors, sixteen cylinders, and two conveyor circuits with four motors. With the use of photoelectric sensors, features such as sorting, accumulation of dissimilar parts, and automated transfer between conveyor circuits provide simple solutions to material handling problems. The APCS has several areas grouped according to their material handling operations as follows:

- **Accumulation area**: Three different holding lanes maintain separation of dissimilar parts according to their shapes and colors on a feed line to varied machining operations. The outside lane, middle lane, and inside lane hold black large size parts, silver large size parts, and small size (square and round) respectively. It allows a selected quantity of parts to be released from each lane sequentially, starting with the outside lane. The outside lane uses two cylinders (3 and 13) to release a selected quantity of black large size parts, the middle lane uses two cylinders (4 and 14) to release a selected quantity of silver large size parts, and finally the inside lane uses two cylinders (5 and 15) to release a selected quantity of two small size parts.

- **Staging area**: Sensor 4 at this collection area controls and maintains the proper number of parts released from the accumulation area. If the sensor 4 detects a delay in part movement in a certain amount of time, then parts cannot be released during that time from the accumulation area. At this metering area, the parts (detected by sensors 5 and 6) that have different vertical height are simultaneously accepted, otherwise rejected according to size (detected by only sensor 5). All parts including correct parts are held for release by cylinder (6 and 16) to the pickup and return area, and black large size parts and small size parts are allowed to pass through for recycling by cylinder 7.

- **Part pick-up and return area:** A silver large part in the metered release position activates a stop to retain the part for transfer to the machining loop via a pick and place device. Machined parts are transferred back to the outside lane of the feed line. These parts are released when the inside lane is clear. The pick and place device uses cylinder (8, 9, and 10) to move vertically or horizontically and start to move to the pallet load/unload area when a pallet arrives to the rocker type device.

- **Pallet accumulation/metering area:** A rocker type device on the machining loop holds pallets one at a time in the pallet load/unload area as called for by the photoeye and releases pallets when a pallet is loaded by the pick and place device.

- **Pallet load/unload area:** Sensor 7 detects loaded or empty pallets with priority on unloading. This eliminates the possibility of placing another part on a loaded pallet.

- **Part seperation area:** Separation is accomplished with the use of two sensors (1 and 2). The first sensor detects color, and the second sensor measures size. The black large size parts, silver large size parts, and small size (square and round) are separated to the outside lane, middle lane, and inside lane respectively. The separated parts are then held in accumulation lanes.

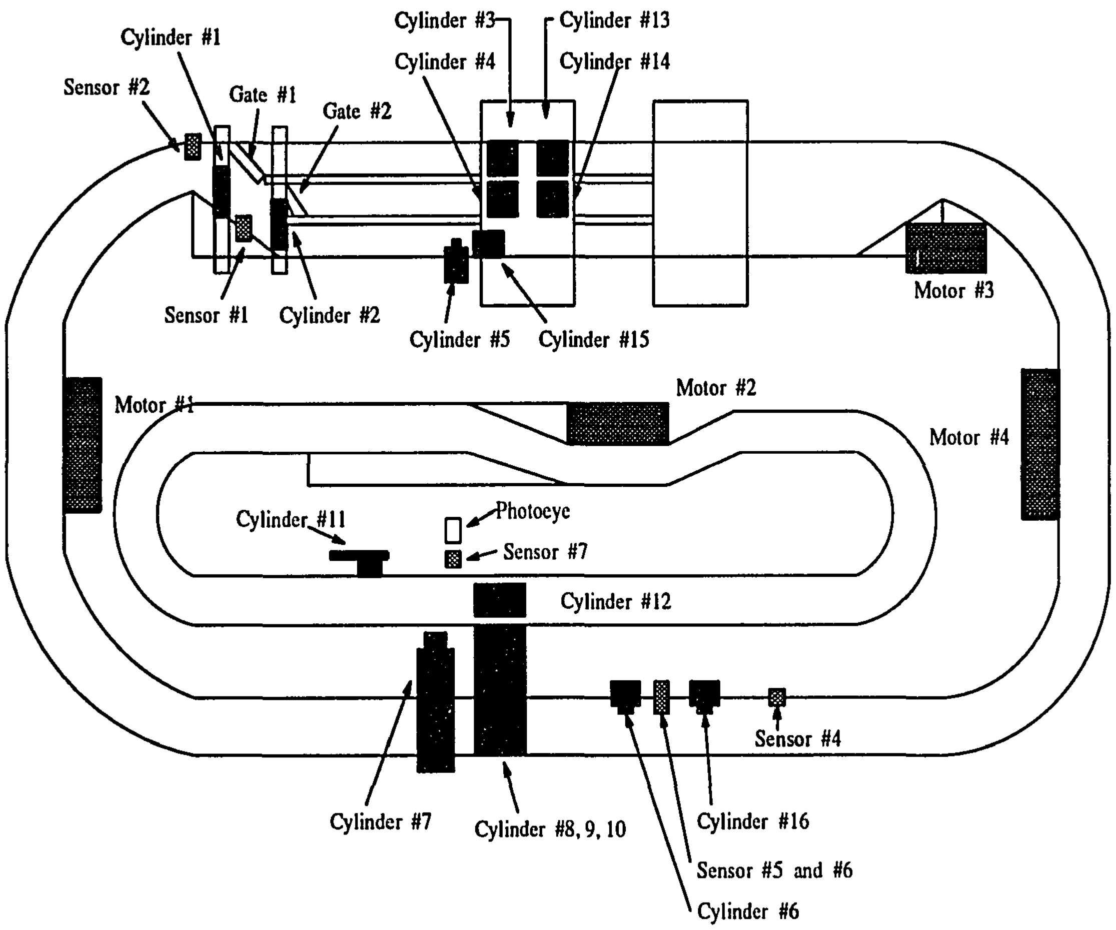


Figure 1: Automated Palletized Conveyor System

MODELING METHODOLOGY

Ordinary Petri nets

- Petri Net (=**PN**)

  A Petri net is a four-tuple, **PN**= (P,T,I,O), where P= $\{ p_1, p_2, \ldots, p_n, \}$ is a set of places, T= $\{ t_1, t_2, \ldots, t_n, \}$ is a set of transitions. The set of places and the set of transitions are disjoint, P $\cap$ T = $\emptyset$. I $\subseteq$ { P $*$ T } and O $\subseteq$ { T $*$ P } are sets of directed arcs.

  A place p_i is an input place of a transition t_j if $p_i \in$ I(t_j); p_i is an output place if $p_i \in$ O(t_j). Similarly, the multiplicity of an input place p_i and output place p_i for a transition t_j is defined as $\#$ (p_i, I(t_j)), and $\#$ (p_i, O(t_j)).

- Marked Petri Net (=**M**)

  A Petri net **M** containing a marking μ is a marked Petri net **M**= (P,T,I,O,μ). Marking μ of a Petri net **PN** is a function from set P to a set of nonnegative integers N, μ: P $\rightarrow$ N, Where μ sets tokens to every place, $\mu_i = \mu(p_i \in$ N indicates the number of tokens in place p_i). We denote a Marked Petri net (=**M**) by (**PN**, μ). We generally associated an initial marking μ_0 with a given **M**

  Tokens reside at a place when it is active. Tokens flow through the net depending on the present marking of the net. The marking of a Petri net is contained in a vector of dimension n, where n is the number of places and each value of the vector corresponds to the number of tokens in the corresponding place.

- Petri Net Graph

  A Petri net graph uses circles to represent places (states) and bars to represent transitions (events). Input-output relationships are represented by directed arcs between places and transitions. A marking μ is represented by tokens in places of the Petri net.

- State (=S) of the Petri net

  Marking μ=(μ_1, μ_2, ..., μ_n)is also called the state of a Petri net. Let state **S** be

$$\mathbf{S}=(s_1, s_2, \ldots, s_n),$$

  where

$$s_i = \begin{cases} 1, & \text{if } \mu_i = \mu(p_i) > 0, \\ 0, & \text{if } \mu_i=\mu(p_i) = 0, \end{cases}$$

  The state **S** shows whether a place has tokens or not.

- Incident Matrix and Firing Rules

  Let **INS** be the (m,n) $\rightarrow$ { 0,1, ..., i } function that defines the multiplicity of an input place of the transition. And let **OUTS** be the (m,n) $\rightarrow$ { 0,1, ..., o } function that defines the multiplicity of an output place of the transition. If the **M** has no self-loops, then the m $\times$ n incident matrix **D** defined by **D= OUTS-INS** characterizes the relationship between places and transitions. Therefore, functions of INS and OUTS of Petri net **M** are represented by two matrices D^- and D^+. Each matrix has m rows for each place and n columns for each transition. The incident matrix **D** is defined by **D**= $D^+[j,i]$ (= # (p_i, I(t_j)) - $D^-[j,i]$ (= # (p_i, O(t_j))[19].

Now a transition t_j is enabled in a marking μ if

$$\mu \geq e[j] \times D^-,$$

where e[j] is the unit m-vector which is zero everywhere except in the jth component.

The result of firing transition t_j in marking μ, if it is enabled, is

$$\mu - e[j] \times D^- + e[j] \times D^+,$$
$$=\mu + e[j] \times (-D^- + D^+),$$
$$=\mu + e[j] \times D.$$

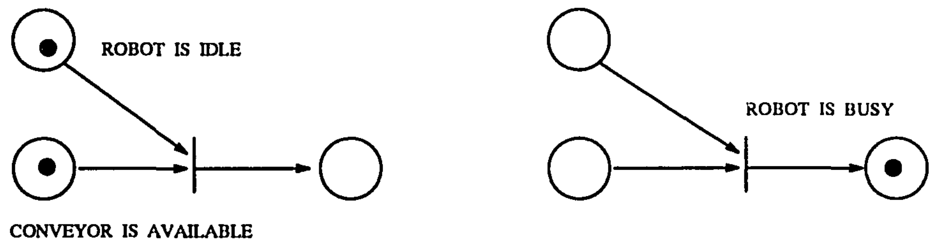
Figure 2(a) shows a Petri net example for accessing a robot. The tokens, places, and transitions correspond to the various elements found in manufacturing systems. Places usually represent resources (e.g., machine, part, and data). A token in a place indicates that the resource is available; otherwise it is unavailable. A place can also be used to imply that a logical condition holds. Transitions are generally used to represent the initiation or termination of an event.

Process Time

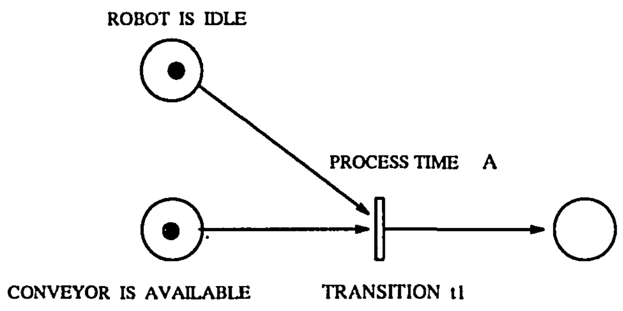
With a set of simple constructs, Petri nets can model a wide variety of discrete event dynamic systems. However, ordinary Petri nets do not account for the passage of time. In most systems, timing is a critical factor for evaluating performance and validating control logic. For manufacturing systems, this is especially true because time is an essential element in functions such as production scheduling and control.

Ramchandani[20] and Sifakis[21] introduced the notion of a Timed Petri net (TPN). Ramchandani described a TPN as a pair (PN,η), where PN is a Petri net and η is a processing time function that assigns a positive rational number to each transition of the net. In a TPN model, each transition t_i(after being enabled)has a time delay of $\eta(t_i)$ before firing. The firing times must be rational so that we can discretize the processing times in units of time and precisely describe the state of the process at each instant of time. The rule of operation of a TPN is similar to an ordinary PN. Once a transition is enabled, the token are removed from the input places and are held for time $\eta(t_i)$, after which the tokens are sent to all the output places. Transitions in TPNs can be viewed as a list of events where multiple sets of tokens can be at different stages of the time delay. The execution of the model would be controlled through the use of a global clock to time events.

A transition associated with time $\eta(t_i)$ is graphically represented using a bar [], which indicates that a token stays in that transition for a processing time $\eta(t_i)$. Figure 2(b) shows a timed Petri net example using a robot. Transition t_1 has an associated delay time of A. When the conveyor and the robot are both available (i.e., a token is present in each place), the processing time for transition t_1 begins. The time delay A represents the material handling time for the conveyor to move a part to the robot.



(a) Petri net example



(b) Timed Petri net example

Figure 2: Ordinary and Timed Petri net example

Resources

- Capacity

  In manufacturing systems, one finds a number of limited resources such as machines

  and robots that, have the same process structure and behavior. Each resource has a fixed number of tokens representing the total capacity. The number of resource tokens in a place indicates the state of those resources (i.e., idle, down, or busy).

- Capability

  Resources are differentiated by their set of capabilities to perform required functions. For example, a milling machine can be used to perform a family of metal removal processes. A Petri net extension that is useful for modeling different types of resources is called Colored Petri nets[7-8,12]. Using a similar approach, we represent token color as a token shape, indicating the identity of a resource. A place can have one or more token shapes that represent a set of different resources. These sets can be

  used to model a system with n different resources that provide different capabilities. A transition can fire with respect to each of these shapes. Transition firing follows rules of ordinary Petri nets except that token shapes must match.

  In other manufacturing environments, there is a many-to-many mapping between product type and resource. This mapping is a direct results of capability requirements of products and available capabilities of resources. If different products (representing different token shapes) require one resource, then the transition cannot fire because the token shape is not matched. We then assign letters to the tokens (a, b, c, ..., x, y, z representing different capabilities). In this case, the transition firing rule will be changed as follows: (1) basically, a transition can fire with respect to the same token; (2)when a different token shape is matched, transition firings follow the rules of ordinary Petri nets except that the letter must be checked. If the letter is matched with the different token shapes, then the transition can be fired.

- Failure Rate

Resource failure is a common stochastic element of behavior. We add a useful extension to Petri nets by allowing a failure rate (percentage of time that a given resource is down) to be assigned to each token. The resource failure rate **FR** specified by

$$\mathbf{FR} = \frac{\text{Repair time}}{\text{Operation time} + \text{Repair time}}$$

is calculated under the assumption that operation and repair time follow the exponential or weibull distribution. When a transition with a non-zero failure rate fires, a random number is generated with a probability **FR** of needing repair. This probability is used to determine which arc should receive a token. If failure does occur, then the resource token moves along the failure arc to a place where the resource failure can be modeled. The resource token is not available until the resource has been repaired. If the resource fails due to the random variable after the transition fires in Figure 3, the resource token in place p_1 is moved to place p_r; then it takes a repair time **R** to make the token available in place p_1 again.

Products Structure

- Types of product

  In manufacturing systems, a number of products, each having different process plans, must be considered. Again, token shapes (based on Colored Petri nets) are used to distinguish the products. Each unique shape has a corresponding process plan, and each process plan has a procedure associated with different resources. When a token arrives at all of the input places of the transition, the procedure is executed.

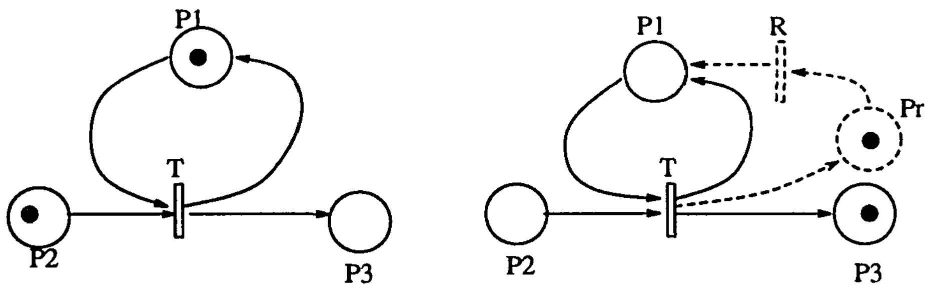


Figure 3: Modified Petri net examples with failure rate

- Process plan

  To complete a process plan, a part uses a number of resources with different capabilities and potentially competes with other parts for the same resources. The process plan for a part can be represented by a unique sub-net. The processing times for each step are incorporated into the transitions as described earlier.

- Priority

  Most manufacturing systems can have a large number of multiple products, with demand varying from high volume products (that are continuously produced) to low volume products (that are produced intermittently). Conflict arises when work orders compete for a single resource. Since a single resource cannot process these work orders simulationously, there must be a set of rules to determine the order in which the work orders are processed. In situations like this, several questions are raised.

1. Which product should be produced first?
2. Which operation should take place first?
3. How many products should be produced according to the inventory and limited capacity?

In order to answer these questions, transitions and tokens are assigned priorities. If two or more transitions are enabled by one or more of the same places, we assign a priority to the transitions on the basis of which transition should fire first [16, 19]. This is done by assigning a different priority number (1,2, ..., n) to each transition, with 1 being the highest priority and n being the lowest priority. In Figure 4(a), two transitions (t_1, t_2) attempt to execute at the same time, but transition t_1 fires because it has a higher priority.

Priorities can also be assigned to token shapes as shown in Figure 4(b). The circle token has higher priority (priority 1) than the square token (priority 2), so the circle token always fires first. In addition, a higher priority can be assigned to a token that has arrived earlier than another token requiring use of the same resource in a particular queue of the manufacturing systems.

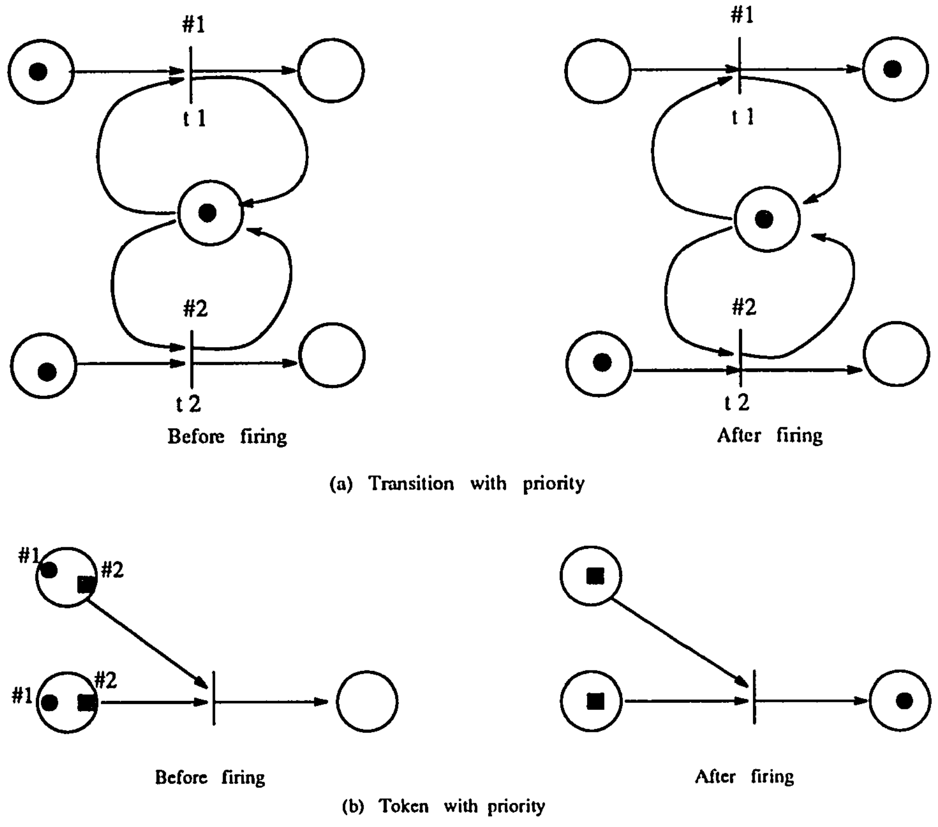


Figure 4: Modified Petri nets example with priority

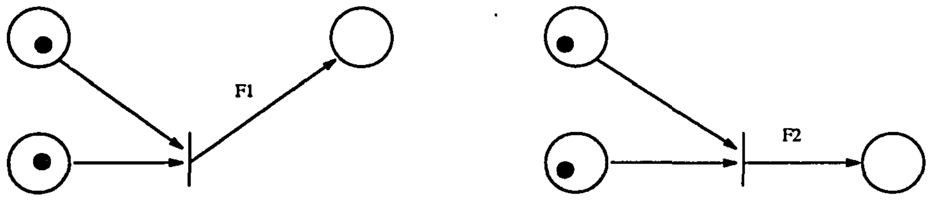
- Defective parts

  After processing, a product is either within specifications or outside of specifications (defective). We model this behavior as independent Bernoulli experiments with a probability p of being outside of specifications. a uniform random number between 0 and 1 is generated when the transition is fired to determine which edge should receive a token. If a failure does occur, the token (i.e., defective part) can be directed to a failure arc that leads to a place where rework is performed; otherwise, the token leaves the system. If both the resource and the part fail, then the procedure for resource failure (as described in Section 3.3) is performed simultaneously.

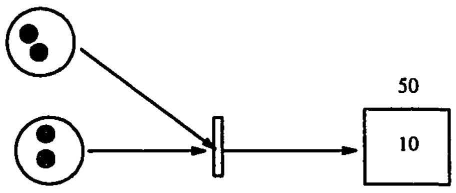
  In Figure 5(a), we have two possible outcomes for the assembly process of a robot, namely a failure or success, along with their probabilities F ($=p$) and F-1 (1-p), respectively. If the defective part occurs after firing the transition (which takes process time T), the token is sent either to the place representing rework or it is disposed of, depending on the quality requirements of the product. With probability 1-p, the part enters the next process.

Storage

Buffers and storage areas are used throughout manufacturing systems. The size or capacity of these areas is an important consideration in the design of such systems. Capacity-designated Petri nets, as introduced in the literature [1, 16], allow for the representation of limited storage space. For instance, a buffer could be represented by a capacity-designated Petri net. We use a modified-capacity Petri net in which a capacity-designated place has a number representing storage capacity limit and another number representing inventory. The difference between these numbers is the space available for tokens. A capacity-designated place is graphically represented by using a large empty square, with the number inside the square indicating inventory (i.e., number of tokens) and the number outside the square indicating the storage capacity limit. Capacity-designated places will prevent input transitions from firing (i.e., blocking) if the inside number is equal to the outside number. Figure 5(a) shows a modified-capacity Petri net.



(a) Failure rate for the defective part



(b) Modified Petri net example with storage

Figure 5: Modified Petri net examples with product

MODELING THE APCS

A Proposed Petri Net Model

We now present our approach to the modeling of the given APCS using extended Petri nets. Petri nets are developed to model discrete event systems with several tuples requiring interaction with each other. They are useful modeling tools because concurrency or parallelism, asynchronous processes, deadlock, boundedness, conflict, and event driven processes can all be considered.

The APCS is composed of separate interacting components as shown Figure 1. The APCS involves numerous concurrent and sequential interactions on various part types and control processes for parts handling equipment. The processing for each part follows a sequence of operations based on the process plan.

The acquisition of a resource occurs when it is idle, and the release of a resource occurs when processing is completed. We assume the processing times can be represented deterministically, which is not without precedence for automated systems. Deterministic processing times are assigned to each process and are represented by each transition. Conflict occurs when two or more processes require a common robot at the same time. Starting with previous contexts of the APCS and ordinary Petri net constructs, we can formulate Petri net models to partially represent the APCS in Figure 6.

To complete the Petri net model of the APCS, we add our extensions previously described in the following manner. In Figure 7, three different parts (represented by hexagon, square, triangle), each having different process plans, are modeled. Also, three different resources (represented by circles) with different capability

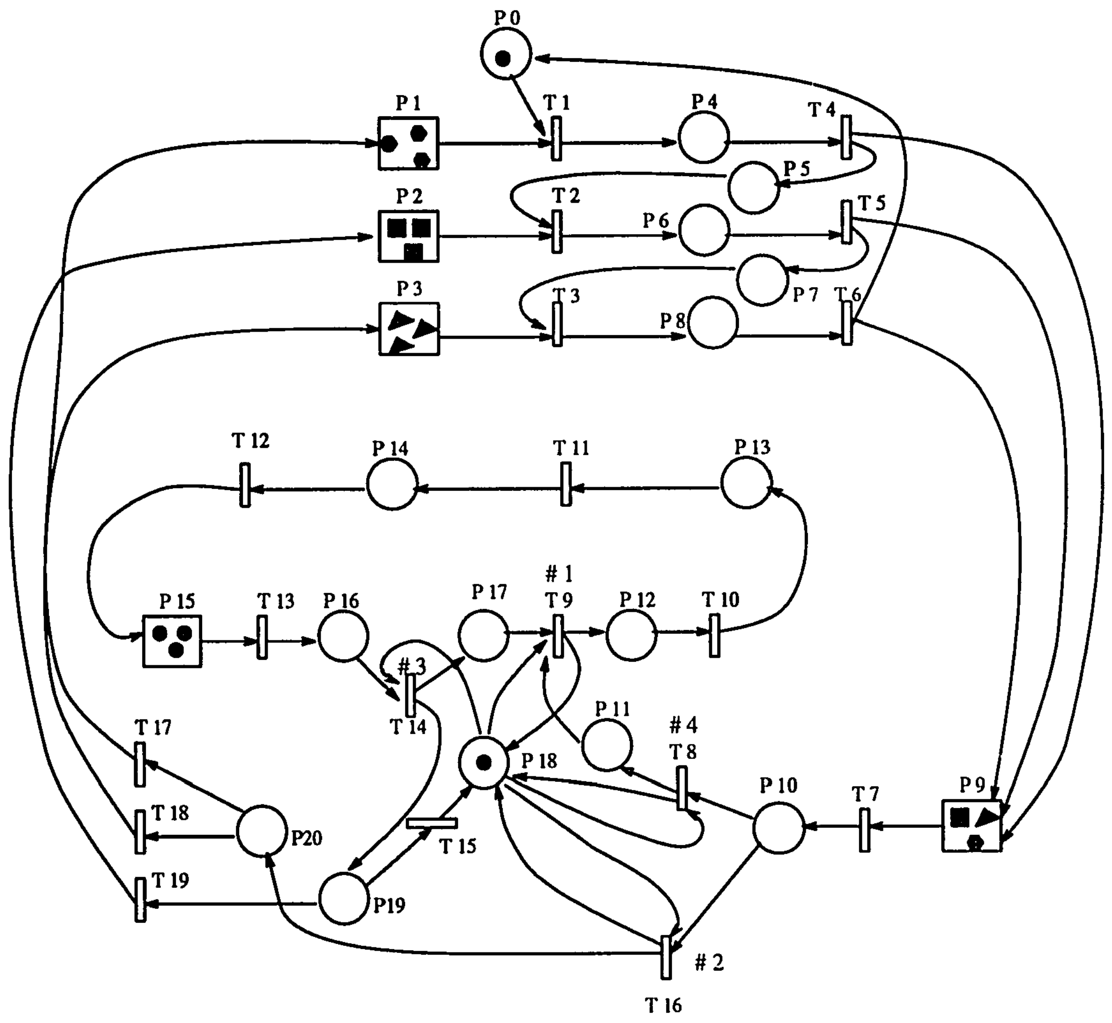


Figure 6: Ordinary Petri nets model

(represented by a, b, c, ...) can be assigned in place $CR_{i,j}$, RR, and NOP for different process plans. The number of these parts and resources can be controlled by increasing or decreasing the tokens in places NP_1, NP_2, NP_3, $CR_{i,j}$, RR and NOP. Two types of places (representing ordinary petri net places and place capacity) are used in the following ways; place NP_1, NP_2, NP_3 and NP_4 to represent buffer areas and place NOP to represent the initial number of pallets with capacities n_i, i=1 to 5, respectively. These places are graphically represented using a large empty square, with the number inside the square indicating an available number of tokens representing inventory and a number outside the square indicating the buffer capacity limit.

Three parts use cylinders and robots with different capabilities that potentially compete with other parts for the same resources. Each process plan for the three parts can be represented by a unique subnet. The processing times for each plan are incorporated into the transitions. When a token arrives at all of the input places of the transition, the procedure is executed based on the firing rule previously described in section 3.1., and if a conflict case occurs, then the procedure is executed according to the priority of the transitions. For instance, when transition t_8 and t_9 can be executed at the same time, transition t_9 can be processed first according to the first priority of the transition. However, in this Petri net model, we do not consider priority tokens as described in section 3.4.

Failure rate also is introduced in the model in the following way. Robots and cylinders are given failure rates. If a robot or one of the cylinders fails according to an exponential distribution, then the resource token in place RR moves to place $P_{r'}$ and the token delays a deterministic amount of repair time in transition $T_{r'}$ before

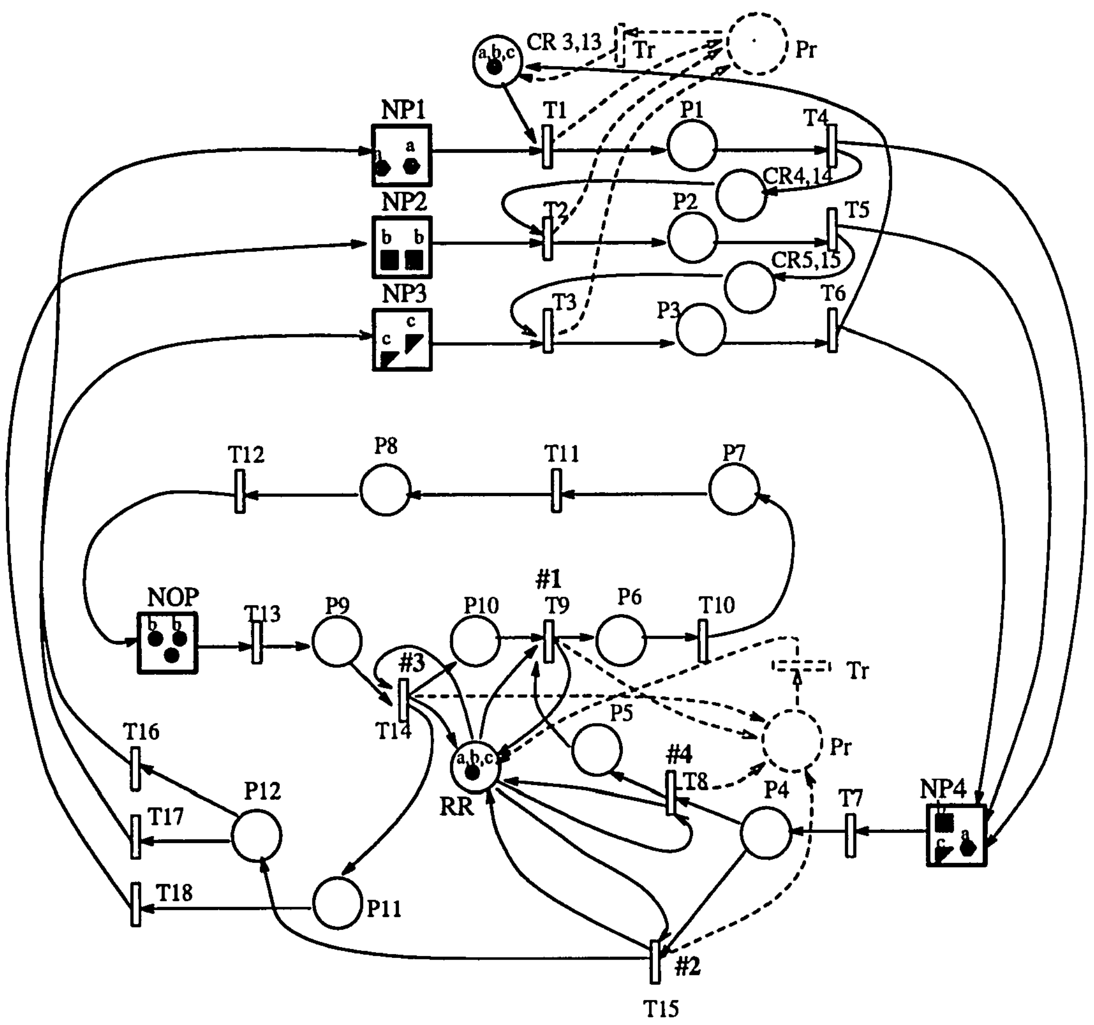


Figure 7: Extended Petri nets model

returing to place RR. The interpretation of the places and transitions of the Petri net model in Figure 7 is as follows:

- C_{ij} : cylinder i and j ready to process a job
- NP_i : number of parts in buffer
- NOP : number of occupied pallets in buffer
- P_i : part ready for processing in place i
- RR : robot ready to process a job
- T_i : a material handling device completely processes a job, start to finish.

Analyzing the Proposed Petri Net Model

After modeling the APCS with Petri nets, we consider several properties of the Petri net model for analyzing, namely: liveness, boundedness, and conservativeness[10,19]. This analysis can lead to a better understanding of the qualitative aspects of the APCS's behavior.

In this paper, the invariant method[10,14,17,19] can be used to analyze a proposed Petri net model for the given APCS, because a knowledge of the invariants is useful for analyzing some important properties of Petri nets such as properness, liveness, boundedness and conservativeness.

We have two kinds of invariants[19]: the *p*-invariant (implies that the weighted sum of the number of tokens in each place of the Petri net model is constant for all reachable markings, the weights being given the *p*-invariant) and *t*-invariant (implies that a *t*-invariant will give the number of times different transitions should be fired in order that a particular marking may be reproducible)[17]. Therefore, the p-invariant will be considered to analyze the qualitative aspect of the APCS behavior such as deadlocks, blocking and starving of a robot, and buffer overflow.

Table 1 shows the p-invariants of the Petri net model for the given APCS (the quantities of w_i are integers). If μ_0 is the initial marking of the Petri net model, corresponding to the initial state, we have

$$\begin{aligned}
\mu_0(NP_1) &= n_1 \\
\mu_0(NP_2) &= n_2 \\
\mu_0(NP_3) &= n_3 \\
\mu_0(NP_4) &= n_0 \\
\mu_0(NOP) &= n_4 \\
\mu_0(CR_{3,13}) &= 1 \\
\mu_0(RR) &= 1 \\
\mu_0(P_1) &= 0 \\
\mu_0(P_2) &= 0 \\
\mu_0(P_3) &= 0
\end{aligned}$$

Let μ be any reachable marking from the initial marking (μ_0) of the Petri net model. If W(μ) denotes the weighted sum of the number of tokens in each place in marking μ, we have W(μ_0) and W(μ) as follows:

$$\begin{aligned}
W(\mu_0) &= w_1 n_1 + w_2 n_2 + w_3 n_3 + w_6 n_4 + w_4 + w_5 \\
W(\mu) &= w_1\mu(NP_1) + w_2\mu(NP_2) + w_3\mu(NP_3) + w_4\mu(CR_{3,13}) + (w_1 + w_2)\mu(P_1) \\
&\quad + w_4\mu(CR_{4,14}) + (w_2 + w_4)\mu(P_2) + w_4\mu(CR_{5,15}) + (w_3 + w_4)\mu(P_3) \\
&\quad + (w_1 + w_2 + w_3)\mu(NP_4) + (w_1 + w_2 + w_3)\mu(P_4) + w_5\mu(RR) + w_6\mu(NOP) \\
&\quad + w_2\mu(P_5) + (w_2 + w_6)\mu(P_6) + (w_2 + w_6)\mu(P_7) + (w_2 + w_6)\mu(P_8) \\
&\quad + (w_2 + w_6)\mu(P_9) + w_6\mu(P_{10}) + w_1\mu(P_{11}) + (w_1 + w_3)\mu(P_{12})
\end{aligned}$$

To show the p-invariant method for a Petri net model of the given APCS, we consider the conservation problem: showing that tokens are neither created or destroyed. Stated another way, a weighted sum of the number of tokens in each place at any instant in time should be constant (i.e., invariant). Hence, we have the following equations:

$$\begin{aligned}
n_1 &= \mu(NP_1) + \mu(P_1) + \mu(P_3) + \mu(NP_4) + \mu(P_4) + \mu(P_{12}) \\
n_2 &= \mu(NP_2) + \mu(P_2) + \mu(NP_4) + \mu(P_4) + \mu(P_5) \\
&\quad + \mu(P_6) + \mu(P_7) + \mu(P_8) + \mu(P_9) + \mu(P_{11}) \\
n_3 &= \mu(NP_3) + \mu(P_3) + \mu(NP_4) + \mu(P_4) + \mu(P_{12}) \\
n_4 &= \mu(NOP) + \mu(P_6) + \mu(P_7) + \mu(NP_8) + \mu(P_9) + \mu(P_{10}) \\
1 &= \mu(CR_{3,13}) + \mu(P_1) + \mu(CR_{4,14}) + \mu(P_2) + \mu(CR_{5,15}) + \mu(P_3) \\
1 &= \mu(RR)
\end{aligned}$$

From these results, we observe that the Petri net model for the given APCS is bounded (no overflow), live (this system does not have deadlocks because transitions are enabled to fire in every state of the APCS), and non-conservative (the number of jobs being processed does not remain a constant at all times). In addition, we also may assume that starving and blocking of cylinders and robots are possible in the APCS.

Table 1: Place and weighted vectors for each place

place	weighted vectors for places	place	weighted vectors for places
NP_1	w_1	RR	w_5
NP_2	w_2	NOP	w_6
NP_3	w_3	P_5	w_2
$CR_{3,13}$	w_4	P_6	$w_2 + w_6$
P_1	$w_1 + w_4$	P_7	$w_2 + w_6$
$CR_{4,14}$	w_4	P_8	$w_2 + w_6$
P_2	$w_1 + w_2$	P_9	$w_2 + w_6$
$CR_{5,15}$	w_4	P_{10}	w_6
P_3	$w_1 + w_3$	P_{11}	w_2
NP_4	$w_1 + w_2 + w_3$	P_{12}	$w_1 + w_3$
NP_4	$w_1 + w_2 + w_3$		

PERFORMANCE ANALYSIS

A simple Petri net model of the logical and causal dependencies in the manufacturing system is not sufficient to answer APCS characteristics such as time, resources availability, multiple products, different processes, priority, and capacity. Adding these characteristics allows for such a temporal performance analysis. The main applications of these nets will be in the APCS.

Two types of constraints affect the performance of the APCS. The first type is related to the internal structure that determines how the various procedures work in the system; some procedures are processed sequentially, and others are processed concurrently. The APCS has both sequential and concurrent processing activities. The second type of constraint magnify time, resources, multiple parts, different processes, capacity, and priority according to the APCS context. The APCS has a limited amount of productive time because of the constrained resources. In addition, resource availability and capacity limits are used to represent the productive capacity of the system. Priority is used to resolve conflict situations that arise when work orders compete for a single resource.

Maximum Production Rate of the System

If demand is continuous at a rate that is low enough, the APCS will be able to meet all the demand. In this case, the rate at which products are being processed will precisely correspond to the demand rate. However, beyond a certain demand rate, products will compete for resources, creating a backlog of work. This bound precisely determines the maximum production rate of the APCS, and is a function of time, resources, capacity, failure rate and priority. This measure of performance, which characterizes the maximum rate of processing of the overall system, is important because it limits the allowable rate of supply that can be produced.

The Process Schedule

The production schedule specifies the time at which work orders are released into the system. Assume that the processing starts at time = 0 and that a large demand forces processing to occur continuously at the maximum production rate. In this section, we want to determine the schedule for the various processes (represented by transitions) for various demand rate. From the flow time (the time interval between the moment the demand was received and the moment a product was made), we can determine the tardiness of the work order.

By changing the control variables in the model (e.g. resources, time, etc.), and executing the Petri net model for the given APCS, we can determine execution schedules for each process and a production schedule for scheduling purposes. Therefore, the production schedule of the APCS will be changed on the basis of process schedules for each process and characteristic dynamic behaviors of the APCS. Since process times are assumed deterministic, the process schedule computed here will characterize the deterministic behavior of the APCS. Starting from the initial stage, the process can be continued repetitively until demand is satisfied. Thus, the best performance of the APCS will be obtained with respect to system characteristics such as time, resource availability, capacity, priority, and demand. The measures of performance described above are important in evaluating the performance of the APCS. If one demand arrives or multiple demands arrive simultaneously, it will be possible from the process schedule to evaluate the performance of the APCS.

Numerical Results

Using our extended Petri net models, we have implemented a deterministic and stochatic algorithm to describe and evaluate the APCS in Figure 7. The approach is based on the Petri net graph structure, firing rules and the state of the Petri net model. We create an input file (including transitions, places, input-output relations of transitions, tokens in each place, processing time for each transition, resource availability, number of parts, capacity of the storage place, priority, and failure rate) in order to model and evaluate the APCS. By changing different variables, we can create output files with the following results:

- The maximum production rate of the APCS [20]

  From the incident matrix, we can determine all the circuits (possible flows of tokens), the maximum circuit time (longest circuit flow time) and the throughput of the Petri net model[7,20].

- Processes and production schedules

  Schedules for the various processes (represented by transitions) and production schedules for various demand rates can be evaluated.

- Availability (percentage of idle time) of resources in the APCS.

  From the state variable (**A**) are defined as followed:

$$A = \begin{cases} 1 & \text{if resource is available} \\ 0 & \text{if resource is not available} \end{cases}$$

  We know the availability and utilization for resources in place RR, $CR_{i,j}$ and NOP of the Petri net model at consecutive time instances.

- Statistical analysis

  Confidence intervals for maximum production rates on the basis of resource failure rates and deterministic repair times.

We investigated five cases in which we perturbed the APCS to produce results that can be applied to redesign the APCS and to design and analysis of the material handling operation. The perturbations included the processing time for each transition (case 1), resource availability (case 2), number of resources (case 3), storage capacity (case 4), and priority and failure rate (case 5) as follows:

- Case 1

  We initially used real deterministic processing times (t_1 = 1.5 seconds, t_2 = 1.5 seconds, t_3 = 1.5 seconds, t_4 = 12.0 seconds, t_5 = 12.0 seconds, t_6 = 12.0 seconds, t_7 = 0.5 second, t_8 = 1.0 second,t_9 = 7.0 seconds, t_{10} = 8.0 seconds, t_{11} = 10.0 seconds, t_{12} = 10.0 seconds, t_{13} = 1.5 seconds, t_{14} = 9.5 seconds, t_{15} = 1.0 seconds, t_{16} = 16.0 second, t_{17} = 16.0 seconds, t_{18} = 16.0 seconds) and varied these times to increase the maximum production rate.

  In this case, there are three process plans found in the Petri nets model of the given APCS based on the different parts. Each process plan has a procedure associated with different shapes of token as follows:

  - Black large size parts are represented by hexagons

    t_1=1.5 second, t_4 = 12.0 seconds, t_7 = 0.5 second, t_{16} = 1.0 second, t_{18} = 16.0 seconds.

  - Silver large size parts are represented by squares

t_2 = 1.5 seconds, t_5 = 12.0 seconds, t_7 = 0.5 second, t_8 = 1.0 second,t_9 = 7.0 seconds, t_{10} = 8.0 seconds, t_{11} = 10.0 seconds, t_{12} = 10.0 seconds, t_{13} = 1.5 seconds, t_{14} = 9.5 seconds, t_{19} = 16.0 seconds.

– Small size parts are represented by triangles

t_3 = 1.5 seconds, t_6 = 12.0 seconds, t_7 = 0.5 second, t_{16} = 1.0 second, t_{17} = 16.0 seconds.

On the basis of the incident matrix, three circuits (three possible flows of tokens) with the longest one being the critical are founded in the Petri net model[7,20] The critical circuit of this Petri net model is associated with processes t_2, t_5, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}, t_{19}. This critical circuit determines the critical circuit time (72.5 seconds) that indicates the maximum flow time of the APCS. However, in general, the maximum production rate governed by the bottleneck. Because of the configuration of the APCS, the maximum production rate (= 1/minimum flow time) is used as a measure of system performance.

To increase the maximum production rate (0.014 per second), we must reduce processing time for transitions on the critical circuit. For example, if the process time for transition t_7 is reduced from twelve to six, then the critical circuit time becomes 66.5 seconds which is an increase in the maximum production rate.

- Case 2

The real deterministic process time is fixed and the number of moving pallets in place NOP is varied to determine the number of pallets that maximize the resource utilization.

To solve this kind of problem, first of all, the APCS needs to increase moving pallets in place NOP from one to three. This would increase the maximum production rate from 0.014 to 0.017 per second. Even if there are more than two pallets in place NOP, we still have the same maximum production rate. Therefore, we do not need more than two pallets to increase the maximum production rate and utilization of the APCS without improving process times for $t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_{19}$.

- Case 3

  In this case, the real deterministic process time, and number of moving pallets (same as used in case 2) are fixed, but the number of robots in place RR is varied by two. Then the maximum production rate is increased but it is difficult to install one more robot in the APCS because of initial bad space allocation. Still, this result can be considered when we design a full-scale APCS in the material handling facility.

- Case 4

  The real deterministic process time, number of moving pallets (same as used in case 2) and the number of robots (=1) are all fixed, but capacities of the places NP_1, NP_2, NP_3, NP_4 are varied. In this case, it is possible to reduce the number of three different parts available to the buffer area NP_1, NP_2, NP_3, NP_4 up to each of two parts to maintain the same production rate.

- Case 5

  A failure rate (0.0125, 0.025, 0.0375, and 0.05) and a deterministic repair time (100 time unit) has been given to a robot in place RR on the basis of the exponnential distribution with $\lambda = 0.0025$ and $\mu = 0.1975$, $\lambda = 0.005$ and $\mu = 0.195$, $\lambda = 0.0075$ and $\mu = 0.1925$, $\lambda = 0.01$ and $\mu = 0.19$ per hour, respectively. If the robot in place RR fails, then the system cannot continue to perform its tasks.

Table 2: Confidence interval for maximum production rate

failure rate	confidence interval for maximum production rate
0.0125	0.0165 - 0.0175
0.025	0.0162 - 0.0177
0.0375	0.017 - 0.0178
0.025	0.0169 - 0.0181

The confidence interval for the maximum production rate that can be obtained by using the stochastic algorithm was run 1000000 times on the basis of the deterministic time (case 2), number of moving pallets in place NOP (= 2 to 10), number of robot in place RR (= 1) and different failure rates. Table 2 shows confidence intervals of 95 % for the maximum production rate under different failure rates.

CONCLUSION

In this paper, we first examine the APCS and extend the fundamental constructs of Petri net models. We then build a Petri net model of the given APCS, analyze important qualitative aspects of APCS behaviors and finally evaluate performances of APCS.

A modified deterministic and stochastic algorithm is developed to describe and evaluate the Petri net model of the given APCS. The input and control mechanisms of the Petri net model are varied, implemented, and evaluated to produce results that can be used to redesign the APCS and also can be directly applied to the design and analysis of the full-scale material handling operation.

BIBLIOGRAPHY

[1] C. Andre, P. Armand, and F. Boeri, "Synchronic Relations and Applications in Parallel Computation", *Digital Processes* 5, 99-113, 1979.

[2] A. Arbel, and A. Seidmann, "Performance Evaluation of Flexible Manufacturing Systems", *IEEE Transactions on Systems, Man, and Cybernetics*, 1(5), 606-617.

[3] B.W. Choi, and W. Kuo, "Performance Analysis of a Generic Naval C2 Battle Group System by Use of Timed Petri Nets", Proceedings of the Winter Simulation Conference, San Diego, 1988.

[4] B.W. Choi, and W. Kuo, "Petri Net Extensions for Modeling and Validating Manufacturing Systems", Int. J. Prod. Res., (accepted for the publication by).

[5] D.H. Crocket, A.A. Desrochers, F. DiCesare, and T. Ward, "Implementation of a Petri Net Controller for a Machining Workstation", *Proceedings of a Conference on Robotics and Automation 3*, IEE Society Press, New York, 1861-1867, 1987.

[6] H.J. Genrich, and K. Lautenbach, "System Modeling with High-Level Petri Nets", *Theoretical Computer Science* 13, 109-136, 1981.

[7] H.P. Hillion, "Performance Evaluation of Decision Making Organization using Timed Petri nets", M.S. thesis, M.I.T., (Cambridge, Mass.).

[8] A.L. Hopkins, and G.R. Walker, "The State Transition Diagram as a Sequential Control Language", *Proceedings of the 25th IEEE Conference on Decision and Control* 1096-1101, Dec, 1986.

[9] K. Jensen, "Coloured Petri Nets", *Net Theory and Applications*(Lecture Notes in Computer Science), No.254, Springer-Verlag, Bad Honnef, Germany, 248-299, 1987.

[10] K. Jensen, "Coloured Petri Nets and the Invariants-Method", *Theoretical Computer Science* 14, 317-336, 1981.

[11] K. Jensen, "High-Level Petri Nets Applications and Theory of Petri Nets", *Informatic-Fachberichte* 66, Springer-Verlag, 166-180, 1981.

[12] J. Magott, "Performance Evaluation of Concurrent Systems using Conflict- Free and persistent Petri Nets", *Information Processing Letter* 26, 77-80.

[13] J. Martinez, P. Muro, and M. Silva, "Modeling, Validation and Software Implementation of Production Systems Using High Level Petri Nets", *Proceedings of the 1987 IEEE international Conference on Robotics and Automation* Raleigh, N.C., 1180-1185, April, 1987.

[14] G. Memmi, and G. Roucairol, "Linear Algebra in Net Theory", *Net Theory and Applications*(Lecture Notes in Computer Science), Springer-Verlag, Hamburg, Germany, 213-223, 1979.

[15] T. Murata, and N. Komoda, "Liveness Analysis of Sequence Control Specifications Described in Capacity-Designated Petri Nets using Reduction", *Proceeding of the IEEE International Conference on Robotics and Automation*, Raleigh, N.C., 1985-1990, April 1987.

[16] T. Murata, "Petri Nets : Properties Analysis and Applications", *Proceedings of the IEEE,* 77(4), April 1987.

[17] Y. Narahari, and N. Viswanadham, "A Petri Net Approach to Modeling and Analysis of Flexible Manufacturing Systems", *Annals of Operations Research*, 3, 449-472.

[18] F. Paul, "Time, Petri Nets, and Robotics", *IEEE Transactions on Robotics and Automation*7(4), 417-433, 1991.

[19] J.L. Peterson, Petri Net Theory and the Modeling Systems, Prentice-Hall, Englewood Cliffs,N.J., 1981.

[20] C. Ramchandani, "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets", Technical Report No.120, Laboratory for Computer Science, M.I.T., Cambridge, Mass, 1974.

[21] J. Sifakis, "Performance Evaluation of Systems using Nets", *Net Theory and Applications*(Lecture Notes in Computer Science), Springer-Verlag, Berlin, Germany, 307-319, 1978.

[22] N. Viswanadham, and Y. Narahari, "Coloured Petri Net Models for Automated Manufacturing Systems ", *Proceedings of the IEEE International Conference on Robots and Automation*, Raleigh, N.C., 1985-1990, 1987.

[23] Z. Weixiong, "Representation of Assembly and Automatic Robot Planning by Petri Net", *IEEE Transactions on Systems, Man, and Cybernetics*19(2), 418-422, April 1989.

PAPER IV.

AN ENHANCED METHOD FOR MANAGING PROBLEMS IN A FLEXIBLE MANUFACTURING MACHINE

ABSTRACT

One of the problems that arises in flexible manufacturing environments is minimizing the number of tool changes. We introduce and review this problem as an overall model that can be formulated as a linear and non-linear integer problem. We then extend this model on the basis of two more constraints: (1)jobs that require more than C tools, with C representing the magazine capacity of the machine, and (2)the increased processing time that is required for tuning the tool offset after a tool in slot #1 is changed. Since this model increases computational complexity, we propose a heuristic approach for job sequencing. This approach is locally optimized to minimize the number of tool changes.

Next, we introduce the fundamental constructs of the Petri net models to describe sequence control specifications for a flexible manufacturing machine. We then examine a flexible manufacturing cell that has two automated guided vehicles and a milling machine (DM4400) with an automatic tool changer. Finally, we build a Petri net model as the interpretation schema and implementation model with respect to the local optimal job sequence, the tool changing-procedure, and the machining job of the milling machine.

INTRODUCTION

The tool-changing and machining problem is recognized as having a key effect in flexible manufacturing environments, an effect that creates financial problems on flexible manufacturing systems(FMS). Automatic tool-changing and machining are seen as major factors for an FMS when a set of computer numerically controlled(CNC) machines is used to manufacture parts. Each machine has a limited-capacity tool magazine. An automatic tool-changer makes it possible to perform several sequential operations on a part, without incurring the setup delays necessary for changing tools manually. In FMS, all machines need (1) tool changes before machining jobs, and (2) control procedures to observe and supervise machining processes. The time required for making tool changes and control procedures becomes a significant portion of the total job processing time, which, in turn, implies that processing time can be reduced by minimizing the number of tool changes and using efficient control methods for machining processes. The reducing processing time is the general motivation for studying a machine-loading problem in which the total number of tool changes is minimized (Section 2,3), and a machine control problem (Section 4).

Stecke and Browne [1], Mortimer [2], Kiran and Krason [3], and Dupont-Gatelmand [4] have introduced advanced concepts of the flexible manufacturing system and environments and have presented descriptions of the flexible manufacturing machine. A machine-loading problem in FMS which was studied originally by Stecke [5] and Stecke and Talbot [6], has been examined and extended by Tang and Denardo [7,8]. In their study, the keep tool needed soonest(KTNS) policy is proved on the basis of the validity of the KTNS that was already established by Roger [9] and Mattson et al. [10]. Roger and Mattson's study is in the context of computer memory storage techniques in the situation where each job requires exactly one tool, and is an optimal method for studying the tool-changing problem.

Tang and Denardo [7,8] believed the total number of tool changes can be minimized on the basis of two performance criteria: (1) minimize the number of tool changes and (2) minimize the number of times tools are changed. In addition, Daskin, Jones, and Lowe [11] analyze implementations of the tool-changing problem associated with a flexible system that produces flat sheet-metal parts with interior holes. Bard [12] considered the problem of scheduling N jobs on a single machine equipped with an automatic tool interchanger. This problem included two considerations: (1) the total number of tools required to process all N jobs is greater than the capacity of the tool magazine, (2) processing times and changing times are independent.

In previous studies, authors assumed that the tool magazine has C tool slots (magazine capacity) and each job requires no more than C tools. Researchers also assumed that the tool-changing time is the same when any tools are changed. However, our industrial background is somewhat different from theirs.

We surveyed FMS environments including several milling machines and found two differences. First, several jobs required more than C tools at certain times. This implied that a single job cannot be accomplished without changing tools. Second, one tool was chosen as a reference tool. This tool was used to setup the X, Y, Z axes (=0,0,0) at the beginning of the job process. However, other tools have different lengths that may be either longer or shorter than the reference tool. This difference is called the "tool offset", which may be either positive or negative. All Z axis moves must add or subtract this offset so that the actual cutting height of the other tools is the same as the reference tool [13]. Once these offsets have been calculated the appropriate values are then simply retrieved from the memory as they are needed. Therefore, we need more processing time when a reference tool is changed than when other tools are changed.

The model just described considers two constraints and gives solutions for an optimal job sequence and for minimizing the number of tool changes. But this model has increased computational complexity that is *NP hard* because it is clear that an optimal job schedule can only be determined by solving the tool changing-problem for each individual job schedule. This indicates that the above model is not realistic. Therefore, we need to develop a heuristic approach that can be locally optimized to minimize tool changes and also simplify the necessary computations.

First, we consider the case in which the tools for all operations are kept in the tool storage area. If all the requisite tools are not initially placed on the magazine, one or more tool changes must occur before the machining job can be processed. A tool change occurs when a tool is removed from the magazine and a different tool is inserted on the magazine of the machine.

On the basis of these basic assumptions, an overview of the machine loading problem in which the total number of tool changes is minimized, a literature review, and the objective of this paper are introduced in section 1. We then extend the overall model for minimizing the number of tool changes and also show that this model has increased computational complexity in section 2. In section 3, we develop a heuristic approach for finding a solution that is locally optimized with respect to job sequencing and minimizing tool changes.

In section 4, we introduce the fundamental constructs of the Petri net models to describe sequence control specifications [14-18] for a flexible manufacturing machine.

We then examine a flexible machine cell(FMC) that has two automated guided vehicles(AGVs) and a milling machine (DM4400), which can hold 10 tools and has an automatic tool changer [13]. We also build a Petri net model as the interpretation schema and implementation model with respect to the local optimal job sequence on the basis of the heuristic approach described in section 3, the tool changing procedure, and the tooling job of the milling machine.

PROBLEM STATEMENTS AND MODELING

Suppose that a batch of jobs has to be processed, one at a time, on a single flexible machine in the flexible manufacturing system. Let **N** be the number of jobs and M be the total number of tools required to process **N** jobs. Each job requires a subset of tools that is represented by an M $\times$ N matrix **A** with a_{ij} = 1 if job j requires tool i and a_{ij} = 0 otherwise, for i = 1,2, ... , M and j = 1,2, ... , N. The tool magazine has a limited capacity $C(< M)$, and sometime jobs require more than **C** tools.

We assume that the tool magazine always has full capacity while the jobs are processed. We also assume that the reference tool (the tool in slot #1) is fixed when a batch of jobs has to be processed because more processing time is needed when the reference tool is changed than when other tools are changed. A tool that will be placed in slot #1 is chosen by comparing the frequency of its use in the total job process. Therefore, at any instant, tool changes occur less than **C-1**. A job sequence is a permutation of (1,2, ... , N), or equivalently, of the columns of **A**. Given a job sequence, a tool change counts every time the automatic tool changer removes a tool, replaces it in its slot in the tool magazine, automatically selects the next needed tool from the tool magazine, and installs it in the quill.

The objective of this section is to determine the optimum sequence in which to process any individual job and correspondingly to determine the set of tools that must be placed on the machine to minimize the total number of changes. Minimizing the number of tool changes is equivalent to minimizing the total amount of time required to manufacture each specific item. An item is a piece of material that is fastened to the milling machine table, and which is cut, milled, drilled, etc., to produce something that either is removed from the milling machine because it is complete, or which must be removed from the milling machine for further machining operations at another work station. In addition, we need several decision variables to specify the model. Let x_{jn} = 1 if job j is at the nth position in the sequence and x_{jn} = 0 otherwise. The moment in time after processing the nth position job, but before any tools are changed, is called instant n. Let u_{in} = 1 if tool i is on the magazine at instant n and u_{in} = 0 otherwise. To model this new problem based on Tang and Denardo [7,8], Daskin, Jones, and Lowe [11], and Bard [12], we implemented our initial study as follows:

1. Let us first consider a case in which jobs (= k) require more than C tools.

   Let k be the number of jobs that require more than C tools and r be the number of subjobs of each job k for k = 1,2, ..., K; r = 1,2, ..., R. Each job k can be divided by subjobs s_{kr} in which $s_{k.}$ are processed in an independent sequence, and each job s_{kr} requires fewer than C tools including the reference tool. This implies that each job k has its own independent sequence that is r factorial for r = 1,2, ..., R, where r is the total number of subjobs to be processed in a fixed sequence. If j_3 (k = 1) requires more than C tools in given jobs j_1,j_2, ..., j_N, then we need to divide job j_3 and select fixed sequences of the job j_3 and combine one of these job sequences with initial jobs j_1,j_2, ..., j_N. The optimal sequence s_{11},s_{12}, ..., s_{1r} for job 3 can be determined by minimizing the number of tool changes of a given jobs j_1, j_2, (s_{11},s_{12}, ..., s_{1r}), ..., j_N, where s_{11}, s_{12}, ..., s_{1r} is a fixed sequence.

2. Now let us consider a case in which the reference tool (the tool in slot #1) needs replacement.

   We fix a tool in slot #1 when a batch of jobs has to be processed as a reference tool because more processing time is needed when the reference tool is changed than when other tools are changed. Two constraints are considered:

$$\begin{aligned} u_{1n} &= 1, n = 1, \ldots, N+H-K \\ \sum_{i=2}^{M} u_{in} &= C-1, n = 1, \ldots, N+H-K \end{aligned}$$

   where H = the total number of subjobs in $\sum_{r=1}^{R} s_{kr}$, k = 1,2, ..., K.

Mathematically, the overall problem can be formulated as a non-linear integer program that minimizes the number of tool changes as follows:

$$Min \sum_{n=1}^{N+H-K} \sum_{i=2}^{M} u_{in}(1-u_{i,n-1})$$

subject to

$$\begin{aligned} \sum_{n=1}^{N+H-K} x_{jn} &= 1, j = 1, \ldots, N+H-K && (1) \\ \sum_{j=1}^{N+H-K} x_{jn} &= 1, n = 1, \ldots, N+H-K && (2) \\ u_{1n} &= 1, n = 1, \ldots, N+H-K && (3) \\ \sum_{i=2}^{M} u_{in} &= C-1, n = 1, \ldots, N+H-K && (4) \end{aligned}$$

$$\sum_{i=2}^{M} s_{ikr} \leq C-1, k=1,\ldots,K, r=1,\ldots,R \qquad (5)$$

$$a_{ij}x_{jn} \leq u_{in}, \forall i,j,n \qquad (6)$$

$$s_{ikr}, a_{ij}, x_{jn}, u_{in} = 0,1, \forall i,j,n \qquad (7)$$

where H = the total number of subjobs in $\sum_{r=1}^{R} s_{kr}$, k = 1,2, ...,K, u_{i0} (i=1, ..., M) are given initial conditions. These will be assumed as $u_{i0} = 1$ for all i, and $s_{ikr} = 1$, if tool i is required for the job s_{kr}, $s_{ikr} = 0$; otherwise, for i = 2,3, ..., M, k = 1,2, ..., K, r = 1,2, ..., R.

Constraints (1) and (2) above ensure that each job is assigned to exactly one instant. Constraints (3) and (4) indicate that one reference tool is fixed and ensure that no more than C-1 tools can be placed on the flexible machine. Constraint (5) indicates each subjob s_{kr} requires no more than C tools including the reference tool. Constraint (6) assures that if job j requires tool i and is assigned to nth position, then tool i will be on the magazine at instant n. Finally, constraint (7) denotes the integrality requirement for each x_{jn}, u_{in}.

The solutions obtained from this model give an optimal job sequence and a way to minimize the number of tool changes and can be equalized in a number of ways on the basis of the suggestions of Tang and Denardo [7,8]. Unfortunately, this model also have tremendous computational complexities that are undesirable. In real manufacturing environments, this model is neither efficient nor useful, even for relatively small problems. For this reason, we wish to discuss and develop one good heuristic approach for job sequencing problem.

THE TOOL-CHANGING PROBLEM

The tool-changing problem is naturally composed of two issues: (1)sequencing: find an (optimal) job sequence and (2)tool changing: determine which tools should be changed on the tool magazine at each moment in order to minimize the total number of changes for a given job sequence.

Tang and Denardo [7], and Bard [12] proved that the tool changing problem can be solved in O(MN) operations by applying a KTNS policy as follows:

1. At any instant, insert the tools that are required by the next job.

2. If tools are inserted, the tools that are not removed are needed the soonest.

The objective of this problem is to determine the set of tools to be placed on the machine so that the total number of tool changes is minimized. A KTNS policy is optimal for the tool- changing problem based on three theorems: (1)Each KTNS policy minimizes the total number of tool changes, (2)every KTNS policy is an optimal tool changing policy, and (3)if the job sequence is specified, then the KTNS policy is optimal for the tool-changing problem from Tang and Denardo [7,8], and Bard [12]. Also many studies show that any KTNS policy is found to be optimal for the tool-changing problem that is a subproblem for each job sequence.

The job-sequencing problem then is to find the optimal job sequence, that is, the sequence where the tool changing problem is solved for every job sequence. Clearly then the optimal job sequence can be determined by solving the tool-changing problem for each individual job sequence. However, this approach is undesirable, as mentioned in Section 2, because the total number of job sequences is N!. This is an extremely complicated computational problem. Therefore, we need to develop a heuristic approach for finding the local optimal job sequence, one that both minimizes the number of tool changes and simplifies the necessary computations.

Job Sequencing

We first assume that the reference tool (the tool in slot #1) is fixed when a batch of jobs has to be processed. The tool which will be in slot #1 is chosen by comparing the frequency of its use in the total job process. Subsequently, we consider jobs requiring more than C tools. These jobs can be divided by subjobs s_{11},s_{12}, $\ldots$, s_{1r}, s_{21},s_{22}, $\ldots$, s_{2r}, $\ldots$, s_{k1},s_{k2}, $\ldots$, s_{kr}. Each subjob $s_{k.}$ will be processed in a fixed sequence Ω_k, where Ω_k is a sequence of $s_{k.}$ for k = 1,2, ..., K.

The main objective of this section is to determine how to control jobs that require more than C tools. To find a optimal fixed sequence s_{kr}, let us first consider one constraint and several variables as follows:

- choose one tool that is required more than any other tool for all of the jobs and set up this tool as the reference tool.
- y_{jk} = 1 if job j is fixed with s_{kr} and y_{jk} = 0, otherwise.
- a_{ij} = 1, if tool i is required for the job j, a_{ij} = 0; otherwise, for i=2,3, ..., M and the reference tool is fixed.
- s_{ikr} = 1, if tool i is required for the job s_{kr}, s_{ikr} = 0, otherwise; for i = 2,3, ..., M, k = 1,2, ..., K, r = 1,2, ..., R.

The value s_{ikr} is obtained by permutating ${}_tPc({}_tP_{c-1})$ when the reference tool is included (or not), where t = the total number of tools that are required for the fixed job sequence Ω_k, and C = limited capacity of the tool magazine. Mathematically, this problem can be formulated to select jobs that will minimize the number of tool changes between one job and the first subjob $s_{.1}$, and between any other other job and the last subjob $s_{.R}$ for $k = 1, 2, \ldots, K, r = 1, 2, \ldots, R$ is as follows:

$$Min \sum_{r=1}^{R} |E_j| + |F_j| \tag{8}$$

$$where \quad E_j = a_{.j} - s_{.k1} \quad and \quad F_j = a_{.j} - s_{.kR}$$

subject to

$$\sum_{k=1}^{K} y_{jk} = 1, j = 1, 2, \ldots, N-K \tag{9}$$

$$\sum_{j=1}^{N-K} y_{jk} = 2, k = 1, 2, \ldots, K \tag{10}$$

$$a_{.j} \leq C - 1 \tag{11}$$

$$s_{.kr} \leq C - 1 \tag{12}$$

$$a_{ij} = 0, 1, \forall i, j \tag{13}$$

$$s_{ikr} = 0, 1, \forall i, k, r \tag{14}$$

The parameter s_{kr} for k = 1,2, ..., K and r = 1,2, ..., R is scheduled in a fixed order immediately after one job and before any other job that satisfies the objective function of Eq. (8) based on constraints of Eqs. (9-14).

We have determined the number of jobs that will be in fixed sequences by considering the previously described two constraints; still, we have other jobs that should be arranged in a job sequence. Therefore, we examine and propose several heuristic approaches for arranging the job sequence.

We first consider two constraints as explained in Section 1, and choose the number of jobs that will be fixed. The data of our problem consist of an M $\times$ (N + H - K) tool-job matrix A and capacity C with several jobs in fixed sequences. Now we focus on solving the sequencing problem since we know that the tool-changing problem is easy to solve according to the individual job sequence. Let us examine several useful heuristics that efficiently reduce complexity (as described in Section 1) for solving the sequencing problem as follows.

1. Traveling salesman heuristics

   To study the job sequencing problem, let us define a graph D as (V,A), where V is a set of vertices and A is a set of ordered pairs of elements of V. The vertices and ordered pairs (so-called edges) represent a set of jobs and the number of tool changes incurred when this pair of jobs is processed, respectively. This graph is related to the Hamiltonian path on the graph in order to find the shortest path. This path corresponds to a minimum number of tool changes. Therefore it is not difficult to find the shortest Hamiltonian path on a graph, because finding the shortest Hamiltonian path is an NP-complete problem, equivalent to solving the traveling salesman problem (TSP) and scheduling industrial processes (SIP) [19].

   These heuristics consist of finding a shortest Hamiltonian path on the complete graph with edge lengths [20,21]. Such a problem is equivalent to solving the TSP that considers a graph D = (V,E,lb), where V is the set of jobs, E is the set of all pairs of jobs, and the length lb(i,j) of the edge {i,j } is an underestimate of the number of tool changes needed between jobs i and j when these jobs are consecutively processed in a sequence.

More precisely, lb(i,j) = max(| $T_i \cup T_j$ | - C, 0), where T_i is the set of tools required by job i (i = 1,2,..., N). Notice that, if each job requires exactly C tools, then lb(i,j) is equal to the number of tool changes between two jobs i and j in any schedule. We have several heuristic approaches for constructing a shortest TSP path in D:

- shortest edge heuristic [7] or greedy feasible [21]; complexity: O(N^2logN)
- nearest neighbor heuristic with all possible starting nodes [22,23]; complexity: O(N^3)
- farthest insertion heuristic with all possible starting nodes [22,23]; complexity: O(N^4)
- branch and bound heuristic [24]; complexity: exponential.

2. Block minimization heuristics [25]; complexity: O(N_3)

We propose a different method with respect to the TSP heuristic for a given instant of the tool changing problem. A directed graph D = (V,E,mb) is considered. The length mb(i,j) of arc (i,j) is defined by mb(i,j) = | $T_i \setminus T_j$ |, where T_i is the set of tools required by job i (i = 1,2, ..., N), and T_0 is an empty set. The length of mb(i,j) represents the number of tool changes between jobs i and j, for any sequence in which jobs i and j must be sequential.

Each TSP path of D finishing at node 0 represents a sequence of jobs, and the length of the path is an upperbound on the total number of tool changes by the sequence. In this heuristic, we propose two implemented heuristics to construct a short TSP path in D as follows:

- NN block minimization, a nearest neighbor with all possible starting nodes; complexity: O(N^3)
- FI block minimization, a farthest insertion with all possible starting nodes; complexity: O(N^4)

3. Greedy heuristics; complexity: O(MN_3)

   TSP heuristics and the block minimization heuristics do not take all job sequences into account when estimating the number of tool changes required between a pair of jobs. For instance, lb(i,j) and mb(i,j) is a lowerbound and upperbound on the number of tool changes between the two jobs i and j. If no job requires more than C/2 tools, then lb(i,j) = 0 for two jobs i and j and a random job sequence will arbitrarily be picked up on the basis of the these edgelengths lb(i,j) = 0. Therefore, we consider this an unsuitable situation and propose the following heuristic.

   - Start with the partial job sequence with job 1 σ=(1), and Q={ 2,3, ...,N}.
   - Let NC(j) be the number of tool changes of the partial sequence (σ,j) for each job j in Q.
   - Let i be a job in Q for NC(i) = {min NC(j), j $\in$ Q}; let σ = (σ,i) and Q = Q\ {i}.
   - If Q is not empty, then go to (c); otherwise stop with the complete sequence σ.

     This heuristic has O(MN^3) computational complexity and gives slightly improved performance by considering all the partial sequences.

General Procedure

Let us consider a simple example that has 10 jobs (N = 10) to be processed on a flexible machine that can hold 5 tools (C = 5). These ten jobs need a total of 12 different tools (M = 12). The tool requirement vectors are given in Table 1.

Table 1: The Tool Requirement Vectors

	job	1	2	3	4	5	6	7	8	9	10
	1	1	1	0	0	1	1	1	0	1	0
	2	0	1	0	1	0	1	0	0	0	1
	3	0	0	1	0	1	0	0	0	1	0
	4	1	1	1	0	0	0	0	0	0	1
t	5	0	0	0	0	1	1	0	1	1	0
o	6	0	0	1	1	0	0	1	0	0	1
o	7	1	0	1	0	0	0	1	0	0	1
l	8	0	0	1	1	1	0	0	0	0	0
	9	0	0	0	0	1	0	0	0	1	0
	10	1	0	1	0	0	1	0	1	0	0
	11	1	0	1	0	0	1	0	1	0	0
	12	0	0	1	1	0	0	0	0	0	0

STEP 1

In this example, tool #1 is required for six jobs in which it is the most frequently used tool. So we choose tool #1 as a reference tool and set this tool in slot #1 on the magazine. There is only one job (= job 3) that requires more than C tools, therefore s_{1r} = job 3 can be divided by s_{11},s_{12}, ..., s_{1r} based on permutation ${}_tP_C$ as described in Section 2.

We choose job 1 and have two subjobs s_{11}, s_{12}, where subjob s_1 requires tools 4,7,10, and 11 and subjob s_{12} requires tools 3,6,8, and 12 for job 3. If s_{11}, s_{12}

are scheduled in a fixed order between job 1 and job 4, then it is minimized by Equation(8). We set up a fixed subsequence (ρ ={job 1, job 3 (= s_{11}, s_{12}), and job 4} and reduce the original set of ten jobs to these eight: {ρ,2,5,6,7,8,9,10}.

STEP 2

Let us consider the next situation. For any job i, F(i) will be the set of jobs that require only a subset of the tools required by job i. More formally, we let F(i) = {j:j$\neq$ i, $A_j \leq A_i$}. In this case, no tool changes are required if all the jobs in F(i) are scheduled in any order immediately after job i. Therefore, it must be optimal to schedule the set of jobs F(i) after job i. Let us consider the tool requirement vectors given in Table 1. It can be seen that $A_9 \leq A_5$ and $A_8 \leq A_6$; hence F(5) = {9} and F(6) = {8}. In this case, no tool changes are required for job sequence (5,9) or (6,8). Based on this observation, we reduce the original set of ten jobs to these six: {ρ,2,5,6,7,10}.

STEP 3

The third step consists of finding a shortest length path using any heuristics that are introduced. In this paper, we use the Greedy heuristics procedure for finding an optimal job sequence as follows.

- In our example, seven remaining jobs are considered in this step. First, we start with job 1 as a partial job sequence σ = (2), and Q = {ρ,5,6,7,10}.

- Compute NC(j) of the partial sequence (σ,j) for each job j in Q. Choose i to be a job in Q for NC(i) = min {NC(j), j $\in$ Q}; let σ = (σ,i) and Q = Q\ {i}. Finally, we find the four shortest sequences, {2,7,10,ρ,5,6}, {2,10,7,ρ,5,6}, {2,7,10,ρ,6,5}, {2,10,7,ρ,6,5}, in Table 2.

Table 2: The number of tool changes for each job sequence

Sequencing	Job Sequence	Number of tool changes
1	2,7,10,ρ,5,6	13
2	2,10,7,ρ,5,6	13
3	2,7,10,ρ,6,5	13
4	2,10,7,ρ,6,5	13

It follows that the length of each sequence is a lower bound on the number of tool changes needed to execute the corresponding job sequence. Consequently, the length of a shortest job sequence is a lower bound on the number of tool changes incurred by the optimal job schedule.

STEP 4

In step 4, we apply the KTNS policy to find a job schedule that may produce a smaller number of tool changes than the current best job schedule. In this example, the four shortest sequences, {2,7,10,ρ,5,6}, {2,10,7,ρ,5,6}, {2,7,10,ρ,6,5}, {2,10,7,ρ,6,5}, in Table 2 can be a local optimal for the minimization of the tool changes. Therefore, we wish to suggest using one of these four job sequences to reduce the total processing time for a batch of jobs.

PETRI NET REPRESENTATION

Petri nets have been useful tools for modeling, analyzing, and evaluating the behaviors and sequence control specifications of the FMS with respect to concurrent, asynchronous, deadlock, and conflict machine actions. Theories and applications of the Petri nets have been useful tools and can be adapted to model and analyze versatile configurations of the FMS by adding suitable extensions.

In FMS, the high flexibility of control procedures is another important factor, because machines that are composed of FMS make it possible to perform many sequential and concurrent (include conflicting) operations without incurring time delays. However, traditional methods for control procedures and for developing control programs do not provide sufficient flexibility because comprehending control procedures is very complicated and difficult [15,16].

To resolve this problem, the Petri net model has been experimentally applied to a FMC to show the flexibility gained by using the model. Using the eloquent representation of the Petri net model, various types of machine and equipment in FMCs can be easily supervised and controlled, and the hours required to develop control programs can be significantly reduced.

Petri Nets

1. Ordinary Petri net

   A Petri net graph uses circles to represent places (states) and bars to represent transitions (events). Input-output relations are represented by directed arcs between places and transitions. Tokens reside at a place when it is active.

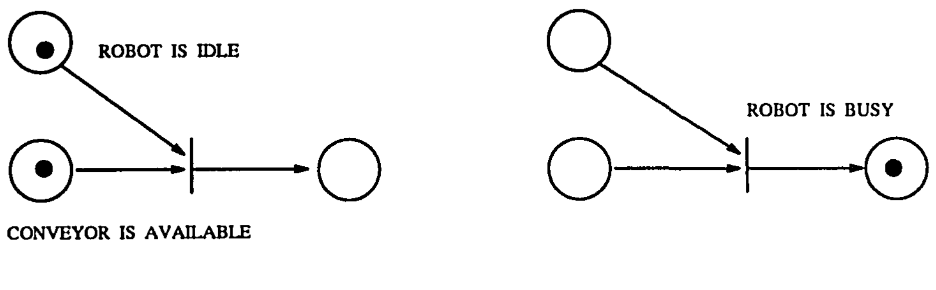
Tokens flow through the net depending on the present marking of the net. The marking of a Petri net is contained in a vector of dimension n, where n is the number of places and each value of the vector correspond to the number of tokens in the corresponding place. When there is a token in each of the input places of a transition, that transition is enabled to fire. If the weights on each of arcs between places and transitions are equal to one, then the transition fires by removing a token from each of its input places and by placing a token in each of its output places [14].

Figure 1 shows an Ordinary Petri net example. The tokens, places, and transitions correspond to the various elements found in manufacturing systems. Places usually represent resources (e.g., machine, part, and data). A token in a place indicates that the resource is available; if no token, that resource is unavailable. A place can also be used to imply that a logical condition holds. Transitions are generally used to represent the initiation or termination of an event.

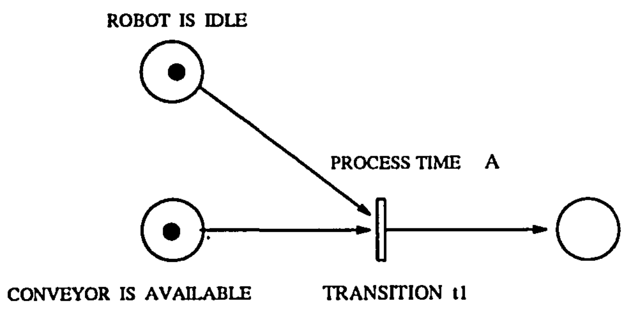
2. Marked and safe Petri nets

A Petri net containing a marking μ is a marked Petri net, defined by **M** = (P,T,I,O,μ). Marking μ of a Petri net **PN** is a function from set P to a set of non-negative integers **N**, μ: P $\longrightarrow$ **N**, where μ sets tokens to every place, $\mu_i = \mu(p_i \in \mathbf{N}$ indicates the number of tokens in place p_i). We denote a Marked Petri net (=**M**) by (**PN**, μ). We generally associated an initial marking μ_0 with a given **M**.

An important property of a Petri net model is safeness. A place in a Petri net



(a) Petri net example



(b) Timed Petri net example

Figure 1: Ordinary Petri net example

model is safe if the number of tokens in that place never exceeds one. A Petri net model is safe if all places in the Petri net model are safe.

In a safe Petri net model, each transition is generally used to represent the initiation or termination of a machine's action, sequence control specifications, and exclusion control of several machine actions. Also, the execution sequence of the machine's actions can be defined as a Petri net model structure, and sequential and concurrent processes of the sequence can be realized as token movements. However, this description needs many places and transitions to describe more complicated control specifications, which include many machine actions. A place in a Petri net model can represent whether the resource is available or not, or the condition is true or false. Similarly, a transition represents only one status corresponding to the token firing, while machine actions usually have plural statuses depending on the results of their operation. To avoid this problem, the Petri nets model includes Control Petri net (CPN) [15,16] for describing control specifications for the tool- changing procedure and the machining job in a flexible manufacturing environment.

3. Control Petri net

Control Petri net (CPN) models are introduced based on initial works [15-18] as defined by the tuples CPN = (P,T,I,O, δ, φ, η, θ, ϑ, ι, U,V,M), where U, V (system status functions) represent execution status at places and transitions, and δ, φ, η, θ, ϑ, and ι (input-output process functions) represent process status. The system status functions allow supervision of the execution status and management of the transition and place statuses, and input-output process functions are used to allow an operator direct control of token movement in the system. This is an example of modeling enhancements quickly limiting the decision and analysis attributes of Petri net models. In order to define a corresponding place and transition in a CPN and the controllable and observable process in a FMS, several functions are needed as follows.

- Definitions

  Let C be a set of control signals (c_i) and O be a set of observable signals (o_{ij}); similarly let CH be a set of checking signals (ch_i) and J be a set of judgment signals (j_i). Input-output process functions δ: $\mathbf{T} \rightarrow \mathbf{C}$ φ: $\mathbf{T} \rightarrow \mathbf{O}$ θ: $\mathbf{P} \rightarrow \mathbf{CH}$ ϑ: $\mathbf{P} \rightarrow \mathbf{J}$ η: $\mathbf{T} \rightarrow \mathbf{O}$ ι: $\mathbf{P} \rightarrow \mathbf{J}$ are defined as follows:

$$\delta(t_i) = c_i, (c_i \in \mathbf{C}, t_i \in \mathbf{T}) \tag{15}$$

$$\varphi(t_i) = o_{i1}, o_{i2}, \ldots, o_{in}, (o_{ij} \in \mathbf{O}, t_i \in \mathbf{T}) \tag{16}$$

$$\eta(t_i) = o_{ij}, (o_{ij}, \in \mathbf{O}, t_i \in \mathbf{T}) \tag{17}$$

$$\theta(p_i) = ch_i, (ch_i \in \mathbf{CH}, p_i \in \mathbf{P}) \tag{18}$$

$$\vartheta(p_i) = j_{i1}, j_{i2}, \ldots, j_{im}, (j_{ij} \in \mathbf{J}, p_i \in \mathbf{P}) \tag{19}$$

$$\iota(p_i) = j_{ij}, (j_{ij} \in \mathbf{J}, p_i \in \mathbf{P}) \tag{20}$$

- Input process function

  When a token enters into a transition t_i, a control signal c_i defined by $\delta(t_i)$ triggers a machining action. Then the token waits to fire in a transition until one of the input signals o_{ij} defined by $\varphi(t_i)$ is shown for completion of a machining action. Input signal o_{ij} defined by $\eta(t_i)$ is used for firing a transition. After detecting input signal o_{ij}, the transition can fire and the token moves to its output places.

- Output process function

  The checking signal ch_i is defined by $\theta(p_i)$ that corresponds to plural statuses on the basis of results of the machining actions in output places. By using the checking signal ch_i , the checking operation is started. Also the token waits to fire in a place until one of the input judgment signals shows completion of a checking action like the input process function. The signals j_{ij} are defined by $\vartheta(p_i)$ that corresponds to its completion of a judgment, including quality specifications. Input signal j_{ij}, defined by $\iota(p_i)$, is used for firing a place. After detecting an input signal j_{ij}, a place can fire and the token move to its output transitions.

- Process status functions

  In order to define the execution status at a transition and a place, and in order to manage the transition and place the open and close statuses and the process status functions [15] The parameters U:**P** ∈ **L**(**L** =0,1, . . ., m), V:**P** ∈ N(N =0,1) are introduced as follows:

$$U(t_i) = \begin{cases} \textit{in} & \text{action associated with } t_i \text{ is executing now} \\ \textit{out} & \text{action associated with } t_i \text{ is completed with return code } o_{in} \end{cases}$$

$$U(p_i) = \begin{cases} \textit{in} & \text{checking associated with } (p_i) \text{ is executing now} \\ \textit{out} & \text{checking associated with } (p_i) \text{ is completed} \end{cases} \tag{22}$$

$$V(t_i) = \begin{cases} \mathit{close} & t_i \text{ is closed} \\ \mathit{open} & t_i \text{ is opened} \end{cases} \tag{23}$$

$$V(p_i) = \begin{cases} \mathit{close} & p_i \text{ is closed} \\ \mathit{open} & p_i \text{ is opened} \end{cases} \tag{24}$$

When an output signal c_i defined by $\delta(t_i)$ has been put out in the transition, U(t_i) is set at *in*. When one of the input signals o_{ij} defined by $\varphi(t_i)$ is detected, U(t_i) is set at the value of *out*. If an input signal o_{ij} defined by $\eta(t_i)$ has not been detected, the value of V(t_i) is set at *0*; otherwise, V(t_i) is set at *open*. Similarly, after the token in transition is moved into its output places, if an output signal ch_i defined by $\delta(p_i)$ has been checked, then U(p_i) is set at *in*, otherwise U(p_i) is set at *out*. If an output signal j_i defined by ι_i has not yet been detected, the value of V(t_i) is set at *0*; otherwise, V(t_i) is set at *open*.

By introducing these functions, execution statues or transition operation modes can be supervised and controlled at a place and transition. In this paper, places and transitions are called CPN-transitions and CPN- places (represented by the fat box and the fat circle) since the process input-output functions and process status functions can be defined at places and transitions.

- The token firing rule in CPN-transitions and CPN-places

A token in all input CPN-places p_i of the transition $t_i \in$ T can be enabled at each marking M(p_i)=1, if and only if,

$$\begin{aligned} V(p_i) &= open, \text{ and} \\ U(p_i) &= out \end{aligned} \tag{25}$$

A token in CPN-transition $t_i \in$ T can be enabled if and only if,

$$\begin{aligned} V(t_i) &= open, \text{ and} \\ U(t_i) &= out \end{aligned} \tag{26}$$

- Other functions

  We have more complicated sequence control specifications for the machining processes such as conditional branches based on the result of a machining action and timing control. A CPN place can have several output transitions and the output transition to be fired is selected according to the result of machining actions. In addition, a time value can be assigned to the token and can be used to evaluate time factors such as production time and rate.

Idustrial Application

Flexible Manufacturing Cell An FMC that has two AGVs and a milling machine (DM4400) is shown in Figure 2. The AGVs are working to carry parts to and from stations. The milling machine (DM4400) is a sophisticated, state-of-the-art electronic and mechanical CNC machine. The DM4400 systems can be logically broken down into the following subsystems.

- Control module

  The control module, as the brains of the system, provides the interface to the machine operator and orchestrates the operation of the system.

  The system controller provides program control and machine status display and performs all the math calculations required by the rest of the system.

  The control module panel and its control module board provide the machine operator with manual spindle control and federate over ride.

- Axes subsystem

  The axes subsystem provides the actual X, Y, and Z axes table movement under command by the controller.

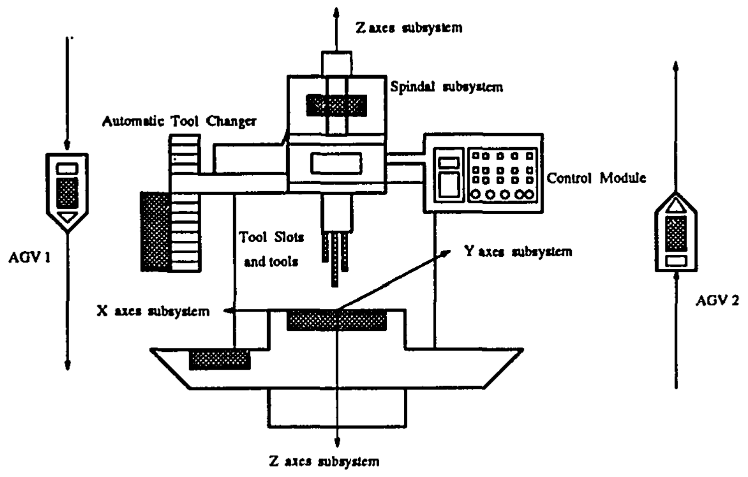
- Spindle subsystem

  The spindle subsystem, consisting of spindle controller board, spindle servo-module, DC servo motor, and spindle Hall Effect board, controls the operation of the DC spindle motor under command from the controller.

- Automated tool changer (ATC) subsystem

  The ATC subsystem provides automatic tool changing under program control.

Petri net model Figure 2 shows an FMC that has two AGVs and a milling machine with automatic tool changer. In this cell, if a batch of jobs have arrived on the AGVs at a certain moment, an optimal job sequence can be founded from the heuristic approach as described in Section 3. The cutting, milling, and drilling processes that require a number of tool changes are performed by the milling machine.



Milling Machine (DM 4400)

Figure 2: Flexible Manufacturing Cell

The entire processes for the optimal job sequence, the tool changing, and machining of the milling machine can be modeled by Ordinary, Safe and Control Petri nets. More specifically, a Safe Petri net is used to model the general approach for local optimal job sequences based on the heuristic approach. In this model, a single token represents a batch of jobs in place RQ1. From the machining procedure, Ordinary and Control Petri nets are used to model the tool-changing procedure and the machining process of the milling machine. In this model, a token in place TJ represents an individual job and a number is assigned to the token with 1 being the first order and n being the last order based on the job sequence. Therefore, the token in place TJ can be enabled to fire with respect to the number (1,2, ..., n) when the milling machine is idle.

Referring to Fig. 3, the Petri net model has a list of the places and transitions (including CPN-transition and CPN-place) along with their procedures for a optimal job sequence and tool changing procedure and the sequence control specifications for tooling processes as follows.

- Transitions

  AR: Can fire when a batch of jobs has been carried by AGV1

  JTM: Set up the job and tool matrix **A** and choose a reference tool to be placed in slot #1 on the magazine

  CON1: Can fire when a fixed subsequence for jobs requires more than C tools as described in Section 3.1

  CON2: Can fire when a batch of jobs does not have jobs requiring more than C tools

  SOJC: Can fire when a subset of jobs F(i) is checked as described in Section 3.2

  SEH & GF: Can fire when the shortest edge heuristic or greedy feasible is used

  NNH: Can fire when the nearest neighbor heuristic with all possible starting nodes is used

  FIH: Can fire when the farthest insertion heuristic with all possible starting nodes is used

  BBH: Can fire when branch and bound heuristic is used

  NNB: Can fire when the NN block minimization, a nearest neighbor with all possible starting nodes, is used

  FIB: Can fire when FI block minimization, a farthest insertion with all possible starting nodes is used

GH: Can fire when the Greedy heuristic is used

KTNSC: Can fire when there is a new job schedule that may produce a smaller number of tool changes than the current best job schedule, using KTNS policy

TC: Can fire when tool changes have been finished

CPNT: When a token enters into a transition CPNT, a control signal c_i defined by $\delta(CPNT_i)$ triggers a tooling action. Then the token waits to fire until one of the input signals o_{ij} defined by $\varphi(CPNT_i)$ is shown for completion of a tooling action. Input signal o_{ij} defined by $\eta(CPNT_i)$ is used for firing the transition. After detecting input signal o_{ij}, the transition can fire and the token moves to place CPNP. In addition, using process status functions and the firing rule as described in Section 4.1, the operator can supervise execution statues or transition operation modes at CPNT.

DP: Can fire when a batch of jobs has been finished cutting, milling, and drilling.

- Places

AGV1: AGV1 is available or not

RQ1: Request of a batch of jobs for tooling

RQ2: Enable to find jobs requiring more than C tools

SOJ: Enable to find subset of jobs F(i), being sets of jobs that require only a subset of the tools required by job i

SLP: Request to find a shortest length path of the job sequence

KTNS: Enable to find a job schedule that may produce a smaller number of tool changes than the current best job schedule using KTNS policy

TJ: Enable machining processes according to the job schedule that allows the minimum number of tool changes. The token can moved when a Soonest job based on the optimal job sequence is available.

CPNP: Using the checking signal ch_{ij} defined by $\theta(CPNP_i)$ that corresponds to plural statuses based on results of the machining processes, the operator may start the checking operation. Also the token waits to fire until one of the input judgment signals j_{ij} completes a checking action like the input process function. The signals are defined by $\vartheta(CPNP_i)$ that corresponds to its completion of a judgment, including quality specifications. Input signal j_{ij} defined by $\iota(CPNP_i)$ is used for firing a place. After detecting input signal j_{ij}, CPNP can fire and the token moves to its output transitions. In addition, using process status functions and firing rule as described in Section 4.1, the operator can supervise execution statues or transition operation modes at a CPNP.

MM: Milling machine is available or not

TS: Tools are available in storage

AGV2: AGV2 is available or not

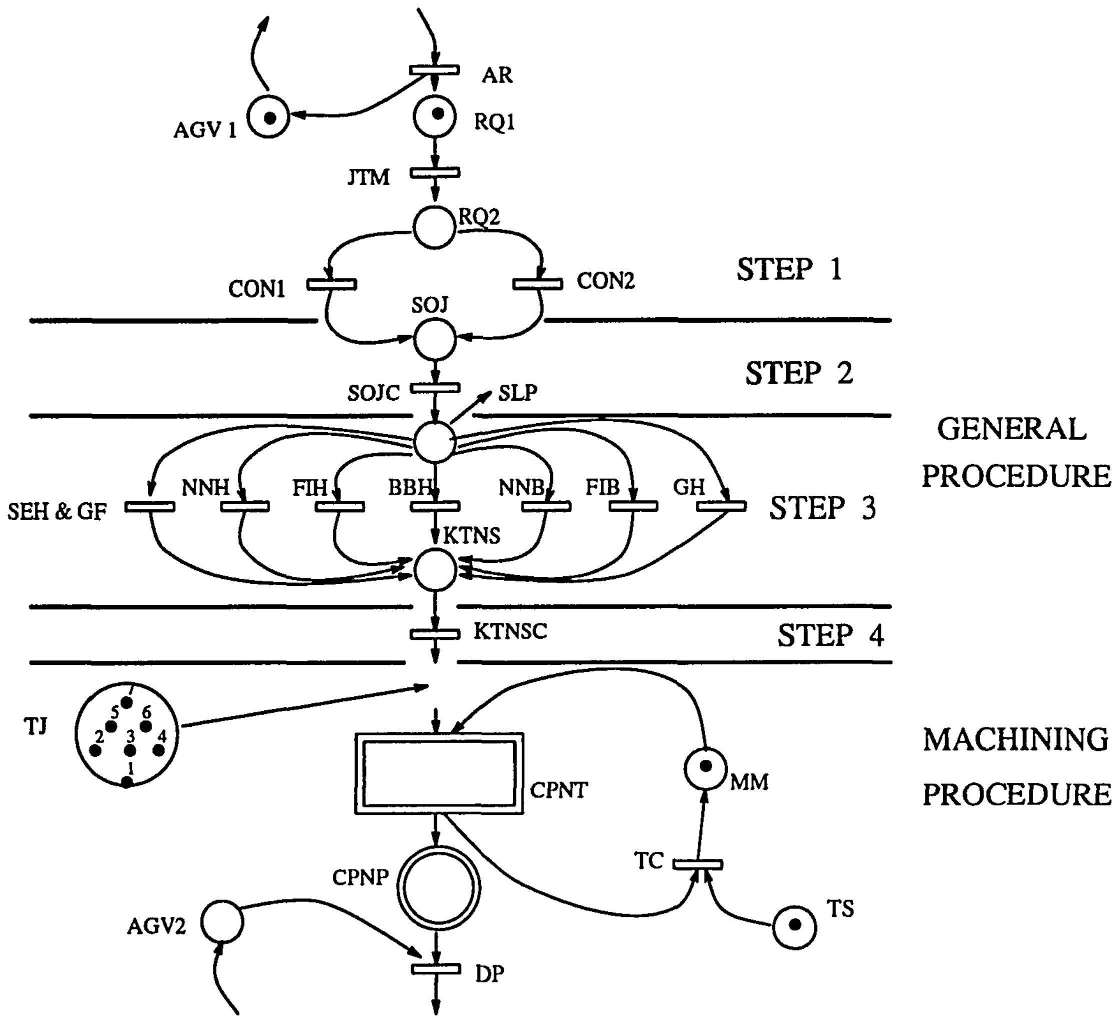


Figure 3: Petri nets model

CONCLUSION

In this paper, we examined the FMS environments including several milling machines and found two constraints that had not been considered by previous studies. We then developed a heuristic approach for finding a solution that is locally optimized with respect to job sequencing and minimizing tool changes.

In addition, we extended the petri net model for describing sequence control specifications and experimentally applied to a FMC to show the flexibility gained by using the model. Finally, we showed that the Petri net model for the local optimal job sequence based on the heuristic approach, and for the tool changing procedure and the tooling job of the milling machine. Using the eloquent representation of the Petri net mode, the FMC can be easily supervised and controlled, and the hours to develop the control program can be significantly reduced.

BIBLIOGRAPHY

[1] Stecke, Kathryn E., and Browne, J., "Variation in Flexible Manufacturing Systems According to the Relevant of Automated Materials Handling", *Material Flow* 2, 179-185,1985

[2] Mortimer, John., The FMS Report, IFS Publ. Ltd., Kempston, Bedford, U.K.

[3] Kiran, A.S., and Krason, R.J., "Automating Tooling in a Flexible Manufacturing System", *Industrial Engineering*, 52-57, April 1988.

[4] Dupont-Gatelmand, C., "A survey of Flexible Manufacturing Systems", *Journal of Manufacturing Systems* 1(1), 1-16, 1982.

[5] Stecke, Kathryn E., "Formulation and Solution of Nonlinear Integer Planning Problems for FMS", *Management Science* 29, 273-288, 1983.

[6] Stecke, Kathryn E., and Talbot, F., "Heuristic Loading Algorithms for Flexible Manufacturing Systems", *Proceeding of the 7th International Conference on Production Research* Windsor, Ontario, 1983.

[7] Tang, Christopher S., and Denardo, E.V., "Models Arising from a Flexible Manufacturing Machine, Part I : Minimization of the Number of Tool Switches", *Operations Researchs* 36(5), 767-777,1988.

[8] Tang, Christopher S., and Denardo, E.V., "Models Arising from a Flexible Manufacturing Machine, Part II : Minimization of the Number of Switching Instants", *Operations Researchs* 36(5), 778-784,1988.

[9] Roger, C., "La gestion des outils sur machines a commande numerique", *Memoire DEA de Recherche Operationnelle*, Universite Joseph Fourier, Grenoble, 1990.

[10] Mattson, R., Gecsei, J., Slutz, D.R., and Traiger, I.L., " Evaluation Techniques for Storage Hierarchies", *IBM Systems Journal* 9, 78-117, 1970.

[11] Daskin, Mark., Jones, Philip C., and Lowe, Timothy J., "Rationalizing Tool Selection in a Flexible Manufacturing System for Sheet-Metal Products", *Operations Researchs* 38, 1104-1115, 1990.

[12] Bard, Jonathan F., "A Heuristic for Minimizing the Number of Tool Switches on a Flexible Machine", *IIE TRansactions* 20, 3820391, 1988.

[13] DM4000/4400 Maintenance and Service Manual, Dyna Mechtronics, Inc, 1989.

[14] Choi, B.W., and Kuo, W., "Petri Net Extensions for Modeling and Validating Manufacturing Systems", Int. J. Prod. Res., (accepted for the publication by).

[15] Murata, Tomohiro., Komoda, Norihisa., Matsumoto, Kuniaki., and Haruna, Koichi., " A Petri Net -Based Controller for Flexible and Maintainable Sequence Control and its Applications in Factory Automation", *IEEE Transactions on Industrial Electronics*, 33(1), Feb 1986.

[16] Masuda, R., and Hasegawa, K., "Mark flow graph and its application to complex sequential control system", *Proc. of 13th Hawaii Int. Conf. on System Science*, 194-203, Jan 1980.

[17] Silva, M., and Velilla, S., "Programmable logic controller and Petri nets: A comparative study", *IFAC Software for Computer Control*, 83-88, 1982.

[18] Valette, R.et al., "Putting Petri nets to work for controlling flexible manufacturing systems", *Proc. of ISCAS' 85*, 929-932, June 1985.

[19] Roberts, F.S., Applied Combinatorics, Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

[20] Liu, C.L., Introduction to Combinatorial Mathematics, McGraw-Hill, New York, 1968.

[21] Nemhauser, G.L., and Wolsey, L.A., Integer and Combinatorial Optimization, John Wiley and Sons, New York, 1988.

[22] Golden, B.L., and Stewart, W.R., Empirical Analysis of Heuristics. In the Traveling Salesman Problem, E.L. Lawler et al., John Wiley and Sons, Chichester, 207-249, 1985.

[23] Johnson, D.S., and Papadimitriou, C.H., Computational Complexity. In the Traveling Salesman Problem, E.L. Lawler et al., John Wiley and Sons, Chichester, 37-85, 1985.

[24] Volgenant, T., and Jonker, R., "A Branch snd Bound Algorithm for the Symmetric Traveling Salesman Problem based on the 1-tree Relaxation", *European Journal of Operational Research* 9, 83-89, 1982.

[25] Kou, L.T., "Polynomial Complete Consecutive Information Retrival Problems", *SIAM Journal on Computing* 6, 67-75, 1977.

GENERAL SUMMARY AND FUTURE STUDY

This dissertation introduces the fundamental ideas and constructs of Petri net models, extends these models based on the context of a versatile manufacturing system, and applies extended Petri nets models to several manufacturing systems such an assembly cell, an Automated Palletized Conveyor System, and a tooling machine to show increased modeling power and efficient analysis methods.

The main contributions of **the this dissertation** are as follow: (1) present some studies that emphasize Petri nets theories and applications as extended research fields that provide suitable platforms in modeling, controlling, validating, and evaluating concurrent systems, information systems, and a versatile dynamic system and and manufacturing systems (2) suggest some extensions that help make Petri nets useful for modeling and analyzing discrete event systems and manufacturing systems models

(3) present validation methods for suggested models.

(4) apply extended Petri nets models to several manufacturing systems such an assembly cell, an Automated Palletized Conveyor System, and a tooling machine to show increased modeling power and efficient analysis methods.

(5) use results of a performance analysis from a deterministic and stochastic model to reorganize and re-evaluate a manufacturing system in order to increase its flexibility.

However, our works have still partial representation of the Petri net theories and applications with respect to complex flexible manufacturing systems, for example deadlock. We did not access many application areas such as controller, conveyor system, and Robots.

APPENDIX A: MODIFIED PETRI NET MODEL WRITTEN IN C

Based on our extended Petri net models, we have developed a modified deterministic algorithm written in C language, to analyze a versertile manufacturing system. The approach is based on the Petri net graph structure, firing rules, and the state of the Petri net model with process time, resource availability, multiple products, capacity, priority, and failure rate. We manually edit an input file in order to model and analyze the assembly system. By changing different variables, we create output files are generated and include the following informations:

- Incident matrix characterizing structure of the Petri net model
- State of the Petri net model. From the state variable, we know dynamic changes of the assembly system at consecutive time steps.
- Compute the maximum flowtime of the assembly system based on the directed circuits of the Petri net model. From the incident matrix, it is possible to determine all the directed circuits of the net[24].
- Compute transition firing schedules. We determine the earliest instance of time when system reached steady state and process schedules by changing the control variables in the Petri net model (e.g. resources, time,etc.), and executing the model for different system configurations.

The analysis can be divided into four main parts based on the several control variables. These are structure of the assembly system, processing time, resources, inventory, priority, and failure rate.

(1)the structure of the manufacturing system resources is fixed, but processing time is varied.

(2)the structure of the manufacturing system and processing time are fixed, but system resources are varied.

(3)the structure of the manufacturing system and processing time are fixed, but resources and inventory are varied.

(4) the structure of the manufacturing system is changed, but other variables are fixed.

```
This simulates a time Petri network composed of places and transitions.
Transitions can take time while places do not take time. Note the
following input file:

+++++++++++++++++++++++++++++++++++++++++++++++++++++++
#Transitions #Places #Outputs
Time Failure-rate #input-places i-place-0 i-place-1 ..
#output-places o-place-0 ...

Time Failure-rate #input-places i-place-0 i-place-1 ..
#output-places o-place-0 ...
...
#tokens-in-place-0 #tokens-in-place-1 ...
Destination-place
Capacity
+++++++++++++++++++++++++++++++++++++++++++++++++++++++

#transitions specifies the number of transitions.
#places specifies the number of places.
```

#outputs denotes the desired number of outputs at the detination place.

Destination-place indicates which place to be act as a destination place.

Capacity represents initial number of outputs held in the destination place.

You can manually edit a input file with your favorite editor or you can execute "mknet" to create new input file.

The included file 'phil.dat' is the network for the dining philosophers which has 10 transitions and 15 places. Note that place numbers and transition numbers start at zero. Consider the following Petri net:

```
0 ------           |
  |    |           |         2 ------
  | ** |---------->|           |    |
  |    |           |---------->|    |
  |    |     |---->|           |    |
  ------     |     |           |    |
             |                 ------
1 ------     |
  |    |     |
  | *  |------
  |    |
  |    |
  ------
```

Here is the input file for this net:

```
1 3 10
1.0 0.05 2 0 1 1 2
2 1 0
2
0
```

```
Now try: petri < thisfile

Command Line Options
---------------------
"petri" can be invoked with several options.
         % petri [-acip] [-s seed] < input-file

-c : to print cycles
-i : to print incident matrix
-p : to print transition firing table
-a : to print all of the above
-s seed

NOTE!!!
Before you compile the programs, make sure that you specify the target
operating system by setting its value to 1 in "global.h".
```

```
#ifndef GLOBAL_H
#define GLOBAL_H
#include <stdio.h>
#include <math.h>
#include "proto.h"

/*****************************************************
*   Set The Type of Target Operating System to 1     *
*****************************************************/
#define BSD   1
#define SYSV  0
#define MSDOS 0

#if SYSV || MSDOS
#define index strchr
#endif
#if BSD || SYSV
#define huge
#endif
```

```
/***************************************************/

#define TRUE 1
#define FALSE 0
#define NIL 0
#define BUSY 1
#define FREE 0
#define BEGINFIRE 1
#define ENDFIRE 2
#define YES 1

#define MAX_TRANS 100
#define MAX_PLACES 200
#define MAX_BRANCH 20

#define MAX_RANDOM_ARRAY 500
#define MAX_HEAP_SIZE    10000

extern int n_transitions, n_places;
extern int Matrix[MAX_PLACES][MAX_TRANS];
extern float trans_time[MAX_TRANS], trans_fail[MAX_TRANS];
#endif /* GLOBAL_H */



#ifdef __STDC__
# define P(s) s
#else
# define P(s) ()
#endif

/* petri.c */
int main P((int argc , char *argv []));
void Init_petrinet P((void ));
void read_net P((void ));
void print_header P((void ));
void print_incident_matrix P((void ));
void check_transition P((int event , int transition_number ));
void report_res P((void ));
void random_array P((int max , int array []));
void schedule P((int event , float inter_time , int token ));
void next_event P((int *event_ptr , int *token_ptr ));
void heap_insert P((ITEM item ));
void heap_remove P((ITEM *item_addr ));
void heap_swap P((ITEM *item1 , ITEM *item2 ));
float time P((void ));
int stream P((int n ));
double ranf P((void ));
int random P((int i , int n ));
void create_list P((LIST *list_ptr ));
int empty_list P((LIST *list_ptr ));
void insert_list P((int trans , float time ));

/* cycle.c */
void detect_cycle P((void ));
void init_stacks P((void ));
void find_cycle P((int trans ));
void update P((int trans ));
void place_cycle P((void ));
int is_dup P((int *cycle_p , int length ));
int is_same P((int *cycle_p , int who ));
void permute P((int *cycle_p , int length , int index ));
void print_cycles P((void ));

/* getopt.c */
int getopt P((int argc , char **argv , char *opts ));

#undef P




#include "global.h"

#define MAX_CYCLE 100
#define MAX_SIZE 200

#define IS_TRANS(x) (((x) < n_transitions) ? 1 : 0)

struct p_cycles {
```

```
int             count;
struct n_cycles {
int            *node[MAX_CYCLE];
int             length[MAX_CYCLE];
float           times[MAX_CYCLE];
float           success[MAX_CYCLE];
}               cycle;
}               cycles;

struct stack {
int             item[MAX_SIZE];
int             top;
}               queue, path, counter;

int             Mark[MAX_PLACES + MAX_TRANS];

#define push(stack,x) {stack.item[stack.top++] = (x);}
#define pop(stack)    {stack.top--;}
#define stop(stack)    (stack.item[stack.top-1])

/* Find out all cycles in the petri-net */
void
detect_cycle()
{
int             i, j;

cycles.count = 0;
for (i = 0; i < n_transitions; i++) {
for (j = 0; j < (n_transitions + n_places); j++)
Mark[j] = 0;
init_stacks();
push(queue, i);
find_cycle(i);
}
printf("\n\n");
}

/* Initialize various stacks */
void
init_stacks()
```

```
{
queue.top = path.top = counter.top = 0;
}

/* Find cycles starting at a given transition */
void
find_cycle(trans)
int             trans;
{
int             i, j, k, branch, temp;

while (queue.top != 0) {
push(path, stop(queue));
pop(queue);
if (Mark[stop(path)])
update(trans);
else {
Mark[stop(path)] = 1;
if (IS_TRANS(stop(path))) {
branch = 0;
for (j = 0; j < n_places; j++)
if (Matrix[j][stop(path)] == 1) {
push(queue, j + n_transitions);
branch++;
}
push(counter, branch);
} else {
branch = 0;
for (i = 0; i < n_transitions; i++)
if (Matrix[stop(path) - n_transitions][i] == -1) {
push(queue, i);
branch++;
}
if (branch != 0) {
push(counter, branch);
} else {
update(-1);
}
}
} /* else */
} /* while */
}

void
update(trans)
int             trans;
{
int             i = 0, temp;

if (stop(path) == trans) {
place_cycle();
pop(path);
} else {
pop(path);
}
while (counter.top != 0) {
temp = stop(counter);
if (--temp == 0) {
Mark[stop(path)] = 0;
pop(path);
pop(counter);
} else {
pop(counter);
push(counter, temp);
break;
}
}
}

void
place_cycle()
{
int             i, j, k;
int             length;
int            *t_cycle;

length = path.top - 1;
t_cycle = (int *) malloc(sizeof(int) * length);
i = 0;
while (i < path.top - 1) {
```

```
t_cycle[i] = path.item[i];
i++;
}
if (!is_dup(t_cycle, length)) {
cycles.count++;
cycles.cycle.length[cycles.count - 1] = length;
i = 0;
while (i < path.top - 1) {
t_cycle[i] = path.item[i];
i++;
}
cycles.cycle.node[cycles.count - 1] = t_cycle;
cycles.cycle.times[cycles.count - 1] = 0;
cycles.cycle.success[cycles.count - 1] = 1;
for (k = 0; k < cycles.cycle.length[cycles.count - 1]; k++) {
cycles.cycle.times[cycles.count - 1] += (
    IS_TRANS(cycles.cycle.node[cycles.count - 1][k])
 ?
  trans_time[cycles.cycle.node[cycles.count - 1][k]]
 : 0);
cycles.cycle.success[cycles.count - 1] *= (
    IS_TRANS(cycles.cycle.node[cycles.count - 1][k])
   ?
   1 - trans_fail[cycles.cycle.node[cycles.count - 1][k]]
   : 1);
}
}
}

int
is_dup(cycle_p, length)
int             *cycle_p, length;
{
int              i, j, index;
int              first_node;
int              duplicated = 0;

for (i = 0; i < cycles.count; i++) {
if (length != cycles.cycle.length[i])
continue;
```

```
first_node = cycles.cycle.node[i][0];
index = 0;
for (j = 0; j < cycles.cycle.length[i]; j++) {
if (first_node == cycle_p[j])
break;
index++;
}
if (index == length)
continue;
permute(cycle_p, length, index);
if (is_same(cycle_p, i)) {
duplicated = 1;
break;
}
}
return (duplicated);
}

int
is_same(cycle_p, who)
int             *cycle_p, who;
{
int              i;
for (i = 0; i < cycles.cycle.length[who]; i++) {
if (cycle_p[i] != cycles.cycle.node[who][i])
return (0);
}
return (1);
}

void
permute(cycle_p, length, index)
int             *cycle_p, length, index;
{
int              i, j, temp;
for (j = 0; j < index; j++) {
temp = cycle_p[0];
for (i = 0; i < length - 1; i++) {
cycle_p[i] = cycle_p[i + 1];
}
```

```
cycle_p[length - 1] = temp;
}
}

void
print_cycles()
{
register int    i, j;
int             l_cycles[20], l_count = 1;
int             longest_cycle = 0;
float           temp;

printf("<<      Cycles for the Simulated Petri-Net      >>\n");
printf("-------------------------------------------------------------\n");
printf("Cycle : Length : Times : Success | Transition-Place Sequences\n");
printf("-------------------------------------------------------------\n");
l_cycles[0] = 0;
for (j = 0; j < cycles.count; j++) {
printf("%5d : %6d : %5.1f : %6.5f : ", j, cycles.cycle.length[j],
       cycles.cycle.times[j], cycles.cycle.success[j]);
temp = cycles.cycle.times[j] - cycles.cycle.times[longest_cycle];
if (temp > 0) {
l_cycles[0] = longest_cycle = j;
l_count = 1;
} else if (temp < 0);
else {
if (j != 0)
l_cycles[l_count++] = j;
}
for (i = 0; i < cycles.cycle.length[j]; i++) {
printf(IS_TRANS(cycles.cycle.node[j][i])
       ? " T%d " : " P%d ",
       IS_TRANS(cycles.cycle.node[j][i])
       ? cycles.cycle.node[j][i] :
       cycles.cycle.node[j][i] - n_transitions);
}
printf("\n");
}
printf("-------------------------------------------------------------\n");
printf("There are %d cycles found in the petri-net.\n", cycles.count);
printf("Longest Cycles (Critical Paths) ==>> ");
for (i = 0; i < l_count - 1; i++)
printf("%2d, ", l_cycles[i]);
printf("%2d\n", l_cycles[l_count - 1]);
printf("-------------------------------------------------------------\n");
}
```

```
#include "global.h"
#ifndef NULL
#define NULL 0
#endif
#ifndef EOF
#define EOF (-1)
#endif
#define ERR(s, c) if(opterr){\
extern write();\
char errbuf[2];\
errbuf[0] = c; errbuf[1] = '\n';\
(void) write(2, argv[0], (unsigned)strlen(argv[0]));\
(void) write(2, s, (unsigned)strlen(s));\
(void) write(2, errbuf, 2);}

extern int      strcmp();
extern char    *index();

int             opterr = 1;
int             optind = 1;
int             optopt;
char           *optarg;

int
getopt(argc, argv, opts)
int             argc;
char          **argv, *opts;
{
static int      sp = 1;
register int    c;
register char  *cp;

if (sp == 1)
if (optind >= argc ||
    argv[optind][0] != '-' || argv[optind][1] == '\0')
return (EOF);
else if (strcmp(argv[optind], "--") == NULL) {
optind++;
return (EOF);
}
optopt = c = argv[optind][sp];
if (c == ':' || (cp = index(opts, c)) == NULL) {
ERR(": illegal option -- ", c);
if (argv[optind][++sp] == '\0') {
optind++;
sp = 1;
}
return ('?');
}
if (*++cp == ':') {
if (argv[optind][sp + 1] != '\0')
optarg = &argv[optind++][sp + 1];
else if (++optind >= argc) {
ERR(": option requires an argument -- ", c);
sp = 1;
return ('?');
} else
optarg = argv[optind++];
sp = 1;
} else {
if (argv[optind][++sp] == '\0') {
sp = 1;
optind++;
}
optarg = NULL;
}
return (c);
}
```

```
/**********************************************************
     GENERAL PURPOSE TIMED PETRI-NET SIMULATOR

     This program simulates Timed-Petri Net. The scheduling
     and heap management routines are based on SIMPACK
     petri-net simulator developed by Paul A. Fishwick.
**********************************************************/
#include "global.h"

/* Incident Matrix */
int             Matrix[MAX_PLACES][MAX_TRANS];

/*
 * t_in[i][0] and t_out[j][0] contains # of input and # of output places,
 * respectively. t_in[i][k] (where k != 0) has input place number.
 * t_out[j][l] (where l != 0) has output place number.
 */
int             t_in[MAX_TRANS][MAX_BRANCH], t_out[MAX_TRANS][MAX_BRANCH];
int             tr_status[MAX_TRANS]; /* transition status */
float           trans_time[MAX_TRANS]; /* transition time */
float           trans_fail[MAX_TRANS]; /* failure rate    */
int             p[MAX_PLACES];
int             n_transitions; /* # transitions   */
int             n_places; /* # places */

int             capacity; /* Initial # outputs placed at destination
 * place */
int             outputs; /* Desired # of total outputs at dest. place */
int             threshold; /**/
int             dest_place; /* Destination place number */
int             dest_count; /* Current # of outputs produced */
int             finish; /* Flag for program termination */
float  start_time = 0.0;     /* initial simulation time */
float           current_time = 0.0, last_event_time;

typedef struct node {
float           time;
struct node    *next;
}               NODE;
typedef struct {
NODE           *front, *rear;
}               LIST;
LIST            trans_list[MAX_TRANS]; /* Transition firing time table */

typedef struct {
float           time;
int             event;
int             token;
}               ITEM;
ITEM huge       heap[MAX_HEAP_SIZE], an_item;
int             heap_count;
int             array[MAX_RANDOM_ARRAY];
int             events;

/* Declarations for random distribution sampling */

#define then

#define A 16807L /* multiplier (7**5) for 'ranf' */
#define M 2147483647L /* modulus (2**31-1) for 'ranf' */

static long     In[16] = {0L, /* seeds for streams 1 thru 15  */
1973272912L, 747177549L, 20464843L, 640830765L, 1098742207L,
  78126602L, 84743774L, 831312807L, 124667236L, 1172177002L,
    1124933064L, 1223960546L, 1878892440L, 1449793615L, 553303732L};

static int      strm = 1; /* index of current stream */
static int      rn_stream = 1;

#include "proto.h"

extern char    *optarg;
extern int      optind;
int             aflag, cflag, iflag, pflag;

main(argc, argv)
int             argc;
```

```
char            *argv[];
{
int              event, transition_number;
int              c;
register int     i;
static char      options[] = "acips:t:";

/* process command line arguments */
while ((c = getopt(argc, argv, options)) != EOF) {
switch (c) {
case 'a':
aflag++;
break;
case 'c':
cflag++;
break;
case 'i':
iflag++;
break;
case 'p':
pflag++;
break;
case 't':
current_time = atof(optarg);
start_time = current_time;
break;
case 's':
rn_stream = atoi(optarg);
if (rn_stream < 0 || rn_stream > 15) {
fprintf(stderr, "seed must be in the range 0 <= seed <=15\n");
exit(1);
}
break;
case '?':
fprintf(stderr, "Invalid option\n");
fprintf(stderr, " -c : to print cycles\n");
fprintf(stderr, " -i : to print incident matrix\n");
fprintf(stderr, " -p : to print transition firing table\n");
fprintf(stderr, " -a : to print all of the above\n");
fprintf(stderr, " -t start_time\n");
fprintf(stderr, " -s seed\n");
exit(1);
}
}
printf("*****************************************************\n");
printf("*         WELCOME TO TIMED PETRI NET SIMULATOR       *\n");
printf("*****************************************************\n\n");
Init_petrinet(); /* Initialize petri-net simulator */
read_net(); /* Read network configuration and parameters */
printf("* Parameters Specified:\n");
printf("             Number of Transitions : %3d\n", n_transitions);
printf("             Number of Places      : %3d\n", n_places);
printf("             Number of Outputs     : %3d\n", outputs);
printf("             Capacity              : %3d\n", capacity);

detect_cycle(); /* detect cycles in the petri-net */

if (aflag || pflag)
print_header();

for (i = 0; i < n_transitions; i++) {
schedule(BEGINFIRE, 0.0, i);
events++;
}
while (finish != YES && events > 0) {
/* take one event from the event list and cause it to occur */
next_event(&event, &transition_number);
events--;
check_transition(event, transition_number);
}
report_res();
}

/* initialize data structures for simulation */
void
Init_petrinet()
{
heap_count = 0;
last_event_time = current_time;
/* set random number stream */
```

```
rn_stream = stream(rn_stream);
rn_stream++;
if (rn_stream > 15)
rn_stream = 1;
}

void
read_net()
{
int             i, j, number_inputs, number_outputs;

/* Read #transitions, #places, #outputs */
scanf("%d %d %d", &n_transitions, &n_places, &outputs);

/*
 * Read transition information: time-delay failure-rate #inputs i1
 * i2 i3 #outputs o1 o2 o3 ...
 */
for (i = 0; i < n_transitions; i++) {
scanf("%f %f %d", &trans_time[i], &trans_fail[i], &t_in[i][0]);
number_inputs = t_in[i][0];
for (j = 0; j < number_inputs; j++) {
scanf("%d", &t_in[i][j + 1]);
Matrix[t_in[i][j + 1]][i] = -1;
}
scanf("%d", &t_out[i][0]);
number_outputs = t_out[i][0];
for (j = 0; j < number_outputs; j++) {
scanf("%d", &t_out[i][j + 1]);
Matrix[t_out[i][j + 1]][i] = 1;
}
}

/* Read place information */
for (i = 0; i < n_places; i++)
scanf("%d", &p[i]);
/* Read detination place and its capacity*/
scanf("%d %d", &dest_place, &capacity);
dest_count = capacity;
```

```
/* Initialize transition time table */
for (i = 0; i < n_transitions; i++)
create_list(&trans_list[i]);
}

/* Print Table Header */
void
print_header()
{
register int    i;

printf("<<   Transition Status Table    >>\n");
for (i = 0; i < (11 + n_places * 3); i++)
printf("-");
printf("\n");
printf("TIME   ");
printf("TRS ");
for (i = 0; i < n_places; i++)
printf("%2d ", i);
printf("\n");
for (i = 0; i < (11 + n_places * 3); i++)
printf("-");
printf("\n");
/* print out initial place array */
printf("%6.2f ", time());
printf("--: ");
for (i = 0; i < n_places; i++)
printf("%2d ", p[i]);
printf("\n");
}

void
print_incident_matrix()
{
register int    i, j;
char            buf[10];

printf("-------------------------------------------------\n");
printf("<< Incident Matrix for the Simulated Petri-Net >>\n");
printf("-------------------------------------------------\n\n");
printf("     ");
for (i = 0; i < n_transitions; i++) {
sprintf(buf, "T%d", i);
printf("%4s", buf);
}
printf("\n     ");
for (i = 0; i < n_transitions; i++)
printf("----");
printf("\n");
for (j = 0; j < n_places; j++) {
sprintf(buf, "P%d", j);
printf("%5s:", buf);
for (i = 0; i < n_transitions; i++)
printf("%3d ", Matrix[j][i]);
printf("\n");
}
printf("\n\n");
}

void
check_transition(event, transition_number)
/*
 * Check a transition for firing. If the transition is not already busy then
 * fire it as long as at least one token exists in each input place for that
 * transition.
 */
int             event, transition_number;
{
int             input_places, output_places, i, fire, tokens, number;

switch (event) {
case BEGINFIRE: /* check transition for firing */
input_places = t_in[transition_number][0];
fire = TRUE;
for (i = 0; i < input_places; i++) {
tokens = p[t_in[transition_number][i + 1]];
fire = fire && (tokens > 0);
}
if ((tr_status[transition_number] == FREE) && fire) {
/* delete one token from each input place */
for (i = 0; i < input_places; i++)
p[t_in[transition_number][i + 1]] -= 1;
tr_status[transition_number] = BUSY;
schedule(ENDFIRE, trans_time[transition_number], transition_number);
events++;
} /* end if */
break;
case ENDFIRE: /* end of transition fire */
tr_status[transition_number] = FREE;
/* add one token to each output place */
input_places = t_in[transition_number][0];
output_places = t_out[transition_number][0];
for (i = 0; i < output_places; i++) {
p[t_out[transition_number][i + 1]] += 1;
if (t_out[transition_number][i + 1] == dest_place)
if (++dest_count == outputs)
finish = YES;
}
insert_list(transition_number, time());
if (aflag || pflag) {
/* firing just occurred, print out the 'p'lace array */
printf("%6.2f ", time());
printf("%2d: ", transition_number);
for (i = 0; i < n_places; i++)
printf("%2d ", p[i]);
printf("\n");
}
/* sweep through all transitions once to schedule new events */
random_array(n_transitions - 1, array);
for (number = 0; number < n_transitions; number++) {
schedule(BEGINFIRE, 0.0, array[number]);
events++;
}
break;
} /* end switch */
}

void
report_res()
{
```

```
register int    i;
NODE           *q;

printf("\n\nTotal number of output at destination place = %d\n",
printf("Last event time = %6.2f\n", last_event_time);
printf("Production Rate = %6.3f\n\n", outputs / (last_event_time
printf("----------------------------------\n");
printf("<< Transitions Firing Time Table >>\n");
printf("----------------------------------\n");
for (i = 0; i < n_transitions; i++) {
printf("T%d : ", i);
q = trans_list[i].front;
while (q != NIL) {
printf("%6.1f", q->time);
q = q->next;
}
printf("\n");
}
printf("\n\n");
if (aflag || iflag)
print_incident_matrix();
if (aflag || cflag)
print_cycles();
}

void
random_array(max, array)
/*
 * take the integers between 0 and max and return a randomly sorted array
 * 'newarray' containing these integers
 */
int             max, array[];
{
int             element, i, swap;

/* initialize array to contain to 0..max */
for (i = 0; i <= max; i++)
array[i] = i;
/* rearrange array to yield a random ordering */
for (i = 0; i < max; i++) {
```

```
element = random(0, max - i);
swap = array[element];
array[element] = array[max - i];
array[max - i] = swap;
}
}

/* schedule an event */
void
schedule(event, inter_time, token)
int             event, token;
float           inter_time;
{
float           event_time;
ITEM            an_item;
int             i;

event_time = current_time + inter_time;
an_item.time = event_time;
an_item.event = event;
an_item.token = token;
heap_insert(an_item);
}

/* cause an event to occur */
void
next_event(event_ptr, token_ptr)
int            *event_ptr, *token_ptr;
{
ITEM            an_item;

heap_remove(&an_item);
current_time = an_item.time;
*event_ptr = an_item.event;
*token_ptr = an_item.token;
last_event_time = time();
}

void
heap_insert(item)
```

```
ITEM            item;
{
int             parent, child;

heap_count++;
heap[heap_count] = item;
if (heap_count > 1) {
child = heap_count;
parent = child / 2;
while ((heap[parent].time > heap[child].time) && (child > 1)) {
heap_swap(&heap[parent], &heap[child]);
child = parent;
if (child > 1)
parent = child / 2;
} /* end while */
} /* end if */
}

void
heap_remove(item_addr)
ITEM           *item_addr;
{
int             parent, child;

*item_addr = heap[1];
heap_swap(&heap[1], &heap[heap_count]);
heap_count--;
parent = 1;
while (1) {
if (2 * parent > heap_count)
goto exit;
else
child = 2 * parent;
if (child + 1 <= heap_count)
if (heap[child + 1].time < heap[child].time)
child++;
if (heap[parent].time < heap[child].time)
goto exit;
heap_swap(&heap[parent], &heap[child]);
parent = child;
} /* end while */
exit: ;
}

void
heap_swap(item1, item2)
ITEM huge      *item1;
ITEM huge      *item2;
{
ITEM            temp;

temp = *item1;
*item1 = *item2;
*item2 = temp;
}


/* provide the current simulation time */
float
time()
{
return (current_time);
}

/* select generator stream */
int
stream(n)
int             n;
{
/* set stream for 1<=n<=15, return stream for n=0 */
/* if ((n<0)||(n>15)) then error(0,"stream Argument Error"); */
if (n)
then            strm = n;
return (strm);
}

#if BSD || SYSV
double
ranf()
{
```

```
short           *p, *q, k;
long             Hi, Lo;
/* generate product using double precision simulation  (comments  */
/* refer to In's lower 16 bits as "L", its upper 16 bits as "H")  */
p = (short *) &In[strm];
Hi = *(p) * A; /* 16807*H->Hi */
*(p) = 0;
Lo = In[strm] * A; /* 16807*L->Lo */
p = (short *) &Lo;
Hi += *(p); /* add high-order bits of Lo to Hi */
q = (short *) &Hi; /* low-order bits of Hi->LO */
*(p) = *(q + 1) & 0X7FFF; /* clear sign bit */
k = *(q) << 1;
if (*(q + 1) & 0X8000)
then             k++; /* Hi bits 31-45->K */
/* form Z + K [- M] (where Z=Lo): presubtract M to avoid overflow */
Lo -= M;
Lo += k;
if (Lo < 0)
then             Lo += M;
In[strm] = Lo;
return ((double) Lo * 4.656612875E-10); /* Lo x 1/(2**31-1) */
}
#endif

#if MSDOS
double
ranf()
{
short           *p, *q, k;
long             Hi, Lo;
/* generate product using double precision simulation  (comments  */
/* refer to In's lower 16 bits as "L", its upper 16 bits as "H")  */
p = (short *) &In[strm];
Hi = *(p + 1) * A; /* 16807*H->Hi */
*(p + 1) = 0;
Lo = In[strm] * A; /* 16807*L->Lo */
p = (short *) &Lo;
Hi += *(p + 1); /* add high-order bits of Lo to Hi */
q = (short *) &Hi; /* low-order bits of Hi->LO */
```

```
*(p + 1) = *q & 0X7FFF; /* clear sign bit */
k = *(q + 1) << 1;
if (*q & 0X8000)
then            k++; /* Hi bits 31-45->K */
/* form Z + K [- M] (where Z=Lo): presubtract M to avoid overflow */
Lo -= M;
Lo += k;
if (Lo < 0)
then            Lo += M;
In[strm] = Lo;
return ((double) Lo * 4.656612875E-10); /* Lo x 1/(2**31-1) */
}
#endif

int
random(i, n)
int             i, n;
{
/* 'random' returns an integer equiprobably selected from the   */
/* set of integers i, i+1, i+2, . . , n.                         */
/* if (i>n) then error(0,"random Argument Error: i > n"); */
n -= i;
n = (n + 1.0) * ranf();
return (i + n);
}

/* Create a list structure */
void
create_list(list_ptr)
LIST           *list_ptr;
{
list_ptr->front = NIL;
list_ptr->rear = NIL;
}

/* Is the list empty? */
int
empty_list(list_ptr)
LIST           *list_ptr;
{
```

```
return (list_ptr->front == NIL);
}

/* insert an item to the list */
void
insert_list(trans, time)
int             trans;
float           time;
{
NODE           *p;
p = (NODE *) malloc(sizeof(NODE));
p->time = time;
p->next = NIL;
if (empty_list(&trans_list[trans])) {
trans_list[trans].front = p;
trans_list[trans].rear = p;
} else {
trans_list[trans].rear->next = p;
trans_list[trans].rear = p;
}
}


#include <stdio.h>
#include <ctype.h>
#include <math.h>

#define MAX_TRANS 100
#define MAX_PLACES 200
#define MAX_BRANCH 20

main()
{
char            buf[80];
FILE           *fp;
int             transitions, places, outputs, n_inputs, n_outputs;
float           trans_time, fail_rate;
register int    i, j;
int             data;
```

```
printf("Enter the name of configuration file ==> ");
gets(buf);
if ((fp = fopen(buf, "w")) == NULL) {
perror(buf);
exit(1);
}
printf("Number of transitions ? (Max. %d) ", MAX_TRANS);
gets(buf);
transitions = atoi(buf);
printf("Number of places ? (Max. %d) ", MAX_PLACES);
gets(buf);
places = atoi(buf);
printf("Total number of desired outputs ? ");
gets(buf);
outputs = atoi(buf);
fprintf(fp, "%d %d %d\n", transitions, places, outputs);
for (i = 0; i < transitions; i++) {
printf("   < Transition %d >\n", i);
printf("     Transition time ? ");
gets(buf);
sscanf(buf, "%f", &trans_time);
fprintf(fp, "%f ", trans_time);
printf("     Failure Rate ? ");
gets(buf);
sscanf(buf, "%f", &fail_rate);
fprintf(fp, "%f ", fail_rate);
printf("     Number of Inputs ? (Max. %d) ", MAX_BRANCH);
gets(buf);
n_inputs = atoi(buf);
fprintf(fp, "%d ", n_inputs);
printf("          Enter input places numbers ==> ");
for (j = 0; j < n_inputs; j++) {
scanf("%d", &data);
fprintf(fp, "%d ", data);
}
gets(buf); /* eliminate dummy newline */
printf("     Number of Outputs ? (Max. %d) ", MAX_BRANCH);
gets(buf);
n_outputs = atoi(buf);
fprintf(fp, "%d ", n_outputs);
printf("          Enter output places numbers ==> ");
for (j = 0; j < n_outputs; j++) {
scanf("%d", &data);
fprintf(fp, "%d ", data);
}
gets(buf); /* eliminate dummy newline */
fprintf(fp, "\n");
}
printf("Enter number of tokens at each place\n");
for (i = 0; i < places; i++) {
printf("     Place %d ? ", i);
scanf("%d", &data);
fprintf(fp, "%d ", data);
}
fprintf(fp, "\n");
gets(buf); /* eliminate dummy newline */
printf("Enter the destination place number ==> ");
gets(buf);
fprintf(fp, "%d\n", atoi(buf));
printf("Enter the Capacity of the destination place ==> ");
gets(buf);
fprintf(fp, "%d\n", atoi(buf));
/*
printf("Enter the Threshold Value for Success ==> ");
gets(buf);
fprintf(fp, "%f\n", atof(buf));
*/
fclose(fp);
}
```

```
4 9 10
2.000000 0.01 2 0 7 2 1 2
1.000000 0.01 2 1 3 2 4 5
6.000000 0.01 2 2 4 1 6
1.000000 0.01 2 5 6 3 3 7 8
10 0 0 2 0 0 0 5 0
8
0
```

```
10 15 10
1.0 0.0 3  0  1  2  1  10
1.0 0.0 3  2  3  4  1  11
1.0 0.0 3  4  5  6  1  12
1.0 0.0 3  6  7  8  1  13
1.0 0.0 3  8  9  0  1  14
1.0 0.0 1 10  3  0  1  2
1.0 0.0 1 11  3  2  3  4
1.0 0.0 1 12  3  4  5  6
1.0 0.0 1 13  3  6  7  8
1.0 0.0 1 14  3  8  9  0
1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
14
3

#
# Makefile for the Petri-net Simulator
# (MacroSoft C)
#
CC=cl
OBJS= petri.obj   getopt.obj     cycle.obj
CFLAGS=          -O -c
# /NOE means NO EXTernal Dictionary
# /EX  means pack EXE file
# /ST:8000 means stack size 8000 bytes
LINKFLAGS = /NOE /EX #/codeview

#################################################################
# Commands and dependencies for individual modules #
#################################################################

default: petri.exe query.exe

# default rules
.c.obj:
$(CC) $(CFLAGS) $*.c

petri.obj: petri.c global.h
$(CC) $(CFLAGS) $*.c
```

```
getopt.obj: getopt.c
$(CC) $(CFLAGS) $*.c

cycle.obj: cycle.c global.h
$(CC) $(CFLAGS) $*.c

query.obj: query.c
$(CC) $(CFLAGS) $*.c

petri.exe: $(OBJS) global.h
link $(LINKFLAGS) @linkopt.msc

query.exe: query.obj
cl $*.obj


#
# Makefile for the Petri-net Simulator
# (UNIX)
#
SRC1 = mknet.c
SRC2 = petri.c cycle.c getopt.c proto.h global.h
SRC3 = phil.dat test.dat
OBJS = petri.o cycle.o getopt.o
LIBS = -lm
all: petri mknet

mknet: mknet.c
cc -o mknet mknet.c

petri: $(OBJS) proto.h
cc -o petri $(OBJS) $(LIBS)

.c.o :
cc -O -c $<




petri+cycle+getopt
```

```
petri
nul;
```

APPENDIX B: PROGRAMMABLE LOGIC CONTROLLER(PLC) AND RELAY LADDER LOGIC FOR THE APCS

In this section, a Programmable Logic Controller(PLC) and Relay Ladder Logic for the APCS, that was examined and developed in Part II, are introduced. The PLC is generally performs two major roles concerned with programming and on-line control of the process. In this section, we introduced a PLC that is a Modular Programmable Controller (MPC), and a subsystem of the APCS as shown in Figure 1. The MPC has a main power switch, a start/stop button, and a count start/stop button, and a maximum thirty-two input or output points. Up to this time, thirteen input and twelve output points are used to control the APCS, and these are related to input signals called by seven sensors and output behaviors by sixteen cylinders as follows:

- Input Points

  #1:start button, #2:stop button, #3:sensor1, #4:sensor4, #5:sensor5, #6:sensor6, #7:sensor7, #8:sensor2, #9:count start button, #10:count stop button, #11:counter release lane1, #12:counter release lane2, and #13:counter release lane3.

- Output points

  #17:cylinder1, #18:cylinder2, #19:cylinder3 and cylinder13, #20:cylinder4 and cylinder14, #21:cylinder5 and cylinder15, #22:cylinder6 and cylinder16, #23:cylinder7, #24:cylinder8, #25:cylinder9, #26:cylinder10, #27:cylinder11, and #28:cylinder12.

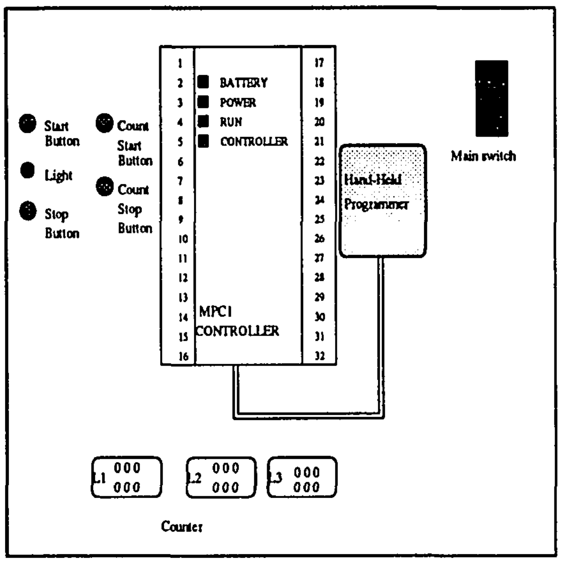


Figure 1: PLC Control Panel.

Also, the MPC has a hand-held programmer which is the communication interface with the APCS. The hand-held programmer is used to create a program (relay ladder logic) of instructions, change or correct existing programs, read a program from the MPC user memory or the program storage cartridge, load a program to the MPC user memory or the program storage cartridge, and monitor and control running programs.

A general MPC hand-held programmer translates a program written in a high level language into relay ladder logic that can be executed by the PLC. The hand-held programmer generates an internal control coil or output, input contact or relay coil, a function such as a timer, up counter, down counter, pointer register, shift register, or clear register, and mathematical operations such as addition, subtraction, less than, equal to, greater than, and not equal to by using several conventions for programming relay ladder logic.

One difficulty in using a PLC is in its programmable nature. This is an important technical aspect for the logic design and realization of the APCS. The programming of the Relay Ladder Logic (RLL) used in a MPC is important for designing and controlling the APCS. A ladder diagram graphically represents a system of push button, limit switches, sensors, contractors, solenoids and other electrical elements. The proper RLL diagrams processed by MPC will be developed based on part separation area, accumulation area, staging area, and pallet control area of the APCS.

In addition to establishing thirty-seven input and twenty-seven output points using the conventions provided, twenty-seven wires and eight registers are used to develop relay ladder logic for the APCS. Several conventions which are introduced, are helpful in documenting the ladder logic diagram for the APCS. The conventions help in understanding how to convert conventional ladder diagrams into a logical sequence of parallel input and output connections, and finally, a Petri net controller of the APCS. From the relay ladder logic diagrams, many derive a PLC programming language whose principal value is familiarity to the traditional logician who has just discovered programmable logic controllers. The constituent of the relay ladder logic diagrams are five in number (normally open/closed relay, opening of the parallel branch, closure of parallel branch, and assignment of result to an intermediate variable or to an output). The basic logic functions such as store (loading of an accumulator with the value of a variable), and , or, not (inverse), out (activation of an output or assignment to an internal bit) are obtained by a suitable assemblage of these constituents and corresponds to the notion of instruction.

Based on these explanations, we developed relay ladder logic for the APCS in following ways:

- Accumulation Area

  Figure 2 shows RLL digram for count and Figure 3 shows for release parts in accumulation area.

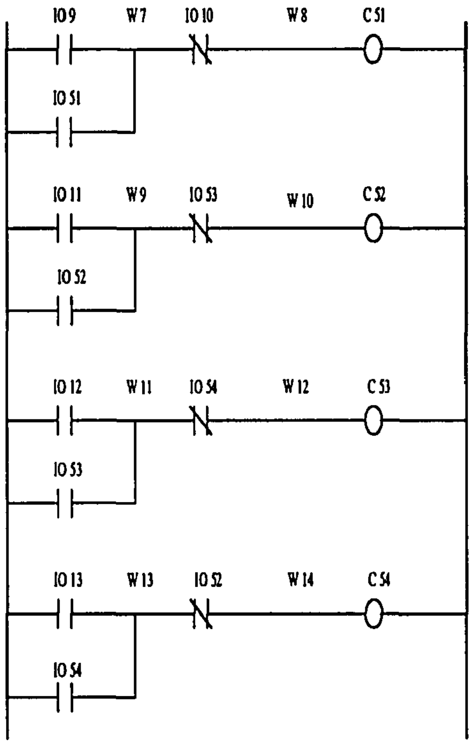


Figure 2: RLL diagram for count in accumulation area

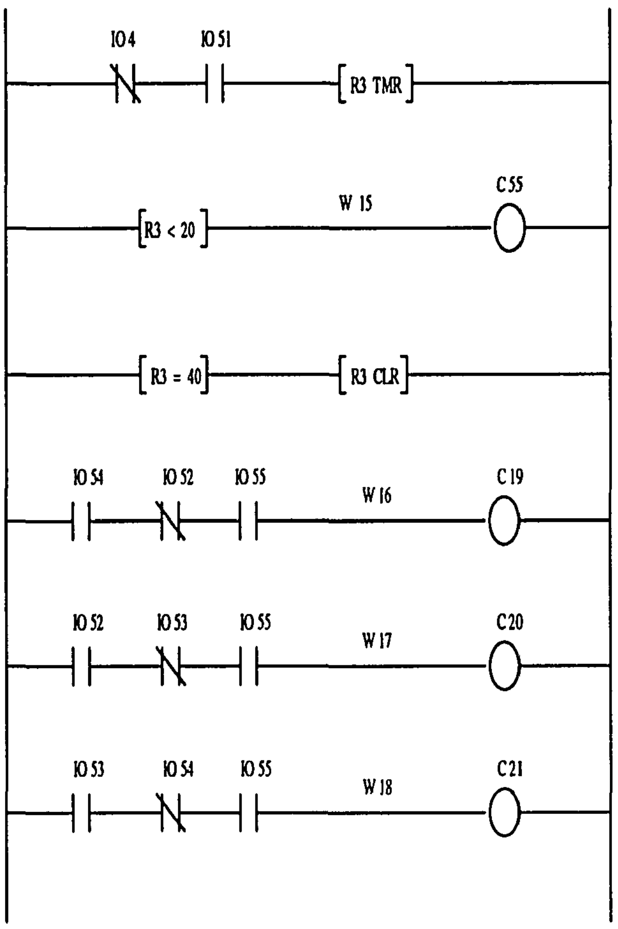


Figure 3: RLL diagram for material release parts

- Staging Area

Figure 4 shows RLL digram for material release

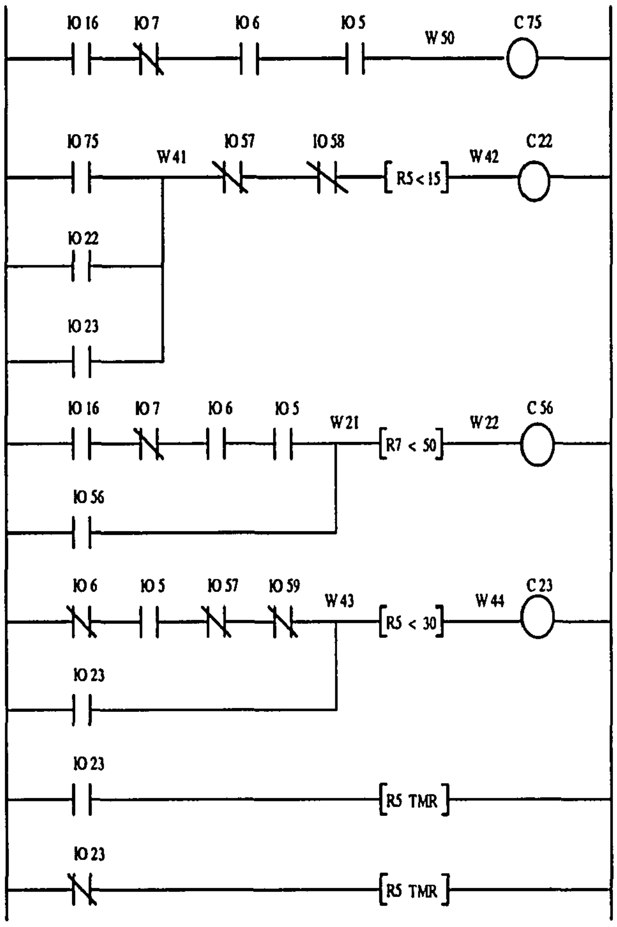


Figure 4: RLL diagram for material release

- Pallet Control Area

Figure 5 and 6 shows RLL digram for pallet control and part pick-up and return

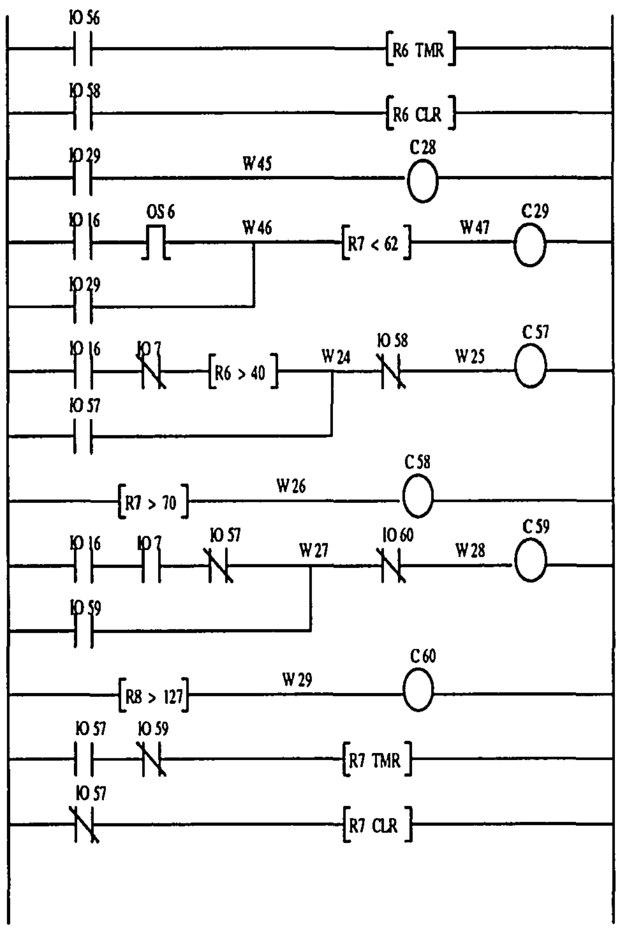


Figure 5: RLL diagram for pallet control

- Part pick-up and Return Area

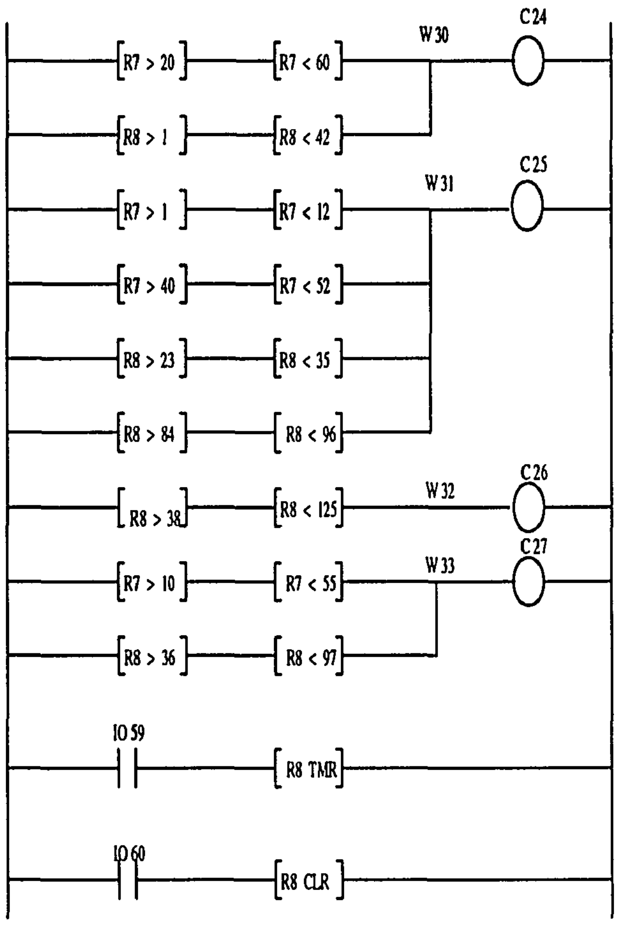


Figure 6: RLL diagram for count

- Part Separation Area

Figure 7 and 8 shows RLL digram for open power and gate1, and gate2 respectively.

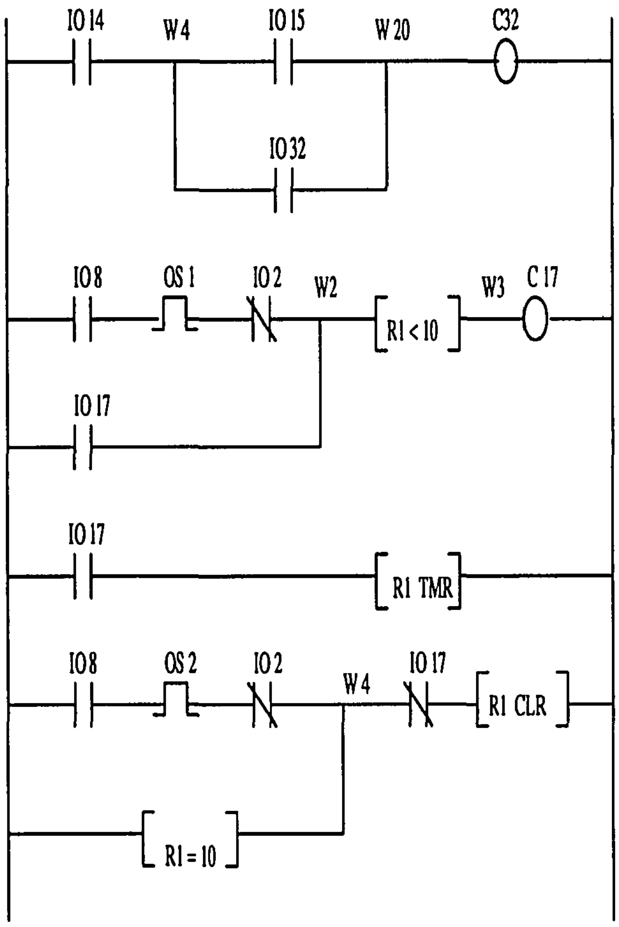


Figure 7: RLL diagram for open power and gate1

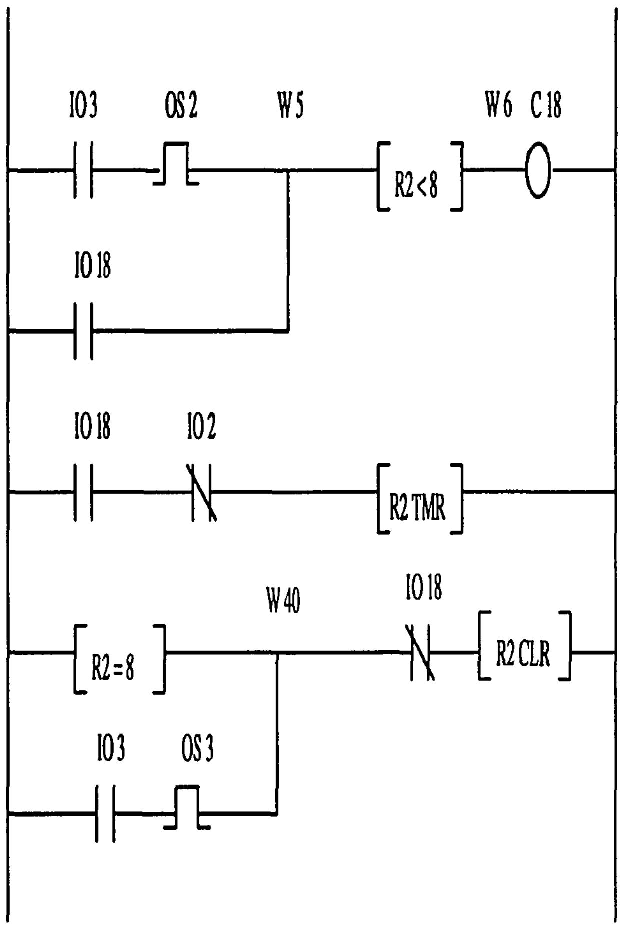


Figure 8: RLL diagram for open gate2