New Jersey Institute of Technology

Digital Commons @ NJIT

Theses

Electronic Theses and Dissertations

12-31-1991

A case study on petri net modeling, animation, and simulation

Suresh C. Gurala

New Jersey Institute of Technology

Follow this and additional works at: https://digitalcommons.njit.edu/theses

Part of the Manufacturing Commons

Recommended Citation

Gurala, Suresh C., "A case study on petri net modeling, animation, and simulation" (1991). *Theses*. 2490.
https://digitalcommons.njit.edu/theses/2490

This Thesis is brought to you for free and open access by the Electronic Theses and Dissertations at Digital Commons @ NJIT. It has been accepted for inclusion in Theses by an authorized administrator of Digital Commons @ NJIT. For more information, please contact digitalcommons@njit.edu.

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

A CASE STUDY
ON
PETRI NET MODELING, ANIMATION, AND SIMULATION

By

GURALA SURESH C.

Thesis submitted to the faculty of the Graduate School of New Jersey Institute of Technology in the partial fulfillment of the requirements for the degree of Master of Science in Manufacturing Engineering

1991

APPROVAL SHEET

Title of Thesis:

**A CASE STUDY
ON
PETRI NET MODELING, ANIMATION, AND SIMULATION**

Name of Candidate: **Gurala Suresh C.**

Master of Science in Manufacturing Engineering, 1991

Thesis and
Abstract Approved:

Dr. Mengchu Zhou, Advisor — Date
Assistant Professor
Electrical & Computer Engineering

Signature of the other
members of thesis committee:

Dr. Raj Sodhi — Date
Director, Associate Professor
Manufacturing Engineering

Dr. Nouri Levy — Date
Associate Professor
Mechanical Engineering

VITA

NAME:	**Gurala Suresh C.**
PERMANENT ADDRESS:	
DEGREE AND DATE TO BE CONFERRED:	
DATE OF BIRTH:	
PLACE OF BIRTH:	

COLLEGE & INSTITUTIONS ATTENDED:

	DATE	DEGREE
Jagadguru Mrugarajendra Institute of Technology (Mysore University) Mysore, India	1984-88	B.E.(Elec. & Com.)
New Jersey Institute of Technology Newark, New Jersey	1990-91	M.S.(MnE)

Major: Manufacturing Engineering

Positions held:

Research Assistant 1990-1991	Center For Manufacturing Systems. N.J.I.T Newark, New Jersey
Graduate Assistant 1991-Present	Finance Department., N.J.I.T Newark, New Jersey

Acknowledgements

I take this opportunity to express my deep gratitude to Dr. MengChu Zhou, Assistant Professor in Electrical & Coputer Engineering Department of N.J.I.T for his valuable guidance and help throughout the course of this work. It was his immaculate advice, which led my work to the set mark of culmination.

I am also very thankful to Dr. Nouri Levy and Dr.Sodhi for spending their precious time to review my work and to provide constructive suggestions for enhancing my work.

To My Parents

ABSTRACT

A CASE STUDY
ON
PETRI NET MODELING, ANIMATION, AND SIMULATION

Gurala Suresh C, Master of Science, 1991

Thesis directed by: Dr. *MengChu Zhou*, Assistant Professor of Electrical & Computer Engineering

Petri nets as a graphical tool has developed over the last decade into a suitable model for representing, modeling, designing, analyzing and studying concurrent systems. In this thesis, Petri nets are applied for analyzing, modeling and simulation of a flexible assembly station. An existing AT&T FWS200 flexible assembly workstation for printed circuit board is modeled using Petri nets without the consideration of insertion failures and other abnormalities in the system. The Petri net model is then simulated using Graphics simulation language (GSL).

TABLE OFCONTENTS

LIST OF FIGURES AND TABLES

CHAPTER
1

INTRODUCTION

1.1 BACKGROUND AND PREVIOUS RESEARCH

Historically speaking, the concept of Petri nets (PN) was developed by Carl Adam Petri in 1962 to be used for as a graphical and mathematical modeling tool to many systems which are characterized as being concurrent, asynchronous, distributed, parallel, non deterministic and/or stochastic. Since that time considerable work has been done in both theory and applications of Petri nets [3, 4, 12]. Petri nets were developed to model discrete event systems [1, 2]. By analyzing a petri net model, useful information about the underlaying system can be obtained [1, 2, 5, 6]. This information may reveal bottlenecks, deadlocks, etc., which exist and can be used tests on the actual system. Because Petri nets are a mathematical model, various tools have developed to analyze them [1, 5].

Petri net theory previously developed for modeling, analysis, and simulation of Flexible Manufacturing Systems (FMS) has been used in this report. The application of

Petri nets for modeling flexible manufacturing systems and production processes are previously proposed by Hack [7], Dubious & Stecke [8], and Narahari Viswanadham [9] .nets Hack proposed a brief description of the mapping of production schemata in Petri nets. His work centered in developing tools for analyzing safeness and liveness in a restricted class of Petri net, free choice nets using top-down analysis approach. Dubious and Steck emphasized the use of timed Petri nets in modeling FMS for analyzing the quantitative aspects of FMS performance and to conduct temporal performance analysis, ie., to determine production rate, resource utilization etc., Hence if modeled with timing it is possible to detect a bottleneck in an FMS or to determine a optimal buffer size, optimal pallet distribution etc. Narahari and Viswanadham proposed a method for developing Petri net models for complex FMS by combining simpler subnets. Krogh and Sreenivas [4] introduced the concept of essentially decision free places in Petri netsnets to represent the absence of unresolved resource allocation conditions.

Petri nets are useful tools for modeling systems with following characteristic:

1. *Concurrency and Parallelism*: In manufacturing system, many operations take place simultaneously.
2. *Asynchronous operations*: Machines complete their operations in variable amounts of time and so the model must be able to represent asynchronous events or operations.
3. *Deadlock*: In this case, a state can be reached where none of the processes can continue. This can happen when two processes share two resources. The order by which these resources are used and released could produce a deadlock.
4. *Conflicts*: This may occur when two or more processes require a common resource at the same time. e.g., two work stations may share a common transport system or might want access to the same database.

5. *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 the many allowed by the system structure.

These types of systems have been difficult to accurately model with differential equations and queueing theory. The application of Petri nets to Flexible Manufacturing Systems which can provide accurate models is because of the following reasons:

1. Petri nets capture the precedence relations and structural interactions of concurrent and asynchronous events.
2. They are logical models derived from the knowledge of how the system works. As a result, they are easy to understand and their graphical nature is very well visualized.
3. Deadlocks, conflicts, and buffer sizes can be modeled easily and concisely.
4. Petri net models have a well developed mathematical foundation that allows a qualitative analysis of the system.
5. Finally, Petri net models can also be used to implement real-time control systems for an automated manufacturing system. They can sequence and coordinate the sub systems as a programmable logic controller does.

Generally speaking, Petri net modeling includes two parts [13]: Ordinary Petri nets for system behavior analysis and Temporal Petri nets for system performance evaluation. The ordinary Petri nets also called non-timed Petri nets are used to analyze such system properties as deadlock-freeness, buffer-boundedness, reversibility and conflict-freeness. After ordinary Petri nets are built, time variables can be used to be associated with their places and transitions resulting in temporal Petri net models. These models serve to derive system performance indices including, productivity an machine utilization. When there are several operational settings possible, different Petri net models can be obtained and used to study the optimal setting by comparing the results of Petri net based performance evaluation.

1.2 OBJECTIVE OF THE THESIS

It is highly desirable for researchers, system analysts, and production engineers to have unified model for modeling, analysis, simulation and control of manufacturing systems [10]. Hence, Petri netsnets as a general graphical tool is very well suited to the distributed and concurrent systems which exhibit synchronization and contention for shared resources. Therefore, Petri nets have been claimed to be an ideal modeling tool for FMS's.

The objectives of this thesis are as follows:

1. To model AT&T Flexible Assembly Station for Printed Circuit Board at NJIT [19].
2. To study the operation of the System via Petri net.
3. To present an overview on Simulation and the IGRIP software [15].
4. To build the Petri net model in Silicon Graphics using IGRIP software.
5.. The Petri net model constructed in the IGRIP is simulated and animated using Graplhics Simulation Language (GSL).

1.3 ORGANIZATION OF THE THESIS

The thesis is organized as follows:

Chapter 2 gives a reveiw of Petri nets. In Chapter 3, a discussion on the application of Petri nets to modeling discrete event systems is presented. Specific attention is paid to the modeling and analysis of a flexible assembly system. In Chapter 4, a general overview of the simulation and the software IGRIP is dicussed. along with the steps as to how the simulation is acheived for the Petri net model are dicussed in Chapter 3. In Chapter 5, contributions, limitations, and the future work to be done are presented.

CHAPTER 2

PETRI NET THEORY

2.1 INTRODUCTION

Petri nets will be used throughout the thesis as a tool for modeling, analyzing, and simulating the existing physical Flexible Assembly Station for Printed Circuit board. Because of the various features of Petri nets they are very appropriate for modeling flexible manufacturing systems. They lead to a description of a system that can be investigated analytically, simulated and implemented for control [1, 2] .

A Petri net is an abstract, formal model of information flow. The properties concepts and techniques of Petri nets are being developed in a search for natural, simple and powerful methods for describing and analyzing the flow of information and control in systems, particularly systems that may exhibit asynchronous and concurrent activities. The major use of Petri nets has been the modeling of systems of events in which it is possible for some events to occur concurrently, but there are constraints on the concurrence of precedence of these occurrences. This chapter reviews the Petri net theory

2.2 BASIC DEFINITIONS

2.2.1 Structure of a Basic Petri Net

I intend to use Petri net to model the Flexible Assembly Station in this thesis. Place-Transition nets are special bipirate graphs. They are also known as Petri nets. An example of Petri net is shown in Figure 2.1. The standard Petri net model or a Petri net graph is defined by a set of places, a set of transitions and a set of directed arcs which connect places to transitions or vice versa. Places are represented by circle, transitions by bars and connection between them by arcs. If an arc is directed from node *i* to node *j* ie., either from place to transition or transition to place, then i is an input to *j* and *j* is an output to *i*. Places may contain tokens represented by dots. A Petri net with tokens is a marked Petri net. The marking of a marked Petri net is a vector, the elements of which are given by the distribution of tokens in the places of the net.

A marking represents the state of system being modeled. Generally places represent conditions and transitions represent events. A place is an input(output) place of transition is an arc which is exists from place(transition) to the transition(place). The dynamic behavior of a system is modeled as follows: the occurrence of an event (state change) is represented by the firing of corresponding transition. The movement of tokens in the net resulting from the firing of one or more transition represents a change in the state.

To complete the definition, it is necessary to define the relationship between places and transitions. This is done by specifying two functions connecting transitions to places*:* *I*, the input function and *O*, the output function.

In a Petri net graph, the input and output functions are represented as arcs going from places to transitions and transitions to places respectively. An arc is directed from place p_i to transition t_j if the place is an input of the transition. Similarly an arc is directed from a transition t_j to place p_i if the place is an output of the transition. A Petri net is thus a directed graph, since the arcs are directed.

Moreover, since the Petri nodes can be partitioned into two sets (places and transitions) such that each arc is directed from an element of one set to an element of the order of the other set, a Petri net is a bipartite directed graph.

Formally, an ordinary Petri net (PN) is a five-tuple,

$$\text{PN} = (P, T, I, O, m) \tag{2.1}$$

where,

$P = \{ p_1, p_2, \ldots, p_n \}$, $n>0$, and is a set of n places;

$T = \{ t_1, t_2, \ldots, t_s \}$, $s>0$, and is a set of s transitions;

I is an $n \times s$ matrix indicating the places which are the input of each transition. It is a mapping : $P \times T \rightarrow \mathrm{N}$ corresponding to the set of directed arcs from places to transitions, where N = {0, 1, 2, ... }.

O is an $n \times s$ matrix indicating the places which are the output of each transition. It is a mapping : $P \times T \rightarrow \mathrm{N}$ corresponding to the set of directed arcs from transitions to places, where N = {0, 1, 2, ... }.

$m : P \rightarrow N$ and is a marking whose ith component represents the number of tokens in the ith place. An initial marking is denoted by m_0.

These four items are the set of places, the set of transitions, the input function define the structure of the Petri net [2].

A Petri net graphically consists of :

1. Circles (called places) representing conditions or availability of resources e.g. machines, parts, data, etc.
2. Bars (called transitions) representing the initiation or termination of an event.
3. Black dot (called a token) in a operation place representing the operation in that place being executed, while that in a resource place representing the availability of the corresponding resources.
4. A pattern of tokens in a Petri net (called a marking) representing the state of the system.

We denote postset and preset as follows :

$t_j^{\bullet}$ is the set of all output places of transition t_j.
${}^{\bullet}t_j$ is the set of all input places of transition t_j.
$p_i^{\bullet}$ is the set of all output transitions of place p_i.
${}^{\bullet}p_i$ is the set of all input transitions of place p_i.

For the Petri net shown in Figure 2.1 :

$$P = \{p_1, p_2, p_3, p_4, p_5, p_6\}, \qquad T = \{t_1, t_2, t_3, t_4, t_5\}$$

$${}^\bullet t_1 = \{p_1\}, \quad {}^\bullet t_2 = \{p_2\}, \quad {}^\bullet t_3 = \{p_3\}, \quad {}^\bullet t_4 = \{p_4, p_5\}, \quad {}^\bullet t_5 = \{p_6\}$$

$$t_1^\bullet = \{p_3\}, \quad t_2^\bullet = \{p_5\}, \quad t_3^\bullet = \{p_4, p_5\}, \quad t_4^\bullet = \{p_6\}, \quad t_5^\bullet = \{p_4\}$$

$${}^\bullet p_3 = \{t_1\}, \quad {}^\bullet p_4 = \{t_3, t_5\}, \quad {}^\bullet p_5 = \{t_3\}, \quad {}^\bullet p_6 = \{t_2, t_4\}$$

$$p_1^\bullet = \{t_1, t_2\}, \quad p_2^\bullet = \{t_2\}, \quad p_3^\bullet = \{t_3\}, \quad p_4^\bullet = \{t_4\}, \quad p_5^\bullet = \{t_4\}, \quad p_6^\bullet = \{t_5\}$$

$$I = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad O = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

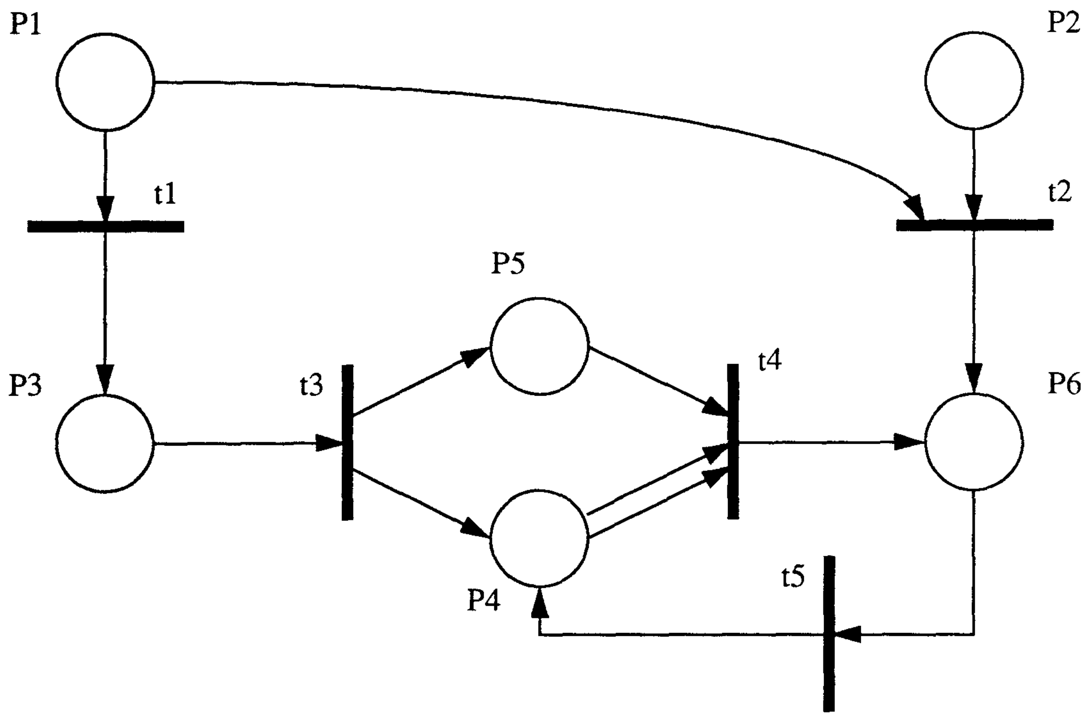


Figure 2.1: Example of PN

2.2.2 Marking of a Petri Net

A marking of a Petri net is an assignment of tokens to the places in that net. Tokens reside in the places of the net. The number and positions of the tokens in a net may change during its execution. On a Petri net graph, tokens are represented by small solid dots inside the circles representing the places of the net. Since any number of the tokens can be assigned to the places, there is an infinite number of markings for a Petri net.

$m = [m_1, m_2,..., m_n]^\tau$ is an n-dimensional integer valued vector indicating the number of tokens in each place, with each element corresponding to one of the places, and is known as the marking of the Petri net.

For the example of Figure 2.2 : $m = [1, 0, 0, 2, 0, 1]^\tau$

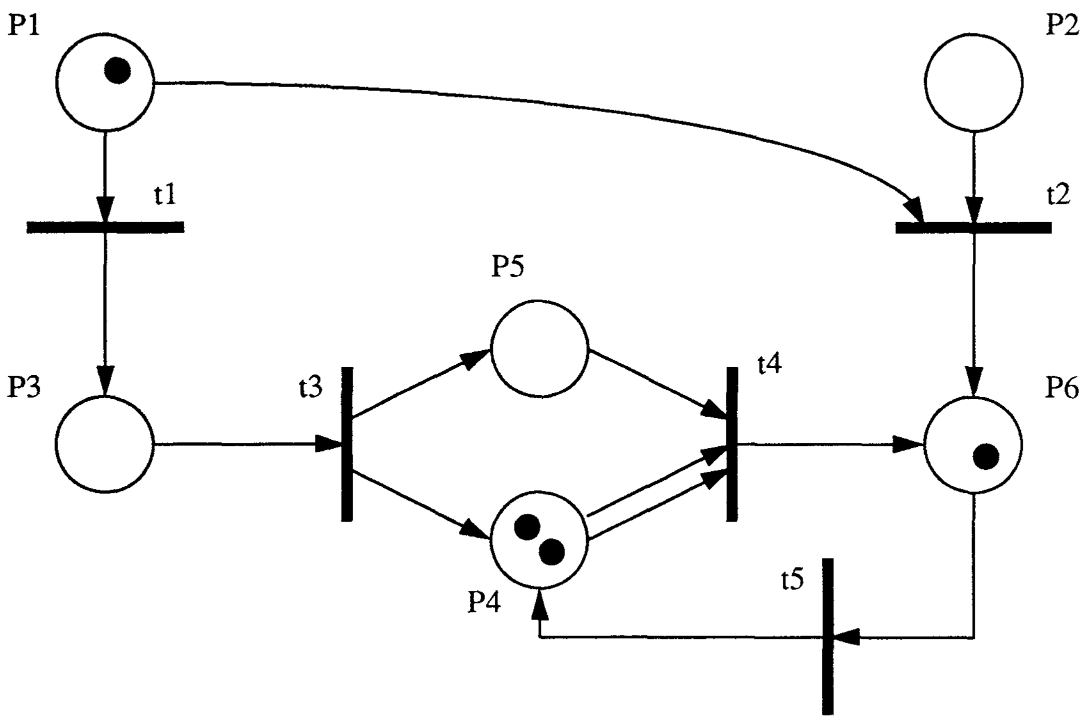


Figure 2.2: Example of Marked PN

2.2.3 Rules of Operation

A Petri net executes by firing transitions. A transition may fire if it is enabled. A transition is enabled if each of its input places has at least $I(p, t)$ tokens in it. A transition fires by removing one token from each of its input places and then placing one token into each of its output places. The movement of tokens through the Petri net graph represents the flow of information or control in the system.

Hence, a transition t is enabled by marking m_0 if every input place of this transition contains at least $I(p, t)$ tokens, i.e., $\forall\, p \in {}^{\bullet}t,\ m_0(p) \geq I(p,t)$

Every transition enabled by marking m_0 can fire. When a transition fires $I(p, t)$ tokens are removed from each of its input places and $O(p, t)$ tokens are added to each of its output places. Therefore a new marking m_1 obtained by the firing of transition t, verifies for each place p.

Using the matrix form, we have

$$m_1(p) = m_0(p) - I(p,t) + O(p,t) \qquad \forall\, p \in P \qquad (2.2)$$

Tokens are indivisible i.e. a token can be removed from a place by only one transition. Except for the above restrictions, firing of transitions proceeds in an asynchronous manner.

For example in the Petri net shown in figure 2.2, t_1 is enabled, but not t_2. After t_1 has fired once, the new marking is : $m_1 = (0, 0,1,2, 0,1)^{\tau}$

This is represented as :

$$m_0 \xrightarrow{t} m_1$$

For any transition t_j and all places p_i ($i \in \{1, 2, ..., n\}$), the relation (2.2) can be written in a compact form as :

$$m_1 = m_0 + (O - I) . X_j = m_0 + CXj \tag{2.3}$$

where,

X_j is a m-dimensional vector with all components equal to zero except the j^{th} one, which equals 1 : $X_j = (0, 0,...,1,...0)^{\tau}$ and

$$C = O - I$$

2.2.4 Incidence matrix

The incidence matrix denoted by C, characterizes the structure of the Petri net in the following way : the columns of the matrix corresponds to the transitions of the net and the rows to the places. C is therefore, an $n \times s$ matrix, such that:

$$C(p, t) = O(p,t) - I(p,t) \qquad \forall\, p \in P, \ \forall\, t \in T$$

For example the Petri net in figure 2.1 has the following incidence matrix,

$$C = \begin{bmatrix} -1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 1 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & 1 & -1 \end{bmatrix}$$

2.3 BASIC PROPERTIES OF PETRI NETS

Once the system is modeled using Petri nets it is essential that the Petri nets support for analysis of many properties and problems associated with concurrent systems. In this basic definitions and results pertaining to the behavioral properties ie., the Petri net model which depends on initial marking is discussed [13].

1. *RECHABILITY*

This is a fundamental basis for studying the dynamic properties of any system. According to the transition rule, once the firing of an enabled transition occurs, there may be a change in the distribution of tokens in the net. Then a marking m is said to be reachable from marking m_0 if there exists a sequence of firings that transforms m_0 to m.

2. *BOUNDEDNESS*

A Petri net is said to be k-bounded if the number of tokens in each place does not exceed a finite number k for any marking reachable from m_0. Also a Petri net is bounded, if for each place in the net, there exists an upper bound to the number of tokens that can be there simultaneously. A Petri net is said to be safe if it is 1-bounded, i.e., if k=1.

Mathematically a Petri net is k-bounded if,

$$m(p) \leq k, \ \forall \mathrm{m} \in \mathrm{R}(m_0), \quad \forall p \in P$$

In a manufacturing environment, the boundness or safeness of a Petri net indicates the absence of overflows in the modeled system [9, 11]. For example, a buffer in the production facility will have a finite capacity, and its representation as a bounden place will guarantee that the resulting control code will not allow this capacity to be exceeded. Safeness means no more than one token will mark the place. This implies that there is no possibility to restart the ongoing process if the place represents the process. For instance, if a place represents a machine that can only process one part at a time, safeness would guarantee that no other parts are loaded until the current one is completed. If a place happens to represent the availability of a single resource, then this place must be safe.

3. *LIVENESS*

A Petri net is said to be live if, no matter what marking has been reached from m_0, it is possible to ultimately fire any transition of the net progressing through some further firing sequence.

Mathematically a net is said to be live if,

$\forall\ t \in T$ and $\forall$ m $\in$ R(m_0), there exists a sequence of transitions σ, such that firing σ leads to a marking which enables t.

A transition that cannot fire is a redundant transition and can be eliminated from the net. However, if such a transition exists in a net model, it needs to be identified since it may represent an error in the model or an inconsistency in the system being modeled. A transition is dead in a marking if there is no sequence of transition firings that can enable it. A transition is potentially firable if there exists some firing sequence that enables it. A transition is live if it is potentially firable in all reachable markings.

Liveness is tied to the concept of deadlock and deadlock-freeness, as related to the modelling of operating systems [11]. Thus it may be important not only that a transition be firable in a given marking, but that it stay potentially firable in all markings reachable from that marking. If this is not true, then it is possible to reach a state in which the transition is dead, perhaps signifying a possible deadlock.

4. *REVERSIBILITY*

A Petri net is said to be reversible, if for each marking m in $R(m_0)$, m_0 is reachable from m. Thus in the reversible net one can always get back to the initial marking (including failure states) [14].

Mathematically a net is said to be reversible if,

$$\forall\, \mathrm{m} \in \mathrm{R}(m_0), \text{ if } m_o \in R(m)$$

i.e. there exists a sequence of transitions whose firing lead m to m_0.

5. *COVERABILITY*

A marking m in a Petri net is said to be coverable if there exists a marking m_1 in $R(m_0)$ such that $m_1(\mathrm{p}) \geq \mathrm{m(p)}$ for each p in the net.

Mathematically the net is said to be coverable if,

$$m_1 \in \mathrm{R}(m_0),\ \exists\ \ m_1(\mathrm{p}) \geq m(p),\ \ \forall\, p \in P$$

6. *PERSISTENCE*

A Petri net is said to be persistent if, for ant two enabled transitions, the firing of one transition will not disable the other. Also, in a persistent net a transition once enabled will stay enabled until it fires.

7. *CONSISTENCY*

A Petri net is consistent if and if only there exists a marking m and a firing sequence σ such that, σ brings the marking m of the net back to itself. and σ contains each transition at least once. σ is called a cyclic firing sequence. Also, consistency is different from reversibility in the sense that in case of reversibility, guarantees consistency but not vice versa.

CHAPTER 3

SYSTEM MODELING WITH PETRI NETS

3.1 SYSTEM MODELING WITH PETRI NETS

Analytical methods have gained widespread acceptance as a powerful and inexpensive tool for the performance evaluation and predictions of the systems. They provide accurate and robust performance indices, even in the case of complex multiprocessor .computer systems. This is of great importance, since the simulation and the setup of prototypes for this type of computing systems is rather inexpensive.

Petri nets a graphical tool very well suited to the description of distributed and concurrent systems which exhibit synchronization and contention for shared resources, have been widely used for modelling logic controllers, computer networks, communication protocols, operating systems, as well as production systems. This chapter discusses how Petri nets can be used to model such systems.

3.1.1 Theory

In many fields of study, a phenomenon is not studied directly but indirectly through a model of the phenomenon. A model is a representation, often in mathematical terms, of what are felt to be the important features of the object or system under study. By the manipulation of the representation, it is hoped that the new knowledge about the modeled phenomenon can be obtained without danger, cost, or inconvenience of manipulating the real phenomenon itself.

Most of the systems we want to model are quite complex. For example, computer systems are very complex, often large, systems of many interacting components. Each component can be quite complex, as can its interaction with other components in the system. This is also true of many other systems. Economic systems, legal systems, traffic control systems, manufacturing systems and chemical systems all involve many individual components interacting with other components, possibly in complex ways. Thus despite the diversity of systems which we want to model, several common points stand out. These should then be featured as a useful model of these systems. One fundamental idea is that systems are composed of separate, interacting components. Each component may itself be a system, but its behavior can be described independently of other components of the system except for well-defined interactions with other components. Each component has its own state of being. The state of a component is an abstraction of the relevant information necessary to describe its (future) actions. Often the state of a component depends on the past history of the component. Thus the state of the component may change over time. The concept of "state" is very important to modeling a component. For example, in a model of hospital, the state of a patient might be critical, serious, fair, good, or excellent. The components of a system may exhibit concurrent behavior. Activities of one component of a system may occur simultaneously with other activities of other components. In a computer for example, peripheral devices such as, card readers, line printers, tape drives and so on, may all may operate concurrently under the ultimate control of the computer. In a factory a number of activities take place simultaneously. The concurrent nature of activity in a system creates some difficult modeling problems. when the components of a system interact, it is necessary for synchronization to occur. The transfer of information or materials from one component to the other requires that the activities of the involved components be synchronized while the interaction is occurring. This may result in one component waiting for another component. The timing of actions of different components may be very complex and the resulting interactions between components become difficult to describe.

The development of high speed computers has greatly increased the use and usefulness of modelling. By representing the system as a mathematical model, converting that model into instructions for a computer and thereby running the computer, it is possible to model larger and more complex systems than ever before. This has resulted in considerable study of computer modeling techniques and of computers themselves. Computers are involved in modeling in two ways: 1). computational tool for modeling and 2). as a subject of modeling.

The use of analytical models requires good background in applied probability theory as well as some experience in the modeling field. Indeed, it is very important to include in the model those features of the system that have a significant impact on the performance indices of interest, while not unnecessarily complicating the model with too many details.

For this reason the availability of high level interfaces for the specification of a stochastic model is extremely important. Queueing theory has long been used for this purpose. Indeed, it is much simpler to specify a model in terms of a network of queues rather than directly at the level of the associated stochastic process. Moreover, a queueing network model is much better understood by a non specialist than a lower level probabilistic description.

More recently, Stochastic Petri nets were introduced to provide a simple graphical interface to the construction of stochastic models of computer systems. The advantage of SPNs over queueing networks are two fold. First, they allow one to easily represent some features of computer systems, such as synchronization, that are not easily described with queueing network models. Second, they allow the description of system at different levels of abstraction, so that the user can choose the one that best suits his needs. Thirdly, if system behavior is non stochastic in nature, then performance results can obtained by event driven simulation.

In brief, the Petri net description of systems concentrates on two concepts : events and conditions. Events are conditions that occur in system and condition describe the state of various parts of the system. The condition is either true or false. In order for events to occur, certain conditions, referred to as pre-conditions, must exist. After an event occurs, these pre-conditions change and another set of conditions, referred to as post-conditions, become valid. The post-conditions of one event may be pre-conditions of another and so a sequence of events may occur. By listing the events that take place in system along with conditions necessary for each event to occur and the conditions that result after each event, a system can be modeled.

3.1.2 Example

A typical example described by peterson [17] is of a computer system :

- Jobs appear and are put on an input list. When the processor is free, and there is a job on the input list, the processor starts to process the job.
- When a job is complete, it is placed on an output list, and if there are more jobs on the input list, the processor continues with another job; otherwise it waits for another job.

This is a very simple system composed of several elements : the processor, the input list, the output list and the jobs. We can identify several conditions of interest.

The processor is idle : p_1

A job is on the input list : p_2

A job is being processed : p_3

A job is on the output list : p_4

and several events :

A new job enters the system : t_1

Job processing is started : t_2

job processing is completed : t_3

A job leaves the system : t_4

After identifying the conditions and events of this system, it can be modeled using Petri nets. The Petri net model thus obtained is as shown in Figure (3.1).

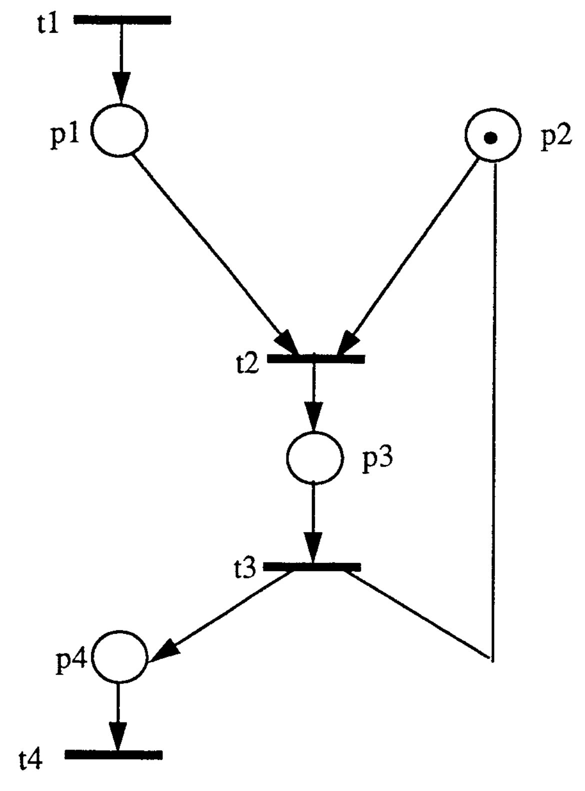


Figure 3.1 : Petri net model of a simple computer system

3.2 PROPERTIES OF PETRI NETS IN MODELING

In the Petri net model, unrelated events can occur independently and in effect simultaneously. Another major feature of Petri net is their asynchronous nature. There is no inherent measure of time or flow of time in a Petri net except for timed transition. The Petri net structure itself contains all necessary information to define the possible sequences of the events for the modeled system.

A Petri net can reflect a real-time situation where several things are happening concurrently. The order of occurrence of events is not unique so that any set of may occur. While non-detrministic is advantageous from the modeling point of view, it introduces considerable complexity in the analysis of Petri net. To reduce this complexity, one limitation is generally accepted in the modeling of systems by Petri net; the firing of a transition is considered to be instantaneous. The exception is the fore mentioned timed transition. Petri nets are also used for analyzing and synchronizing nature events, like the dining Philosophers problem explained by J.L. Peterson.

3.3 MODELING OF A FLEXIBLE PCB ASSEMBLY SYSTEM

The AT&T FWS200 flexible robotic assembly system for printed circuit boards include the following components:

1. Robots to perform assembly operations
2. Tools provided to robots
3. Feeders to supply components
4. Conveyors to supply boards.

The above mentioned systems have different layouts, hence it is essential to have the optimal sequence of component insertions/placements which vary from one board to another. Also some insertions may be more restricted than others, as different insertions may require different robots with perhaps different tools. In order to achieve global optimum, all these tasks must be well planned and their operations should be coordinated.

3.3.1 Modeling Issues

The first important issue is the modeling of sequential operations for each robot in the system. For example, after certain layout of the system such as sequence of assembly tasks, sequence of component insertions, and locations of feeders have been determined, there exists a problem of planning robot operations including pick-up, transport, insertion, etc., and making decisions in case of robot failures. Hence a certain order of operations needs to be followed by each robot in the system [19].

The second modeling issue is synchronization. For example, a robot will pick up a component only when it is present. It will never finish pick up operation if the component is missing. Similarly, the PCB to be assembled has to be present before the insertions of components can be fulfilled. Also, when other resources are needed to complete the operation, they must also be available and ready.

The third issue is modeling of concurrency. Concurrency means that there are parallel relations among the concerned events [18]. For example, two physical events pick up a part by robot1 and pick up a part by robot2 are concurrent if both the events occur simultaneously. High concurrency among system resources implies high productivity.

The fourth issue in modeling is the conflict. This is because of the resource sharing or sharing of the space. The mutual exclusion concept proposed in [21, 20] is especially useful to model the characteristic of resources shared by independent processes or sequentially related ones.

3.3.2 System Description

The major components of the AT&T FWS200 flexible workstation shown in Figure (3.2) for which the Petri net model is constructed is as follows [19]:

1. The frame structure supports the control cabinet, robot arms, platen, work table, etc.
2. The Control Cabinet houses hardware including an AT&T personal computer, Industrial I/O modules and STD bus, control electronics, display, pneumatic and ventilation systems, and power supplies.
3. The Robot Arms are able to accomplish pick and place operations, with their X, Y, Z and O motions in high accuracy (+/-0.001 in. along X, Y, Z axes and +/-0.150 about Z axis).
4. The AT&T PC 6386 computer serves as the supervisory controller for the system.
5. The Drive Electronics contain the electronics for controlling the various axes, with two servo motor amplifiers for separate Z and O motions and a stepper motor amplifier for both X and Y motions.
6. The Control Panel contains power buttons, touch display, pressure and vacuum gauges and include IRI SV512 and ICOS M1000 modules.
7. The Peripheral Equipment includes part feeders and others.

The above system is used to accomplish tasks such as assembly of surface mount components, assembly of through hole components etc. The common work area is $.24" \times 32"$

3.3.3 Operation of the System

The working cycle of the above system without considering the tool

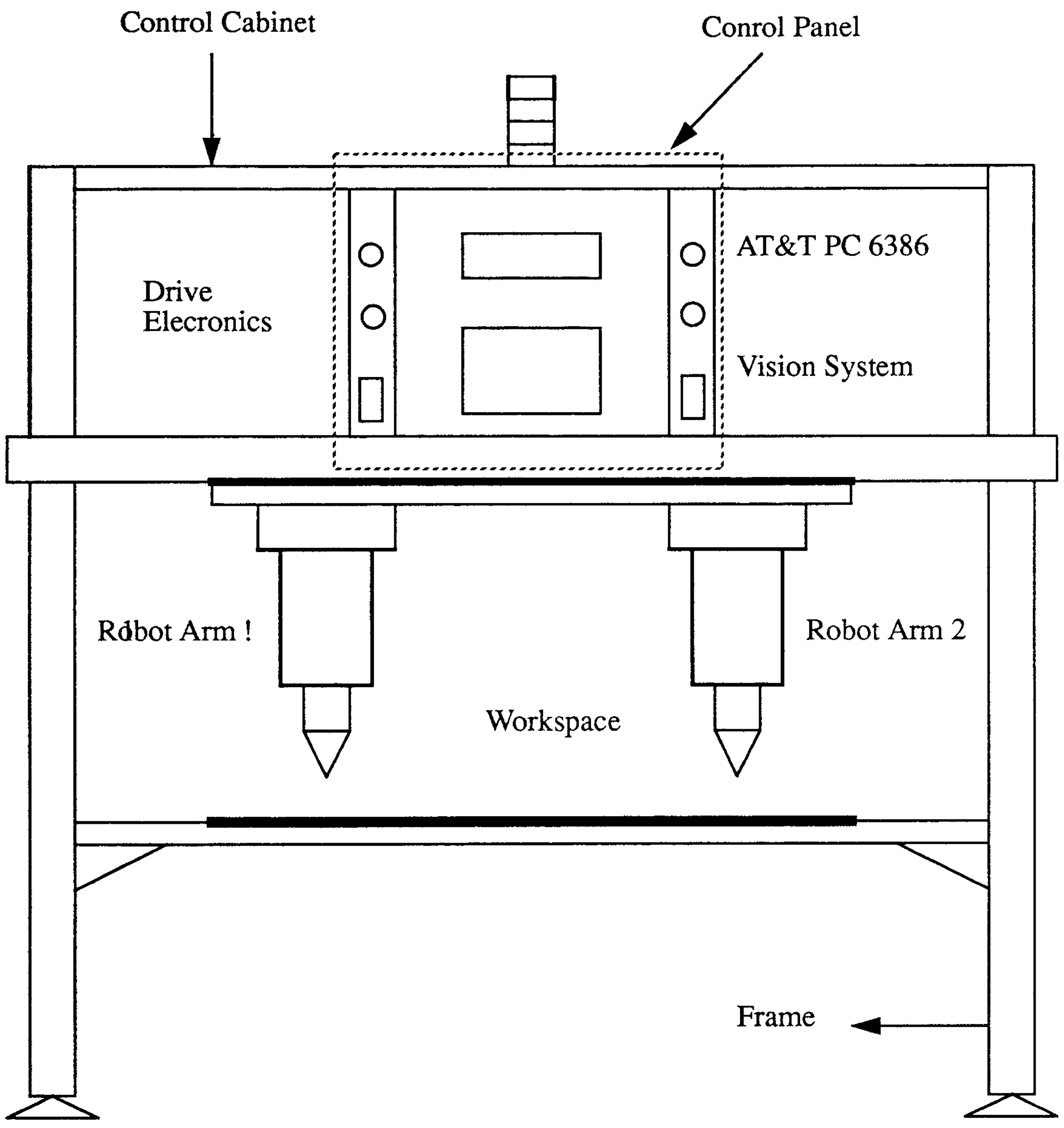


Figure 3.2 : AT&T Workstation

changes, insertion failures and robot errors is as follows:

1. Initially the PCB area, feeder area, components for robot1 and robot2, and the robots themselves are available.
2. Depending on the timing, the robot 1 picks up a component which is available for it from the feeder area.
3. Once the component is picked by robot 1, it moves into the PCB area thereby making the feeder area available for robot 2.
4. The robot 1 then inserts the component, and in the meantime the robot picks up the component available for it from the feeder area.
5. After the component insertion by robot 1 is completed, it moves away from the PCB area thereby, making the PCB area available for robot 2.
6. The robot 2 now moves into the PCB area, thus leaving the PCB area available for robot1.
7. The robot then inserts the component and moves outfrom the PCB area, during which robot 1 is ready to pick up component from the feeder area.
8. The cycle repeats.

3.3.4 Petri Net Execution

In a dedicated production line for assembly of PCB, a robot does repeatedly the following jobs: picking-moving-inserting-moving. Like such a robot, each arm of the FWS-200 workstation can do these jobs. However since both the arms work on the same circuit boards and obtain components from the same feeder area, avoidance of arm collision has to be considered. Collision avoidance can be achieved by mutual exclusion structure. Thus when one robot is doing a job above the PCB area, the other is prohibited from moving into the same area. Mutual exclusion structure is also used in the feeder area because of the possible collision which may occur even in this area. Using the mutual exclusion theory developed by Zhou [18, 21, 20], conclude that the net is live, safe, and reversible. These properties gaurentee the following important charecteristics of the system:

1. The liveness implies that the system is free from deadlock.
2. The safeness of a resource place such as p1 and p2 guarantees that the modeled resource is unique. The safeness of an operation place such as p12 and p22 guarantees that there will not be any attempt to re-initiate any ongoing operatio until it is completed.
3. The reversibility implies that the system can be restarted, i.e. it can return to its begining state from any current state. This guaranrees the repetitive operatios of this system.

For the above activities of the system, the design of the Petri net model is as shown in Figure (3.3). The meaning of places pi (i = 1 to 25) and transitions tj (j = 11 to 25) is explained in Table1. R1 and R2 represent Robot1 and Robot2 and their activities respectively.

Table 1. Places and Transitions in the Petri net of Figure 3.3

p1	: Robot1 ready	t11	: Completion of Robot1 picking the component from the feeder area
p2	: Robot2 ready	t12	: Availability of feeder area and non-availability of PCB area
p3	: Feeder area available	t13	· Completion of Robot1 moving into PCB area
p4	: PCB available	t14	: Completion of insertion by Robot1
p11	: Component for Robotl available	t15	: Completion of Robot1 moving out of PCB area and there by making the PCB area and Robot1 available
P12	: Robot1 picking the component	t21	: Completion of Robot2 picking the component from
p13	: Robot1 moving into PCB area		
p14	: Robot1 inserting component		
p15	: Robot1 moving out of PCB area		

p21 : Component for Robot2 available
p22 : Robot2 picking
p23 : Robot2 moving into PCB area
p24 : Robot2 inserting
p25 : Robot2 moving out of PCB area

the feeder area
t22 : Availability of feeder area and non availability of PCB area
t23 : Completion of Robot2 moving into PCB area
t25 : Completion of Robot2 moving out of PCB area and thereby making the PCB area and Robot2 available

The sample of complete cycle of the system operation of the system is illustrated by Petri nets as shown in Figures 3.3 to 3.14, when the assumption is made that Robort1 is prioritized.

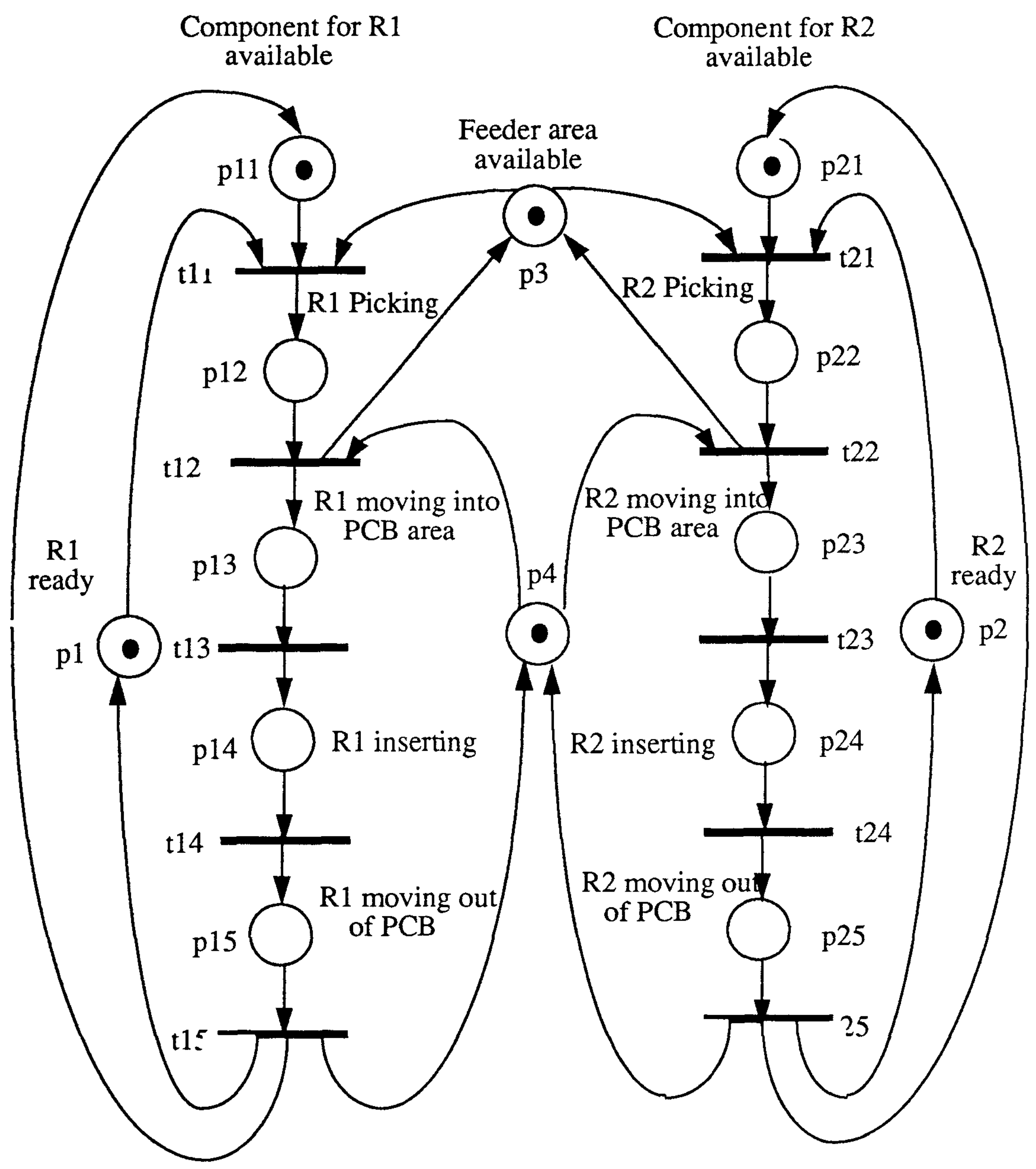


Figure 3.3 : Initial marking

m_0 = { 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0 }

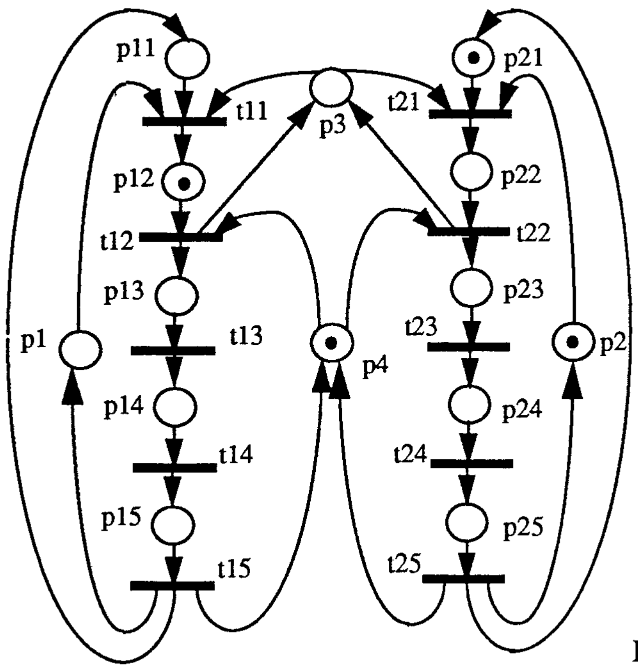
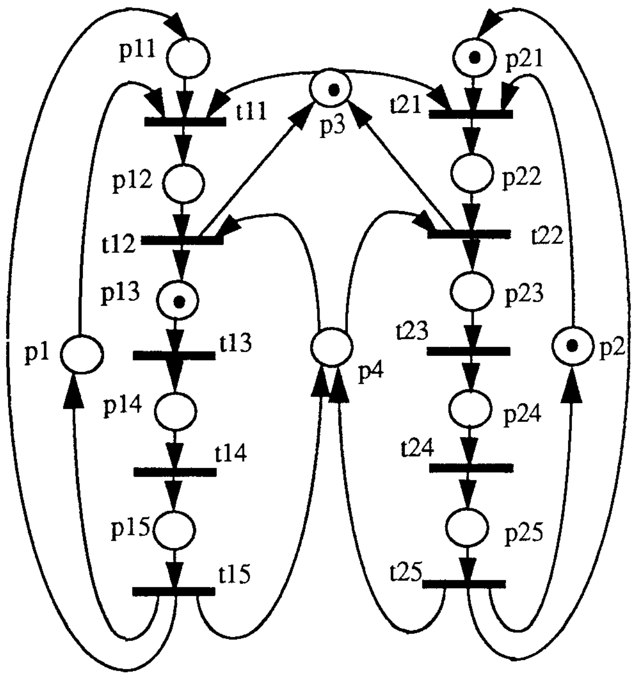


Figure 3.4 : Robot1 picking

m_1 = { 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0 }

Figure 3.5 : Robot1 moving into PCB area.

m_2 = { 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, }



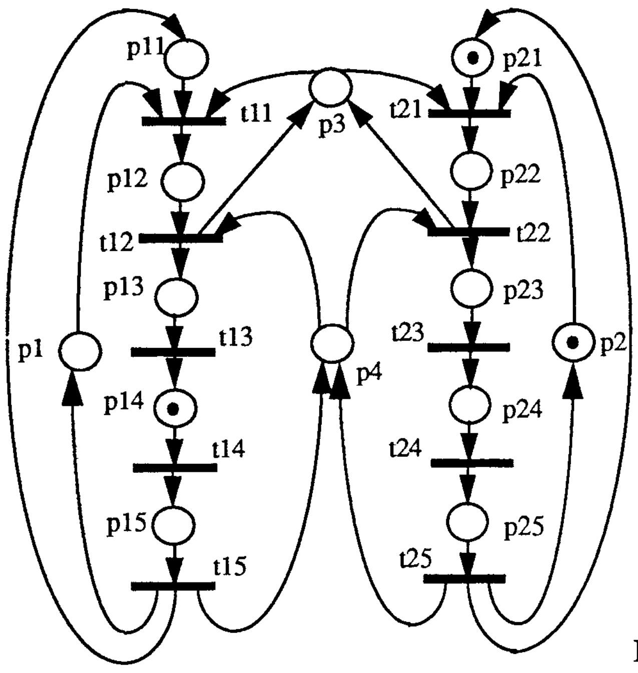
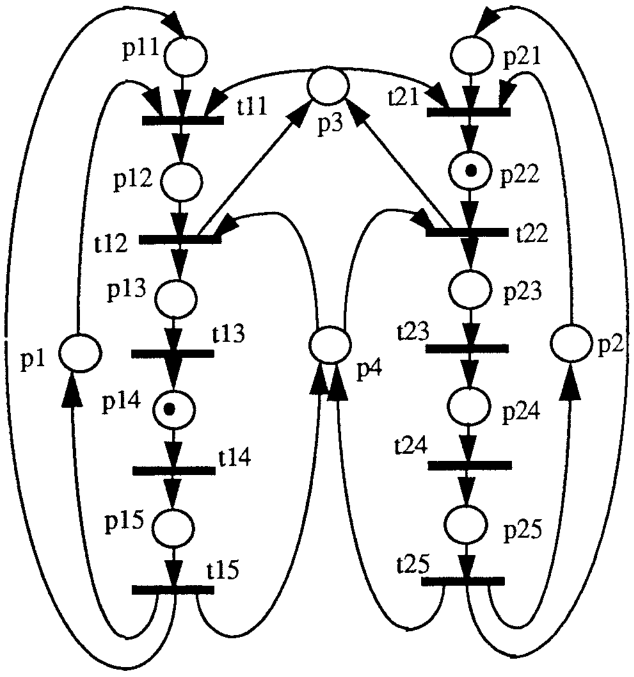


Figure 3.6 : Robot1 inserting

$m_3 = \{ 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0 \}$

Figure 3.7 : Robot 2 picking.

$m_4 = \{ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, \}$



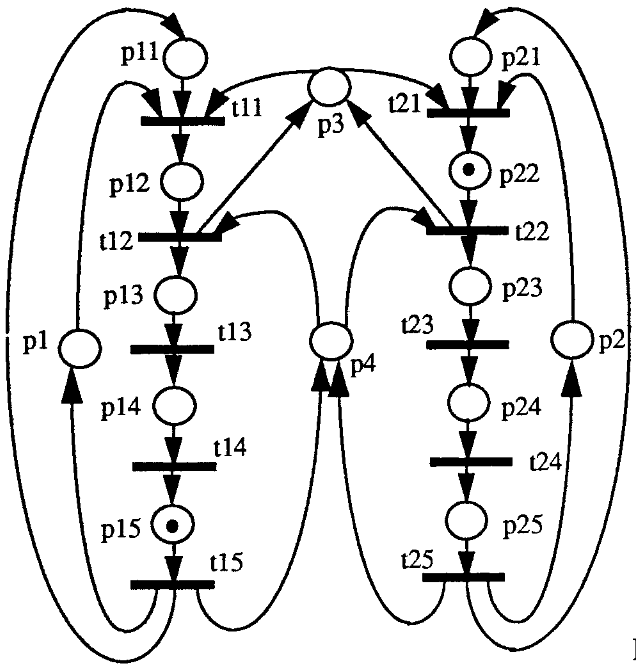
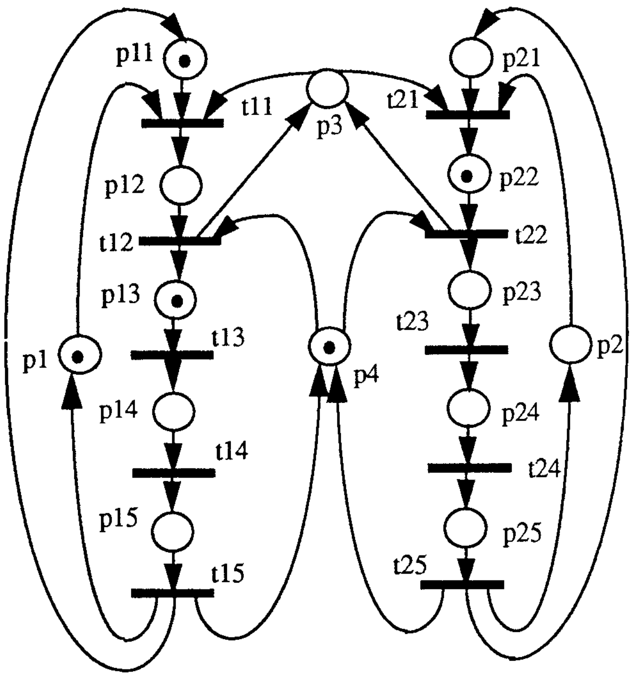


Figure 3.8 : Robot1 moving out

m_5 = { 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0 }

Figure 3.9 : Robot1 ready.

m_6 = { 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, }



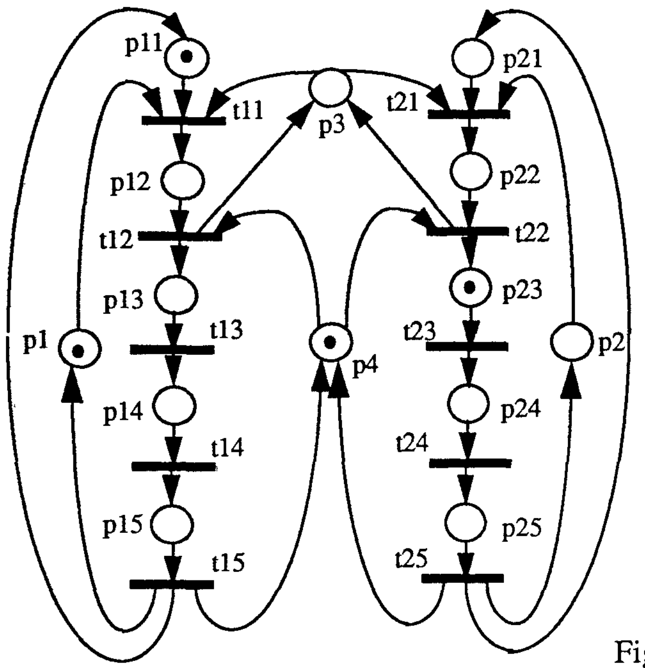
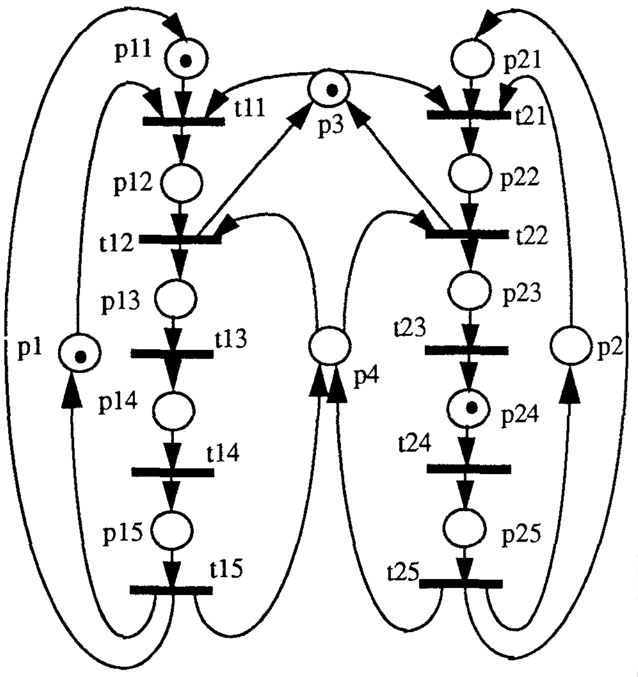


Figure 3.10 : Robot2 moving into PCB area

$m_7 = \{ 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0 \}$

Figure 3.11: Robot2 inserting.

$m_8 = \{ 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, \}$



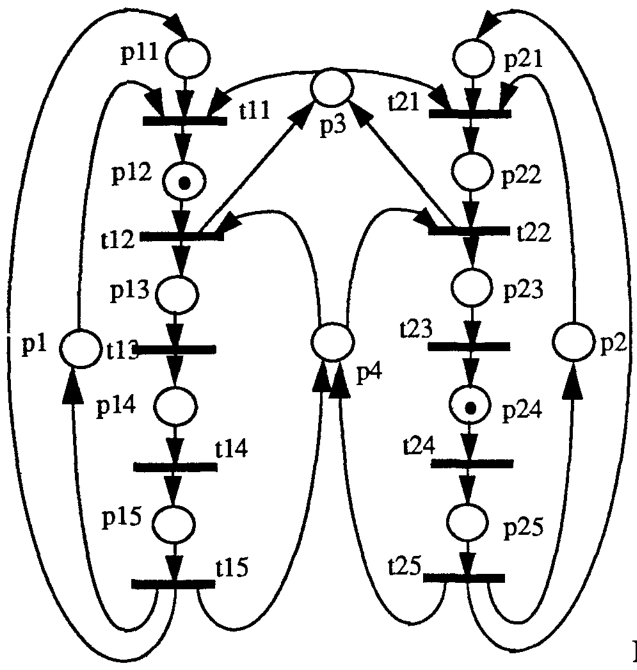
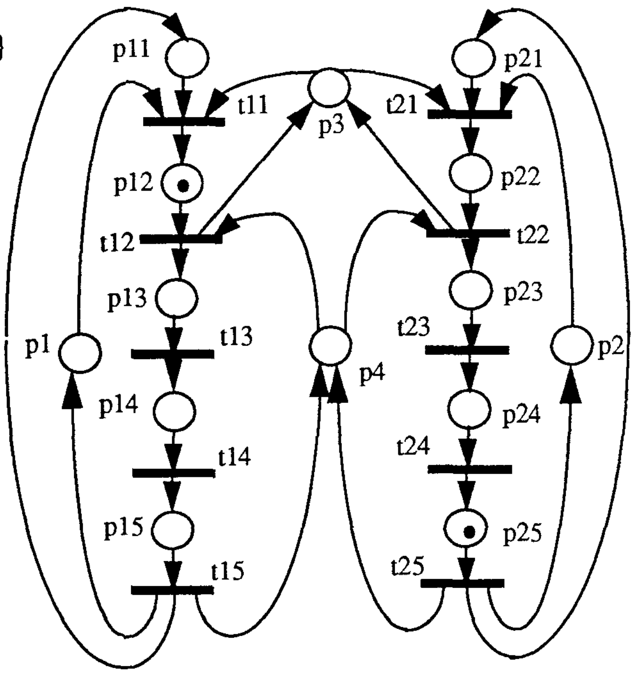


Figure 3.12 : Robot1 picking

$m_9 = \{ 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0 \}$

Figure 3.13 : Robot2 moving out.

$m_{10} = \{ 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, \}$



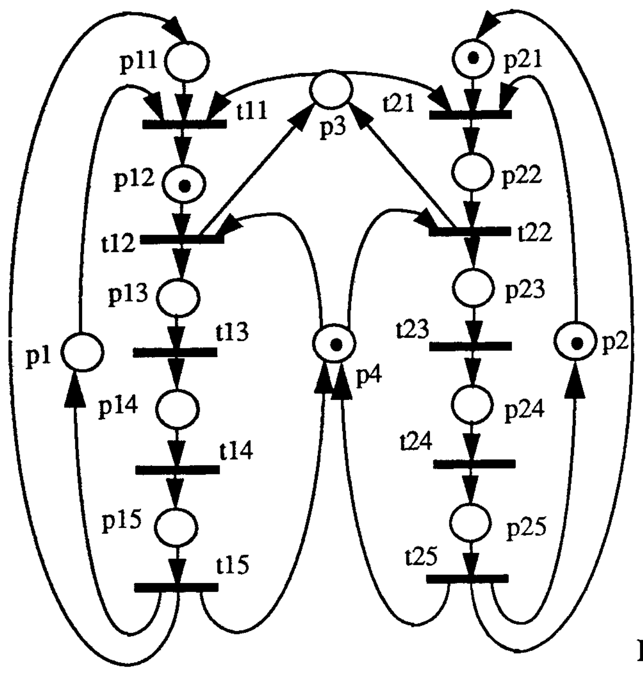


Figure 3.14 : Robot2 ready.

m_{11} = { 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0 }

CHAPTER 4

SIMULATION

4.1 INTRODUCTION

Simulation is the technique of constructing and running a computer model of a real system in order to make dynamic analysis without disrupting its environment, prior to its implementation. Simulation provides requsite information to determine the feasibility of a system by studying it under totally dynamic conditions. It is used to construct and display mathematical models with the same constraints as those of the actual robots, so as to manipulate and visualize the three dimensional models accuretely. It is also a real time analysis and control tool as it allows the designer to visualize motions at every stage of the cycle in real time. It also helps an engineer select the best robot for the job based on interference checks, and recahability of various points in the workcell. Various studies like cycle time analysis and collision detection help in optimization.

Simulation systems provide significant time savings in the layout and modeling of robotic work cells. Furthermore, as manufacturing equipment becomes more integrated complex and costly, these systems provide added assurance in cell layout optimization. It has been estimated that the task of implementing a robotic workcell is devoted to cell layout, equipment design, robot selection, and hardware mockup of the workcell. Remaining efforts are in the programming and actual implementation of the factory floor.The general characteristics of a simulation systems as well as the salient features of the IGRIP, the simulation system used in this work, are discussed. They are examplified by the design of Petri net animation and simulation for the flexible assembly workstation dicussed in Chapter 3.

4.2 CHARECTERISTICS OF A SIMULATION SYSTEM

The main objectives of simulation system:

1. Improved Accuracy
2. Improved Communication
3. Reduces Development Time

A simulation system should have good graphics capabilities as well as solid modeling modules so that the user can validate the model accuracy and completeness. It should also allow the import of standard file format such as IGES (Initial Graphics Exchange Specification). The system should be user friendly to enable easy, interactive modeling and simulation. Ability to simulate different operations like painting, welding, etc., is desired to increase the system's versatility. Features like collision detection and cycle time analysis are a must to assist in optimization. The user should have an access to a database of the most commonly used robots. The system should be able to simulate the kinematic and dynamic behavior of the robots. The simulation system should possess the capability of interfacing with the shop floor equipment so as to download the simulation sequence directly to the robot controller.

Simulation should be continued even after workcell implementation on the shop floor for optimization. The underlying philosophy is to provide a dynamic, interactive environment on a high-performance engineering workstation and to be able to shorten the design/evaluation cycle as well as optimize the operations.

4.3 IGRIP

IGRIP^{TM} (Interactive Graphics Robot Instruction Program from Deneb Robotics, Inc.) is a user-friendly computer graphics based simulation system for workcell layout, simulation and off-line programming (OLP). Parts modeled within the Part Modeler (CAD Context) are put together to define Devices with multiple degrees of freedom. A Device has both geometric and non-geometric information stored with it. Non-geometric information like kinematics, dynamics, velocities etc. can be entered through interactive menus. A workcell is composed of devices, positioned relative to each other (WORKCELL Context). Devices may be selected from library of robots, conveyors, and effectors or modeled by the user in the Device context. IGRIP has the capability to generate robot programs interactively (MOTION Context). Several Devices may be simulated simultaneously with Input/Output signaling between them.

4.3.1 Invocation

The IGRIP simulation system is invoked by changing to the /user/deneb/ igrip.4d directory, where the igrip executable file resides. Enter **'cd/usr/deneb/igrip.4d'** after logging in to the SGI (Silicon Graphics) workstation. Enter **'igrip -f'** to invoke the fullscreen mode of IGRIP. At this point IGRIP's main menu should appear.

The IGRIP menu system is divided mainly into Contexts, which are arranged across the top of the IGRIP screen, each which has a group of subdivisions called Pages. The Contexts are :

1. CAD The CAD Context allows the user to create and modify 3-D surface or wireframe geometry used to represent parts.
2. DEVICE he DEVICE COntext allows the user to build and modify devices by putting together the Parts built in the CAD context.
3. LAYOUT The LAYOUT Context allows the user to lay out a workcell. This includes positioning Devices, creating Paths for motion definition, connecting I/O signals and creating Collision Queues.
4. MOTION The MOTION Context allows the user to define and execute motion for Devices. Motion can be commanded interactively or through Program control (when running a simulation). Simulation programs can also be downloaded to specific controller or generic formats.
5. DIMENSION The DIMENSION Context allows the user to create and manipulate various kinds of dimension entities to document workcell layouts and geometric data. Dimensions are fully three dimensional planar entities, and can be translated and rotated in space with respect to coordinate system that is local to the dimension. Dimensions are also dynamically associative, or "data-driven", which implies that the dimensions are attached to geometry, and are continuously updated to reflect the current state of that geometry.
6. USER The USER context allows customization of the user interface to define custom Menu Pages with functions taken from other Pages, or functions to invoke CLI (Command Line Interpreter) macro files.
7. ANALYSIS The ANALYSIS Context allows user to perform various forms of analysis. Functions on the MEASURE Page allow identification of various items in world , as well as the determination of the distances and the angles between them. The units for

reporting as well as the frame of reference may be set by the user. Entity properties such as area and volume may be queried using functions on the PROPERTIES Page. All analysis functions utilize "Analysis Registers" that can be used in conjunction with IGCALC. Some of the Registers and the data values they represent are :

- c: Value of the current Popup field
- p: Last value entered
- x, y, z: x-, y-, and z-coordinate of a point respectively
- dx, dy, dz: Distance in the x-, y-, and z-direction respectively
- d: Total Cartesian distance
- V: Object volume
- A: Object area
- dia: Polygon diameter
- ang: Angle between entities
- R, P, Y: Pitch, and Yaw angle about Z-, Y-, and X-axis respectively.

8. SYSTEM The SYSTEM Context provides system utilities to modify the system environment and world attributes as well as to interact with UNIX file system.
9. CLI The CLI (Command Line Interpreter) Button is used to enter CLI commands interactively. This enables the expert user to type in a command from any Context without switching to the relevant Context.

4.3.2 CAD

Before beginning the design of Parts, the units should be set up (if other than default , mm, is desired) as below :

1. Select the ANALYSIS context. Select the UNITS Button and enter the new units in the Popup (or use the LMB to select from the choices),. The new units will be used for all the subsequent operations.

2. To actually design the Parts, select the CAD context and click on the CREATE Button to go to the CREATE page. The CAD context is used to model the geometry used to design Parts, which consist of one or more Objects which in turn comprise one or more Subobjects composed of Lines and Polygons.
3. Objects are created using the CAD primitive Block, Cylinder, Cone, Wedge, Pipe, and sphere. These objects are modified using the CAD operators such as Cut, Mirror, Loft, Clone, Extrude & Revolve. From MODIFY page, use Merge, Smash, Scale, Cut, Color Object, and Extract Obj to further modify the Objects. From Auxiliary page, create Coorsys to assist in attaching subobjects to make up an Object.

The places (blue circles), transitions (green bars), arcs (yellow lines), enabled transitions (red bars), and tokens (red dots) are all constructed at the Systems cordinate system using the CAD primitives, defining the required dimensions as shown in Figure 4.1. Once the required number of objects are created, cloned, translated and roated, they are saved at the it the coordinate system.

4.3.3 DEVICE

To build a new Device , use the following procedure:

1. Select the DEVICE Context, then the NEW DEVICE Button. You will be placed in the syslib/PARTS directory. Select the appropriate directory on the Popup to move to your directory.
2. Select the part to be used as the base of the new device.
3. Enter a name for the Device and accept the defaults for the Device parameters in the Popup.
4. Select the AXES Button, then pick the Part that was just retrieved, to force the coorsyses to always stay visible.

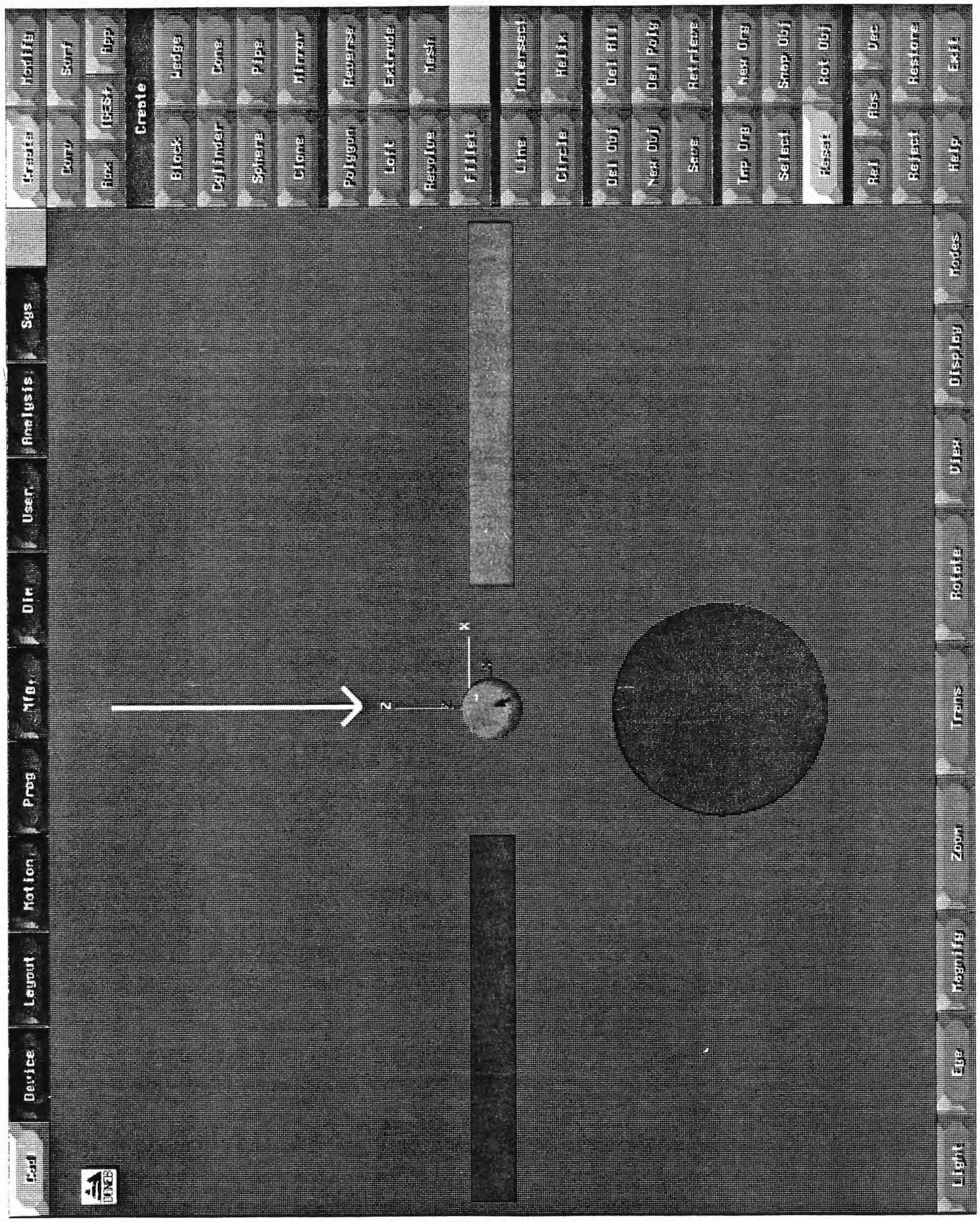


Figure 4.1: CAD primitives of IGRIP used for modeling

5. Select the ATTACH PART Button and then pick the base of the Part to indicate the Part that the new Part will be attached onto.
6. Choose the other Parts and then the key Coorsyses, placed in the CAD Context to facilitate easy positioning of the Parts.
7. Select the KIN Page, and select the JOINT TYPES Button. This is where the Translational vs Rotational dof are assigned at each joint. Change these first three to be Translational and accept the default, Rotational for other three joints.
8. Select the SET DOF Button and pick the Part which is supposed to move along X- and Y-axis. The "Link Transformation" Popup is used to describe how the Joint should move. The most common choices are to Translate along an axis, or to Rotate about an axis.
9. First specify " Set Home", next specify "Trans X" enter a 1 for the
10. Translate X Expr:" finally select "Return". The "Set Home" option indicates that the system should use the current location and orientation as the "zero" position whencalculating the location. The "Trans X" option defines motion along the Part's X axis. This motion is tied to degree of freedom number 1. In other words, Joint 1 is a Translational Joint that moves positive in the parts positive direction. The "Return" choice completes the DOF definition. It is possible to have more than one DOF number for the same part, as also to have one DOF number control motion for many Parts, in many different directions, as well as mixing types of motion.
11. Select the other Parts and repeat the above except choose "Trans Y", "Trans Z", "Rotate Z", "Rotate Y", and "Rotate X" with DOF numbers 2,3,4,5 and 6 respectively.
12. Now select the KINEMATICS Button and choose the "Inverse Kinematics" option from the Popup.

To model a new device the kinematics of any existing Devices can be used, in the following conditions are satisfied:

1. The base coordinates systems for each part on each device must match exactly. The Positive/negative directions of rotation must be identical.

2. The number of dofs for both devices must be same.
3 The type of dof must be the same in both Devices(i.e, ROTATIONAL vs TRANSLATIONAL). Selecting 'Device Kinematics' from the displayed Popup allows the user to select any device existing in the DEVICE directory. The Device being built will assume the kinematics math-routine defined for the Device name selected from the file-list Popup. Note that if a theparameters mentioned above (link lengths, link offsets, link types, mounting plate offset, or DOF definition) do not match those of the selected Device, the new Device will not be able to reach its points.

13. Select JOINT LENGTHS. This is where the D.H parameters (The Denavit Hartenberg notation is used to represent the robot kinematically using link lengths and offsets based on the coordinate system of each link fixed at arbitrary locations i.e, lengths between base coordinate systems as well as offsets from the principal lane) are assigned.
14. Select BASE PRT and pick the Part of the Device which is serve as the base. Use UFRAME under the MOTION Context to verify by picking the 'Display' option from the Popup, that the UFRAME is the base Coorsys of the Base Part.
15. Select the MNT PLT Button. Here the user graphically selects the part which represents the device mounting plate and defines the offset values. Use UTOOL Button under MOTION Context to set Tool Point. Pick 'Display' from the Popup and verify the Tool Point.
16. Select HOME POSITION Button and set the home position to be the current Joint value by choosing the "Use Current Position" option in the Popup.
17. Select the SPEEDS Button located under the LIMITS Title and complete the Popup. Select the ACCELS Button and set the maximum accelerations.
18. Select the TRAVEL Button and set the travel limits.
19. This completes the definition of the new Device. If any modifications are necessary,

use REDEFINE DEVICE Button and make the requisite changes in the Popup.

20. Finally, select SAVE DEVICE Button and complete the Popup with the name of your directory and the filename.

In this module, the parts created in the CAD module are retrieved and the required number of parts are cloned, rotated, translated inorder to construct the Petri net as shown in Figure 4.2. Also the enabled transitions (red bars) and the tokens are retrieved from the CAD module, then they are translated, roated and saved at respective positins. Kinematics are applied to all the parts retrieved in the DEVICE module.

4.3.4 WORKCELL

To layout a Workcell, follow the steps below:

1. Select the LAYOUT Context and the WORKCELL Page.
2. Select the RETRIEVE DEVICE Button. Choose your directory from the Popup, and pick the relevant Device.
3. Select the AXES Button; this will toggle the Device's display mode so that it's Coorsyses are always displayed.
4. Select the RETRIEVE DEVICE Button again to pick the other Devices to be laid out in the Workcell.
5. Select TRN DEV or ROT DEV to arrange the Devices in the Workcell. The SNP Button is a quick way to do 90 degree rotations about the primary axis. The LMB rotates about X, MMB about Y, and the RMB about the Z axis. If any Devices are to be attached to another Device, select SNAP DEV Button. Choose the 'Frame' option from the "Snap Device On..." and pick the Coorsyses on the Device.
6. Select the ATTACH Button using the MMB(Middle Mouse Button). This will give a list of all the Devices that are currently in the Workcell. Choose a device and pick a part

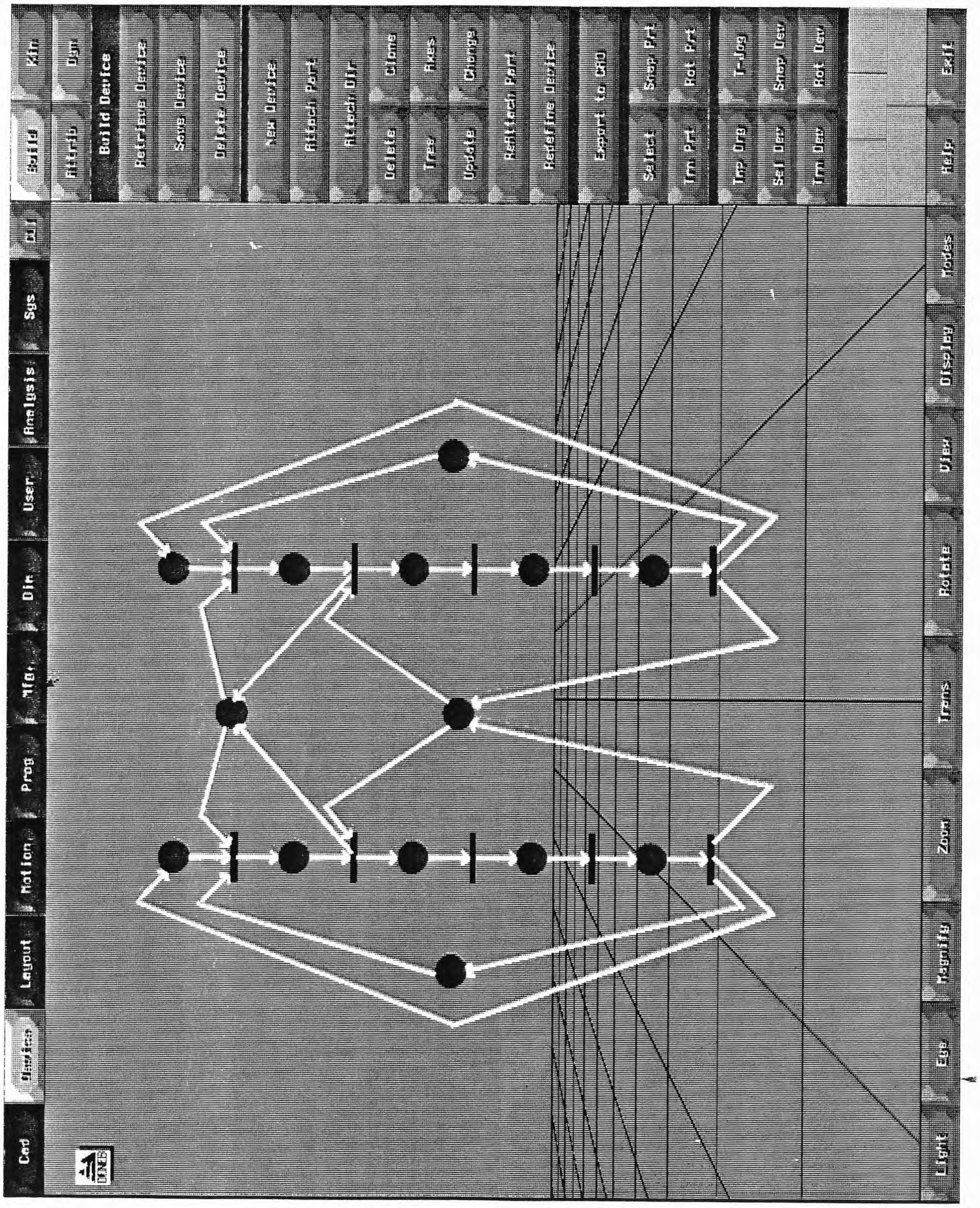


Figure 4.2 : Petri net model as Device

on the Device to attach it to. The part should highlight and any Coorsyses, if present, will appear. Pick the right Coorsys and the Device will snap onto the part using the orientation of the Coorsys.

7. At this point, the locations and orientations of each Device should be saved. Move to the SYSTEM Context, WORLD Page, and pick the SAVE POSITIONS Button. Choose the "All Devices" option from the "Save/Restore Positions" Popup. This establishes "Restore Positions" for the location and the Joint values of each Device. This can be used as the starting point when running simulations. If this is not done, when the simulation is RUN using "Previous Values" all the Devices jump to the World Origin.
8. Use the World Display functions to move to a view of the Workcell that shows most of the Devices, and save the Workcell using SAVE WORKCELL Button.

In this module all the devices created in the DEVICE module are retrieved and arranged as a layout by using the translation and rotation option. The workcell is saved as per the arrangement of the devices as shown in Figure 4.3.

4.3.5 Tag Points

Tag Points are primarily used to indicate destination positions for robot motion. The user places Tag Points at the desired location and orientation and then, instructs the robot to move to the Tag Point position.

Tag Points may be set up as follows:

1. Select the LAYOUT Context and then the RETRIEVE WORKCELL Button. Choose the Workcell from the Popup.
2. Select the TAGS Page and then the NEW PATH Button, pick the Device to attach the Path to.

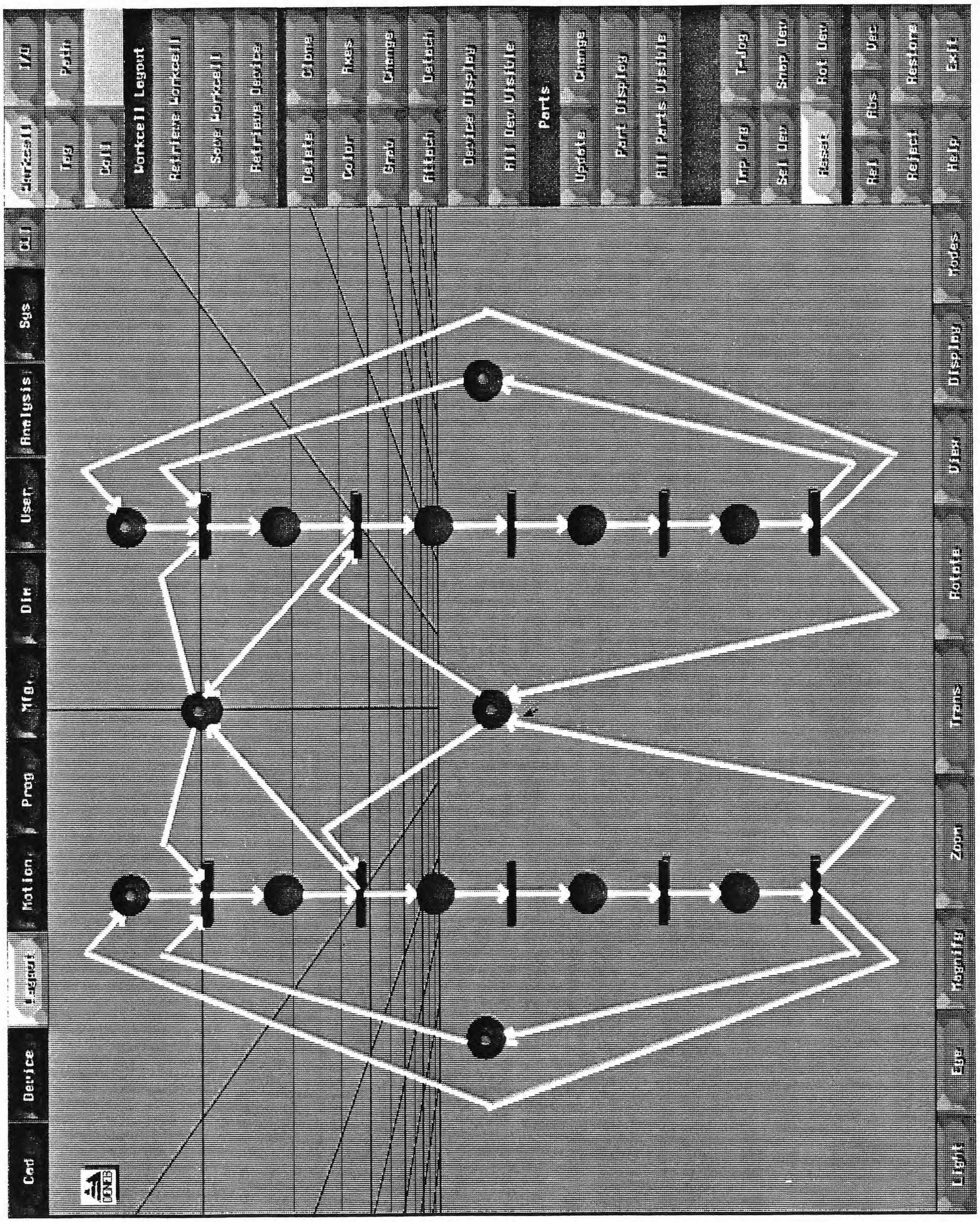


Figure 4.3 : Petri net model in the Workcell

3. Select the SETUP Button and complete the Popup. This allows you to constrain or free the Degree of Freedom.
4. Select the SURFACE Button and then, using the LMB(Left Mouse Button), pick the surface to snap the Tag Point onto. You may also select the VERTEX, EDGE, FRAME Buttons as appropriate.
5. If the orientation of the most recently created Tag Point is desired for subsequent Tag Point placements, pick the surface with the RMB(Right Mouse Button). This will place a new Tag Point at that position with the same orientation as the previous. If your Tag Point names end in integer numbers, the new Tag Point will be added to the current Path and given the next available ending number. If only part of the Tag Point is visible, part of it is hidden inside of the polygon. You may want to go to the SYSTEM Context and select the Z-BUFFER Button, it should dehighlight(which means the real time Z-Buffer is turned off) and all of the Tag Point axes become visible.

This completes the Path layout. To check on the reachability of these Tag Points, select the T-JOG Button. Pick the Device(Robot) to be T-Jogged when the Tags are moved. Now the Device will move to any Tag Point, align itself to the Tag Point using its Utool, and follow it. The Tag Points can be selected one by one to check position and orientation. If needed, any changes may be using TRN TAG and ROT TAG Buttons.

The tag points are attached to the devices in the workcell by selecting individual devices, new path and autotag commands. Figure 4.4 shows the tag points attached to the devices in the workcell.

4.3.6 Input/Output Signals

To layout the I/O connections:

1. Select the LAYOUT Context and the WORKCELL Page. Select the RETRIEVE

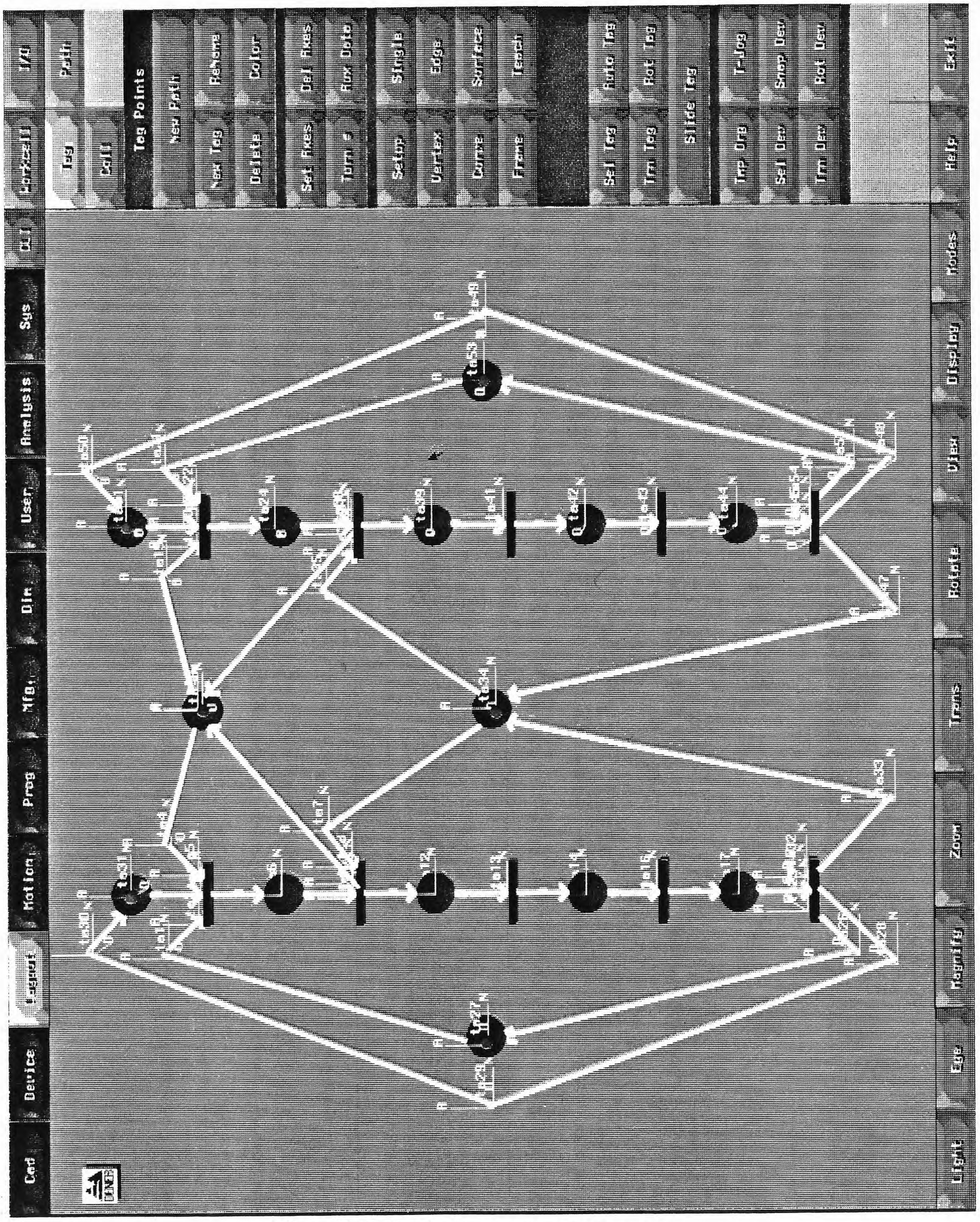


Figure 4.4 : Petri net model with Tag points

WORKCELL Button and retrieve your Workcell from your directory.

2. Select the I/O Page
3. Select the DUAL CONNECTION Button to allow signals to be sent both directions. Pick the first Device and select I/O 01 as the line you want to connect to it. Now pick the second Device to connect 01 to. Select I/O 01 as the corresponding file.
4. Select DISPLAY CONNECTION Button and pick the Device whose signals you want to display. The Popup will show the Device's input 01 coming from the other Device and it's output 01 going to the other.

All the devices are given dual input/output connections, to assertain that all the operations are performed sequentially.

4.3.7 PROGRAM

The PROGRAM Page in the MOTION Context is primarily used for Program Scripting. This is the process of automatically scripting program statements with correct syntax to a GSL program using menu Buttons. The program statements are executed when they are scripted so that you can interactively see the effect of each statement. To use the PROGRAM Page for program writing:

1. Select NEW PROGRAM, pick the Device to be programmed and enter the program name. A Program Edit Window should appear with a basic program template in it.
2. Select the SYSTEM VARS Button, set the variables desired by choosing the UNITS,Speed, Motype options.
3. Select the MOVE Button and choose the "Move To" option. Pick the Tag Point to be moved to and the Device should move to it. If you can't see the Tag Point to pick it, select the "Move To" option using the MMB(Middle Mouse Button) and the system will let you select the Tag Point from the list of all available Tag Points.

4. You can also add Routines or Procedures, If, While, For conditions to your program by picking the function Buttons. You can enter text like"sim-update" using the GSL Button and the "Enter Text" option. I/O statements may be added by means of the I/O Button.
5. To see the program run, move the mouse up in the file until the highlighted line is the UNITS=METRIC line. Select the EXECUTE Button and watch the system step through the program.
6. If the program runs satisfactorily, select the WRITE Button located on the left side of the Program Edit Window. Save the program into your directory.

In this module, individual device is selected and the required program is return as listed in the appendix.

4.3.8 MOTION

1. Select the MOTION Context, and the SIMULATE Page.
2. Select the RETRIEVE WORKCELL Button and pick the desired Workcell from the Popup.
3. Select Load using the MMB(this displays a list of all the Devices in the Workcell). Pick the Device to be loaded, choose the "Load Selected Program"option from the Popup and pick the program to be loaded into it. If the message window doesn't read "Program x x x.gsl successfully loaded", there are errors in the Program; choose the "yes"option from the Edit Program window. The igedit window will appear to allow you to debug the Program(If vi editor is preferred, change the editor option to vi by picking the Environ Button under the WORLD Page in the SYSTEM Context).
4. Select the STEPSIZE Button and change the "Simulation Step Size" to be 0.2 seconds, and "Steps per graphic update" to 1. This means that the system will

calculate and display the simulation at 0.2 second intervals, and update and display the graphics every step.

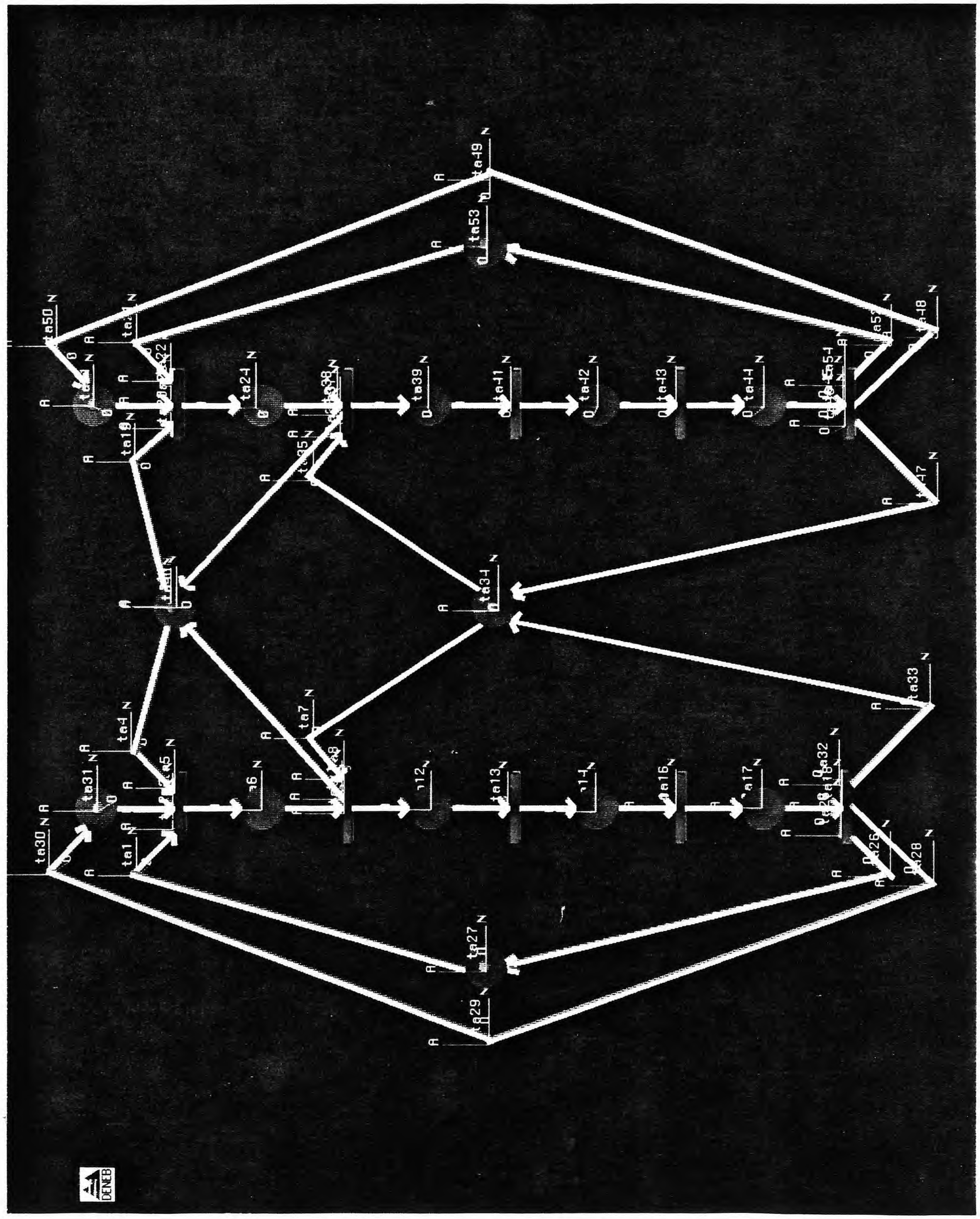
5. Select the ACTIVATE Button and choose the "All Devices" option to activate all the Devices that have GSL programs loaded into them.
6. Select the RUN Button using the RMB to skip the Popup. This will use the "Previous Values" for the Device locations, and their Joint positions.
7. The World Display Buttons on the bottom of the screen may be selected and used any time during a simulation run.
8. To inspect the simulation while it is running, select the CYCLE Button, and then pick the Device. Choose the "Cycle Time On" option to display a Popup that indicates the current Cycle Time for the Device. Similarly, to see a display of the Joint values select the JNT VALS Button.

In this module, all the programs are loaded and the devices are activated to see the operation of the Petri net model.

4.4 Simulation Set-up:

The simulation set-up is as shown in Figure (4.5). The places (blue circles), transitions (green bars), arcs (yellow lines) are contructed as one device. The enabled transitions (red bars) and the tokens (red dots) are constucted as individual devices. The operation of the net is as explained in chapter 3. For any transition in the net to fire all the tokens must be present at the transition, if any of the token is not available, the trasition does not fire.

After all the devices are retrieved into the workcell, the home positions are restored. If one desires a demonstration run is provided which gives one a better idea of how the Petri net model for Flexible Work Station operates, thus making the choices to



Figur 4.5 : Final Petri net model for Simulation

answer the querries apperent. If one desires to get on with the simulation, the demo can be hurried along by increasing '>>' Button which dobules the step size.

4.5 Running the Simulation :

After logging into the SGI Workstation :

1. Move to the IGRIP directory by entering "cd /usr/deneb/igrip.4d"
2. Invoke full screen mode of the IGRIP by entering "igrip -f"
3. Select the MOTION context and then the SIMULATE page.
4. Select RETRIEVE WORKCELL button, select "suresh", and then from the popup, pick "test".
5. Select MOTION context.
6. Select LOAD, pick up each component and then pick the appropriate program from the popup.
7. Select ACTIVATE and choose "All Devices" option from the popup.
8. Select RUN to get the simulation running.

The block diagram of the simulation run is as shown in Figure 4.6

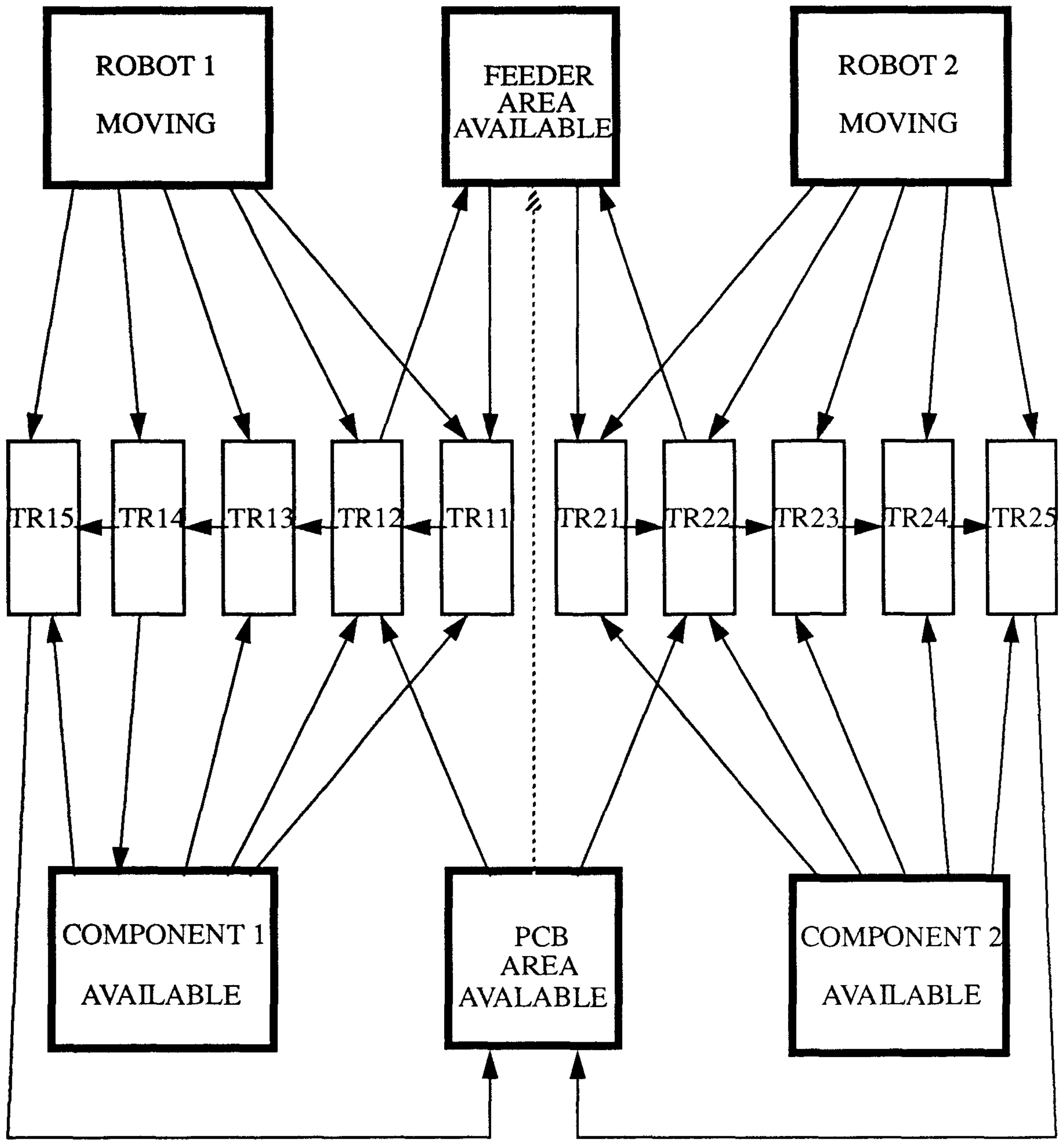


Figure 4.6 : Block diagram for Simulation Run

CHAPTER
5

CONCLUSIONS AND FUTURE RESEARCH

This chapter briefly outlines the contributions of this thesis and dicusses the results obtained. Therefore some suggestions to enhance the work done are given and directions for future work, in the application of Petri nets for modeling, analysis and simulation of manufacturing systems are outlined.

5.1 CONCLUSIONS

This thesis is a tutorial on Petri nets and their models, and the begining of real time simulation of manufacturing systems. The concepts of how the events and conditions of a system can be mapped to the transitions and places of a Petri net model have been illustrated with examples. The execution of Petri net is dicussed along with some important properties of Petri nets. Also, simulation can be successfully used to study a typical robotic workcell operation by observing such performance criteria as cycle time. Cycle time analysis can help the engineer in searching an optimum of the various parameters such as speed of manipulator offset, usage of two types of manipulators and the chip placement sequence.

The mathematical model of a flexible manufacturing cell has been built using Petri nets. The concurrency, resource-sharing, and sequential operations have been built in this mode. The resulting net model has the system properties such as liveness, boundedness, and reversibility. Furthermore the deterministic timing information is incorporated into the places and transition of this model.

The simulation program allows a layman to run the FWS without prior knowledge of either M^{2}L or the FWS. IGRIP is a very effective simulation package from the point of view of analysis. The models of actual devices such as robots and automatic guided vehicles can be visualized in great deal. It is possible to observe the rechability of various devices. Collisions between the component devices in a workcell can be detected. Optimizations can be carried out based on cycle time analysis.

New products can be analyzed before purchase to determine suitability for the desired purpose. Sequences of the animated display can be put on to video tape for future reference. Pictures of layouts can be printed directly on to a color printer providing the designer with a series of drawings.

5.2 DIRECTIONS FOR FUTURE RESEARCH

The popularity of Petri nets to model, analyze, design and control of automated manufacturing systems has grown multiple-fold in the past five years. This is mainly because Petri nets have a great potential to overcome many difficuties encoun teered in complex automated manufacturing systems. This should allow for true flexibility and integration. Evidently a lot of work can be done and thus, has to be one in this field. This thesis provides industrial engineers and academic researchers with a comprehensive real life example of applying Petri net theory to help them develop their own applications.

Off-line programming of the simulated model can be implemented by downloading, and thus it is obvious that the next step is to reverse the process. Uploading is the process of carrying out an actual run on the FWS and then sending over the data to the simulation system to enable real-time simulation update and any required modifications.

Real-time control of the FWS can be implemented using two way communication. Every motion of the FWS model on the terminal should be replicated by the FWS on the factory floor. In this case, every step of the simulation run should instantly downloaded to the FWS instead of downloading the entire sequence of motions at one time. At the other end, the FWS should send a signal on completion of motion replecation to the simulation system. On the other hand, a motion of the FWS on the floor could be replicated on the terminal.

REFERENCES

[1] J. L. Peterson, "Petri nets", *ACM Computing Surveys*, Vol. 9, pp. 223-252, September 1977.

[2] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Inc., Engle wood Cliffs, NJ, 1981.

[3] C. L. Beck and B. H. Krogh, "Models for simulation and discrete control of Manufacturing Systems," *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, San Francisco, CA, pp.305-310, April 1986.

[4] B. H. Krogh and R. S. Sreenivas, "Essentially decision free Petri nets for real time resource allocation," *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, Raleigh, N.C., pp. 1005-1011, April 1987.

[5] C. Ramchandani, "Analysis of asynchronous concurrent systems by timed Petri nets," Ph.D. dissertation, MIT, Cambridge, Project MAC Rep. MAC-TR-120, 1974.

[6]J. Sifakis, "Structural properties of Petri nets," *Mathematical Foundations of Computer Science, Lecture notes in Computer Science*, No. 64, Springer-Verlag, Berlin, FRG, pp. 474-483, 1978.

[7] M. Hack, "Analysis of production schemata by Petri nets," Technical Report 94, MIT, February 1972.

[8] D. Dobois and K. Stecke, "Using Petri nets to represent production processes," *Proceedings of the 22nd IEEE conference on Decision and Control*, San Antonio, TX, pp. 1062-1067, 1983.

[9] Y. Narahari and N. Viswanadham, "A Petri net approach to the modeling and analysis of Flexible Manufacturing Systems," *Annals of Operations Research,* vol. 3, pp. 449-472, 1984.

[10] M. C. Zhou, K. J. McDermott, P. Patel, and D. Tang, "Construction of Petri net based mathematical models of an FMS cell," *Proceedings of 1991 IEEE Int. Conference on Systems, Man, and Cybernetics*, Charlottesville, Virginia, pp. 146-151, October 1991.

[11] M. C. Zhou, F. DiCesare and A. A. Desrochers, "A top-down modular approach to systematic synthesis of Petri net models for manufacturing systems," *Proceedings of IEEE Robotics and Automation Conference*, Scottsdale, AZ, pp. 534-539, May 1989.

[12] H. P. Hillion, "Performance evaluation of decisionmaking organizations using timed Petri nets," Master's Thesis Report LIDS-TH-1590, Laboratory for Information and Decision Systems, MIT, Cambridge, MA., August 1986.

[13] T. Murata, "Petri nets: properties, analysis and applications," *Proceedings IEEE,* vol. 77, no. 4, pp. 541-579, April 1989.

[14] M. C. Zhou and F. DiCesare, "Adaptive design of Petri net controllers for error recovery in automated manufacturing systems," *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-19, 5, pp. 963-973, 1989.

[15] IGRIP *"Simulation System User Manual"*, Deneb Robotics Inc., Version 2.0, Mar '90.

[16] K. H. Lee, J. Favrel, and P. Baptiste, " Generalized Petri net reduction method," *IEEE Trans. Systems, Man, and Cybernetics*, SMC-17, No. 2, pp. 297-303, 1987.

[17] J.L Petrson, "Petri Nets", *ACM Computing Surveys,* Vol. 9, p. 223-252, September 1977.

[18] M. C. Zhou, *A Theory for the Synthesis and Augmentation of Petri Nets in Automation,* Doctoral Dissertation, ECSE, Rensselaer Polytechnic Institute, Troy, NY, 1990.

[19] M.C. Zhou and Ming C. Leu, "Petri net modeling of a flexible assembly station for printed circuit boards," *Proceedings of IEEE Int. Conference on Robotics and Automation,* Sacramento, CA, pp. 2530-2535, April, 1991.

[20] D. Crockett, A. A. Desrochers, F. DiCesare, and T. Ward, "Implementation of a Petri net controller for a machining workstation," *Proceedings of IEEE Int. Conference on Robotics and Automation,* Raleigh, NC, pp. 1861-1867, 1987.

[21] M.C. Zhou and f. DICesare, "A Petri Net design method for automated manufacturing systems with shared resources," *Proceedings of IEEE Int. Conference on Robotics and Automation,* pp. 526-531, Cincinnati, OH,1990.

[22] Y. C. Ho, "Performance evaluation and perturbation analysis of discrete event dynamic systems," *IEEE Trans. on Automatic Control,* Vol. AC-32, No. 7, pp. 563-572, 1987.

[23] M. Molloy, "Performance analysis using stochastic Petri nets," *IEEE Trans. Comput.*, vol. C-31, no. 9, pp. 913-917, 1982.

[24] D. L. Guo, F. DiCesare, and M. C. Zhou, "A moment generating function-based approach for evaluating extended stochastic Petri nets," Accepted for *IEEE Trans. Automatic Control,* 1991.

```
:::::::::::::
ROBOT1
:::::::::::::

PROGRAM token1
VAR
---------------------------- Main Declaration Section ------------------------------
ta27  : POSITION
ta26  : POSITION
ta25  : POSITION
ta18  : POSITION
ta17  : POSITION
ta16  : POSITION
ta14  : POSITION
ta13  : POSITION
ta12  : POSITION
ta10  : POSITION
ta9  : POSITION
ta6  : POSITION
ta3  : POSITION
ta2  : POSITION
ta1  : POSITION
------------------------------------------------------------------------------------

BEGIN MAIN
------------------------------------------------------------------------------------
 $speed = 2000
 MOVE TO ta1
 MOVE TO ta2
 WAIT UNTIL DIN[ 1 ] == ON
 WAIT UNTIL DIN[ 2 ] == ON
 $speed = 500
 MOVE TO ta3
 GRAB 'Dummy#3' AT LINK 1
 GRAB 'Dummy#2' AT LINK 1
 DOUT[ 3 ] = ON
 delay 1500
 MOVE TO ta6
 delay 3000
 DOUT[ 6 ] = ON
 delay 3000
 MOVE TO ta9
 WAIT UNTIL DIN[ 4 ] == ON
 DOUT[ 5 ] = ON
 DOUT[ 11 ] = ON
 delay 4000
 MOVE TO ta12
 delay 3000  WAIT UNTIL DIN[ 7 ] == ON
 MOVE TO ta13
 DOUT[ 8 ] = ON
 delay 1500
 MOVE TO ta14
 delay 3000
 DOUT[ 16 ] = ON
 DOUT[ 15 ] = ON
 DOUT[ 14 ] = ON
 WAIT UNTIL DIN[ 13 ] == ON
 MOVE TO ta16
 DOUT[ 12 ] = ON
 delay 1500
 MOVE TO ta17
 delay 3000
 MOVE TO ta18
 DOUT[ 10 ] = ON
 DOUT[ 0 ] = ON
 delay 1500
 RELEASE 'Dummy#3'
 RELEASE 'Dummy#2'
 $speed = 1500
 MOVE TO ta25
 MOVE TO ta26
 MOVE TO ta27
 delay 3000
---------------------------------------END MAIN-------------------------------------
END token1
```

```
:::::::::::::
COMPONENT1
:::::::::::::

PROGRAM token11
VAR
---------------------------- Main Declaration Section--------------------------------
ta31  : POSITION
ta30  : POSITION
ta29  : POSITION
ta28  : POSITION
ta3  : POSITION
-------------------------------------------------------------------------------------

BEGIN MAIN
-------------------------------------------------------------------------------------
 $speed = 500
 MOVE TO ta3
 DOUT[ 1 ] = ON
 WAIT UNTIL DIN[ 0 ] == ON
 $speed = 3000
 delay 1500
 MOVE TO ta28
 MOVE TO ta29
 MOVE TO ta30
 MOVE TO ta31
 DOUT[ 4 ] = ON
 delay 3000
------------------------------------ END MAIN ------------------------------------
END token11
:::::::::::::
```

TRANSITION11

```
::::::::::

PROGRAM fire
VAR
----------------------------- Main Declaration Section---------------------------------

BEGIN MAIN
---------------------------------------------------------------------------------------
 WAIT UNTIL DIN[ 3 ] == ON
 MOVE JOINT 2 BY -.2 NOSIMUL
 delay 1000
 MOVE JOINT 2 BY 0.2 NOSIMUL
------------------------------------ END MAIN -----------------------------------------
END fire
:::::::::::::
```

TRANSITION12

```
:::::::::::::

PROGRAM firet12
VAR
---------------------------- Main Declaration Section---------------------------------

BEGIN MAIN
-------------------------------------------------------------------------------------
 WAIT UNTIL DIN[ 6 ] == ON
 WAIT UNTIL DIN[ 5 ] == ON
 MOVE JOINT 2 BY -0.2 NOSIMUL
 delay 1000
 MOVE JOINT 2 BY 0.2 NOSIMUL
---------------------------------- END MAIN -----------------------------------------
END firet12
:::::::::::::
```

TRANSITION13

```
:::::::::::::

PROGRAM fireta3
VAR
------------------------------ Main Declaration Section-------------------------------

BEGIN MAIN
---------------------------------------------------------------------------------------
 WAIT UNTIL DIN[ 8 ] == ON
 MOVE JOINT 2 BY -0.2 NOSIMUL
 delay 2000
 MOVE JOINT 2 BY 0.2 NOSIMUL
------------------------------------ END MAIN ----------------------------------------
END fireta3
:::::::::::::
```

TRANSITION14

```
:::::::::::::

PROGRAM firetal4
VAR
------------------------------- Main Declaration Section-----------------------------

BEGIN MAIN
-------------------------------------------------------------------------------------
 WAIT UNTIL DIN[ 12 ] == ON
 MOVE JOINT 2 BY -0.3 NOSIMUL
 delay 1000
 MOVE JOINT 2 BY 0.3 NOSIMUL
------------------------------------- END MAIN -------------------------------------
END firetal4
:::::::::::::
```

TRANSITION15

```
:::::::::::::

PROGRAM fireta15
VAR
----------------------------- Main Declaration Section------------------------------

BEGIN MAIN
----------------------------------------------------------------------------------
 WAIT UNTIL DIN[ 10 ] == ON
 MOVE JOINT 2 BY -0.3 NOSIMUL
 delay 1000
 MOVE JOINT 2 BY 0.3 NOSIMUL
------------------------------------- END MAIN ------------------------------------
END fireta15
:::::::::::::
```

ROBOT2

```
:::::::::::::

PROGRAM token2
VAR
------------------------------- Main Declaration Section----------------------------
ta53  : POSITION
ta52  : POSITION
ta54  : POSITION
ta42  : POSITION
ta43  : POSITION
ta45  : POSITION
ta44  : POSITION
ta41  : POSITION
ta39  : POSITION
ta38  : POSITION
ta36  : POSITION
ta24  : POSITION
ta23  : POSITION
ta11  : POSITION
ta22  : POSITION
ta21  : POSITION
------------------------------------------------------------------------------------

BEGIN MAIN
------------------------------------------------------------------------------------
 WAIT UNTIL DIN[ 15 ] == ON
 $speed = 2000
 MOVE TO ta21
 MOVE JOINT 2 BY -.2 NOSIMUL
 MOVE TO ta22
 WAIT UNTIL DIN[ 0 ] == ON
 WAIT UNTIL DIN[ 1 ] == ON
 $speed = 500
 MOVE TO ta23
 GRAB 'Dummy#5' AT LINK 1
 GRAB 'Dummy#7' AT LINK 1
 DOUT[ 2 ] = ON
 delay 1500
 MOVE JOINT 2 BY 0.2 NOSIMUL
 MOVE TO ta24
 delay 3000
 DOUT[ 13 ] = ON
```

```
WAIT UNTIL DIN[ 4 ] == ON
 MOVE TO ta38
 MOVE JOINT 2 BY -0.2 NOSIMUL
 WAIT UNTIL DIN[ 10 ] == ON
 DOUT[ 5 ] = ON
 delay 2000
 MOVE TO ta39
 delay 3000
 MOVE TO ta41
 DOUT[ 7 ] = ON
 delay 1500
 MOVE TO ta42
 delay 3000
 MOVE TO ta43
 DOUT[ 8 ] = ON
 delay 1500
 MOVE TO ta44
 delay 3000
 MOVE TO ta45
 DOUT[ 11 ] = ON
 DOUT[ 12 ] = ON
 delay 2500
 RELEASE 'Dummy#7'
 RELEASE 'Dummy#5'
 DOUT[ 9 ] = ON
 MOVE TO ta54
 $speed = 1000
 MOVE TO ta52
 MOVE TO ta53
 delay 3000
------------------------------------END MAIN----------------------------------------
END token2
:::::::::::::
```

COMPONENT2

```
::::::::::::

PROGRAM token21
VAR
------------------------------ Main Declaration Section-----------------------------
ta51  : POSITION
ta50  : POSITION
ta49  : POSITION
ta48  : POSITION
ta23  : POSITION
-------------------------------------------------------------------------------------

BEGIN MAIN
-------------------------------------------------------------------------------------
 $speed = 500
 WAIT UNTIL DIN[ 14 ] == ON
 MOVE JOINT 2 BY -0.4 NOSIMUL
 MOVE TO ta23
 DOUT[ 1 ] = ON
 WAIT UNTIL DIN[ 11 ] == ON
 $speed = 3000
 delay 3000
 MOVE TO ta48
 MOVE JOINT 2 BY 0.4 NOSIMUL
 MOVE TO ta49
 MOVE TO ta50
 MOVE TO ta51
 delay 3000
------------------------------------ ENDMAIN  ------------------------------------
END token21
::::::::::::
```

TRANSITION21

```
:::::::::::::

PROGRAM firet21
VAR
-------------------------------- Main Declaration Section----------------------------

BEGIN MAIN
------------------------------------------------------------------------------------
 WAIT UNTIL DIN[ 2 ] == ON
 MOVE JOINT 2 BY -0.3 NOSIMUL
 delay 1000
 MOVE JOINT 2 BY 0.3 NOSIMUL
-------------------------------------- END MAIN ----------------------------------
END firet21
:::::::::::::
```

TRANSITION22

```
:::::::::::::

PROGRAM fireta22
VAR
------------------------------- Main Declaration Section----------------------------

BEGIN MAIN
------------------------------------------------------------------------------------
 WAIT UNTIL DIN[ 16 ] == ON
 WAIT UNTIL DIN[ 5 ] == ON
 MOVE JOINT 2 BY -0.3 NOSIMUL
 delay 1000
 MOVE JOINT 2 BY 0.3 NOSIMUL
------------------------------------- END MAIN ------------------------------------
END fireta22
:::::::::::::
```

TRANSITION23

```
:::::::::::::

PROGRAM fireta23
VAR
--------------------------------- Main Declaration Section----------------------------

BEGIN MAIN
---------------------------------------------------------------------------------------
 WAIT UNTIL DIN[ 7 ] == ON
 MOVE JOINT 2 BY -0.3 NOSIMUL
 delay 1000
 MOVE JOINT 2 BY 0.3 NOSIMUL
-------------------------------------- END MAIN ------------------------------------
END fireta23
:::::::::::::
```

TRANSITION24

```
:::::::::::::

PROGRAM fireta24
VAR
------------------------------ Main Declaration Section-----------------------------

BEGIN MAIN
------------------------------------------------------------------------------------
 WAIT UNTIL DIN[ 8 ] == ON
 MOVE JOINT 2 BY -.3 NOSIMUL
 delay 1000
 MOVE JOINT 2 BY 0.3 NOSIMUL
------------------------------------ END MAIN -------------------------------------
END fireta24
:::::::::::::
```

TRANSITION25

```
::::::::::::::

PROGRAM fireta25
VAR
-------------------------------- Main Declaration Section------------------------------

BEGIN MAIN
-----------------------------------------------------------------------------------------
 WAIT UNTIL DIN[ 9 ] == ON
 MOVE JOINT 2 BY -0.3 NOSIMUL
 delay 1000
 MOVE JOINT 2 BY 0.3 NOSIMUL
---------------------------------------- END MAIN ------------------------------------
END fireta25
::::::::::::::
```

FEEDER

```
:::::::::::::

PROGRAM token3
VAR
-------------------------------- Main Declaration Section----------------------------
ta40  : POSITION
ta37  : POSITION
ta36  : POSITION
ta35  : POSITION
ta34  : POSITION
ta33  : POSITION
ta32  : POSITION
ta3  : POSITION
ta4  : POSITION
ta5  : POSITION
-------------------------------------------------------------------------------------

BEGIN MAIN
-------------------------------------------------------------------------------------
 $speed = 1000
 MOVE TO ta4
 MOVE TO ta5
 MOVE TO ta3
 $speed = 500
 DOUT[ 2 ] = ON
 WAIT UNTIL DIN[ 0 ] == ON
 $speed = 1000
 MOVE TO ta32
 MOVE TO ta33
 MOVE TO ta34
 delay 5000
 $speed = 1800
 MOVE TO ta35
 MOVE TO ta36
 delay 2500
 MOVE TO ta37
 DOUT[ 10 ] = ON
 DOUT[ 16 ] = ON
 delay 2000
 MOVE TO ta40
 delay 3000
-------------------------------------- END MAIN ------------------------------------
END token3
:::::::::::::
```

PCB_AREA.
:::::::::::::

```
PROGRAM token4
VAR
-------------------------------- Main Declaration Section---------------------------
ta47  : POSITION
ta34  : POSITION
ta46  : POSITION
ta23  : POSITION
ta20  : POSITION
ta19  : POSITION
ta11  : POSITION
ta10  : POSITION
ta8  : POSITION
ta7  : POSITION
-----------------------------------------------------------------------------------

BEGIN MAIN
-----------------------------------------------------------------------------------
 WAIT UNTIL DIN[ 3 ] == ON
 $speed =1000
 MOVE TO ta7
 MOVE TO ta8
 DOUT[ 4 ] = ON
 delay 5000
 $speed = 500
 MOVE TO ta10
 DOUT[ 6 ] = ON
 WAIT UNTIL DIN[ 11 ] == ON
 MOVE TO ta11
 MOVE JOINT 2 BY -0.4 NOSIMUL
 delay 3000
 DOUT[ 7 ] = ON
 WAIT UNTIL DIN[ 16 ] == ON
 $speed = 1000
 MOVE TO ta19
 MOVE TO ta20
 MOVE TO ta23
 $speed = 500
 DOUT[ 0 ] = ON
 DOUT[ 1 ] = ON
 WAIT UNTIL DIN[ 12 ] == ON
 $speed = 1000
 MOVE TO ta46
 MOVE JOINT 2 BY 0.4 NOSIMUL
 delay 3000
 MOVE TO ta47
 MOVE TO ta34
 delay 3000
---------------------------------------- END MAIN ----------------------------------------
END token4
```