

12-31-1991

## A graphical tool for the simulation of timed petri nets

Javaid Aslam Siddiqi  
*New Jersey Institute of Technology*

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



Part of the [Electrical and Electronics Commons](#)

---

### Recommended Citation

Siddiqi, Javaid Aslam, "A graphical tool for the simulation of timed petri nets" (1991). *Theses*. 2622.  
<https://digitalcommons.njit.edu/theses/2622>

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](mailto: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.

# ABSTRACT

Thesis Title: **A Graphical Tool for the Simulation of Timed Petri Nets**

Name of Candidate: **Javaid Aslam Siddiqi**

Thesis directed by: **Dr. Anthony Robbi**  
**Associate Professor**  
**Department of Electrical and Computer Engineering**

A highly interactive graphical tool is developed for the drawing and simulation of discrete event systems involving time constraints. The Timed Petri Net Simulator tool (TPNS) is capable of simulating systems with deterministic and stochastic delays. Performance and utilization parameters are collected during simulation for behavior and analysis purposes. Simulation results are stored in different files (log, verify, utilize, and mark). Conflict resolution on the basis of priority or random selection of multiple transitions in conflict is implemented to handle complex modeling situations. A user friendly interface is developed for effective interaction between the tool and the user. This interface utilizes pop-up windows and an on-line help facility.

**A GRAPHICAL TOOL FOR THE  
SIMULATION OF TIMED PETRI NETS**

Thesis presented

*by*

Javaid Aslam Siddiqi

*to*

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements  
for the degree of

**Master of Science in Electrical Engineering**

New Jersey Institute of Technology  
Newark, New Jersey

December 1991



---

## APPROVAL SHEET

Thesis Title: **A Graphical Tool for the Simulation of Timed Petri Nets**

Name of Candidate: **Javaid Aslam Siddiqi**  
**Master of Science in Electrical Engineering**

Thesis and Abstract Approved by:

**Dr. Anthony Robbi** Date  
Associate Professor  
Electrical and Computer Engineering Department

Members of the  
Thesis Committee:

**Dr. MengChu Zhou** Date  
Assistant Professor  
Electrical and Computer Engineering Department

**Dr. Daniel Chao** Date  
Assistant Professor  
Computer and Information Science Department

---

---

## VITA

**Name:** Javaid Aslam Siddiqi

**Permanent Address:**

**Degree and date to  
be conferred:** M.S.E.E. December 1991

**Date of Birth:**

**Place of Birth:**

**Secondary Education:** Pakistan Embassy School, Jeddah, Saudi Arabia.

Institutions attended	Date	Degree	Graduation
New Jersey Institute of Technology	Sept. 1989- Dec. 1991	M.S.E.E.	Dec. 1991
N. E. D University of Engineering & Technology	July 1983- Feb. 1989	B.S.E.E.	Feb. 1989



# Contents

---

<b>LIST OF FIGURES.....</b>	<b>ix</b>
-----------------------------	-----------

<b>LIST OF TABLES.....</b>	<b>xi</b>
----------------------------	-----------

## **PETRI NETS : CONCEPTS, HISTORY AND APPLICATIONS.**

1.1	Basic Concept.....	1
1.2	Introduction to Petri Nets.....	3
1.2.1	Terminology and Representation.....	4
1.2.2	Transition Enabling and Firing.....	5
1.3	Applications.....	6
1.4	Modeling Systems.....	7
1.4.1	Finite State Machines.....	7
1.4.2	Flexible Manufacturing Systems.....	8
1.4.3	Data Flow Computation.....	10
1.5	Parallel Activities.....	11
1.5.1	Communication Protocols.....	12
1.6	Research Objectives.....	14
1.7	Overview of Chapters.....	15

## **TIMED PETRI NETS.....**

2.1	Time In Petri Nets.....	16
2.2	Status of PN Models Including Time.....	17
2.2.1	Ramachandani's Timed Petri Nets.....	17
2.2.2	Merlins's Time Petri Nets.....	18
2.3	Properties of Time Extensions.....	19
2.3.1	General Model.....	19
2.3.2	Firing Sequences.....	19
2.4	Timed Petri Nets and Firing Delays.....	20
2.5	Stochastic Petri Nets and Transition Delays.....	21
2.5.1	SPN and transition delays.....	21
2.5.2	Generalized SPN (GSPN).....	21
2.5.3	SPN with Deterministic and Exponential firing times (DSPN).....	22
2.6	Status of Current TPNS model.....	22

<b>TPNS GRAPHICAL USER INTERFACE.....</b>	<b>25</b>
3.1 Introduction to the User Interface.....	26
3.2 Pop-up Windows.....	26
3.2.1 Message Pop-ups.....	26
3.2.2 Data Entry Pop-ups.....	26
3.2.3 Alerts and their Application.....	26
3.2.4 Information Pop-ups.....	27
3.3 On-line Help Facility.....	27
3.4 Sunview Environment.....	28
 <b>STRUCTURE OF TIMED PETRI NET SIMULATOR TOOL.....</b>	 <b>29</b>
4.1 Modular Programming.....	29
4.2 Parts of the Tool.....	31
4.2.1 Graphical Editor.....	31
4.2.2 Petri Net Simulator (PNS).....	31
4.2.3 Timed Petri Net Simulator (TPNS).....	31
4.3 Extended Editor Features.....	32
4.3.1 Automatic Backup And Overwrite Protection.....	32
4.3.3 Action Choices.....	32
4.3.2 Built-in File Extensions.....	33
4.3.4 Error Pop-ups.....	33
4.4 Enhanced Simulator Features.....	34
4.4.1 Date Stamping.....	34
4.4.2 Conflict Detection.....	34
4.4.3 Conflict Resolution By Random Selection.....	35
4.5 Description of Modules.....	36
4.5.1 check_cycles.c.....	36
4.5.2 display_file.c.....	36
4.5.3 enable_timed.c.....	36
4.5.4 fire_immediate.c.....	38
4.5.5 fire_timed.c.....	38
4.5.6 net_utilization.c.....	38
4.5.7 priority_selection.c.....	39
4.5.8 random_selection.c.....	40
4.5.9 sort_enabled_trans.c.....	40
4.5.10 trans_modify.c.....	40
4.5.11 update_place.c.....	40
4.5.12 update_remaining_ticks.c.....	41
4.5.13 vacant,c.....	41
4.6 Data Structures.....	41
4.6.1 Places.....	42
4.6.2 Transitions.....	43
4.6.3 Arcs.....	44
4.6.4 Cycles.....	45

<b>DESCRIPTION AND WORKING OF TPNS TOOL .....</b>	<b>47</b>
5.1 General Description.....	47
5.2 Net Management.....	49
5.3 Panel Buttons.....	49
5.3.1 Load.....	49
5.3.2 Save.....	50
5.3.3 Clear.....	50
5.3.4 Redraw.....	50
5.3.5 DelSeg.....	51
5.3.6 DelArc.....	51
5.3.7 Verify.....	51
5.3.8 Step.....	52
5.3.9 Run.....	54
5.3.10 Help.....	55
5.3.11 Last Action.....	55
5.3.12 Display.....	55
5.3.13 Reset .....	57
5.3.14 Quit.....	57
5.4 Panel Object.....	58
5.4.1 Places.....	58
5.4.2 Transitions.....	58
5.4.3 Arcs.....	59
5.4.4 Tokens.....	60
5.4.5 Eraser.....	60
 <b>EXAMPLE APPLICATIONS</b>	
6.1 An Example of Timed Net.....	61
6.2 Application of TPNS to FMS.....	68
6.2.1 System Representation.....	68
6.2.2 System Operation.....	68
6.2.3 System Behavior.....	72
 <b>CONCLUSION.....</b>	<b>78</b>
7.1 Summary.....	78
7.2 Future Work.....	79
7.2.1 Text Processing Program.....	79
7.2.2 Extensions to the Stochastic Model.....	79
7.2.3 Extensions to Conflict Handling.....	79
7.2.4 Integration of TPNS And CPNS.....	79
7.2.5 Enhancements to On-line Help.....	79
 <b>REFERENCES.....</b>	<b>81</b>
 <b>Appendix-A List of Programs Files used in TPNS.....</b>	<b>84</b>
<b>Appendix-B List of Image Files used in TPNS.....</b>	<b>85</b>

# List of Figures

Figure #	Description	Page #
1.1	A simple Petri net .....	4
1.2	A Petri net (A state machine) representing the state diagram of a vending machine, where coin return transition are omitted.....	7
1.3	A Petri net structure called conflict, choice or decision. It is a structure exhibiting non determinism. ....	8
1.4	A block diagram for a Flexible Manufacturing System. ....	9
1.5	A Petri net model of FMS in Fig. 1.4. ....	10
1.6	A Petri net showing a data flow computation for $x = (a+b)/(a-b)$ .....	11
1.7	A Petri net representing deterministic parallel activities. ....	12
1.8	A simplified model of a communication protocol. ....	13
2.1	Equivalence of Timed Petri nets and Time Petri net. ....	19
2.2	TPNS $\supset$ TdPNs $\supset$ Classical Petri nets. ....	20
2.3	The preselection scheme. ....	24
3.1	A Tag pop-up window. ....	27
4.1	A Timed Petri net with ticking transitions. ....	36a
5.1	TPNS tool Panel Canvas window. ....	48
5.2	A sample net with 3 places and 2 transitions. ....	51
5.3	Petri Net saved with a temporary name. ....	52
5.4(a)	User specified termination condition reached. ....	54
5.4(b)	A Petri Net reached dead lock state. ....	54
5.5	Run mode options. ....	55
5.6	An on-line Help window. ....	56
5.7	Display menu. ....	57
5.8	A Tag pop-up window. ....	59
5.9	A data entry pop-up for average delay of a stochastic transition .....	59
6.1	Working example of a Timed Petri net. ....	62
6.2	Firing diagram. ....	63

6.3	Utilization file created at the end of simulation for the net of of Fig. 6.1 .....	64
6.4	A marking file that contains instantaneous state information at each step for Fig. 6.1 .....	65
6.5	The log file that contains the results of simulation of net in Fig. 6.1 .....	67
6.6	A block diagram of a Flexible Manufacturing System (FMS). .....	70
6.7	Petri net model of FMS of Fig. 6.6. ....	71
6.8	Utilization file created at the end of simulation for the net of Fig. 6.6. ....	73
6.9	A marking file that contains instantaneous state information for each step for Fig. 6.6. ....	74
6.10	The log file that contains the results of simulation of net in Fig. 6.6. ....	75
6.11	Graphical representation of maximizing utilization of M1 & M2 .....	77

# List of Tables

Table No.	Description	Page#
1.1	Some typical interpretations of Places and transitions.....	5
4.1	Output file extensions and their description.....	33
5.1	Connection matrix for net in Figure 5.2.....	52
6.1	Places and their representation in an FMS application of Figure 5.15 and 5.16.....	69
6.2	Transitions and their representation in an FMS application of Figure 5.15 and 5.16.....	69
6.3	Simulation results after running net of Fig. 6.1 for 2000 steps .....	76

---

# PETRI NETS: CONCEPTS, HISTORY AND APPLICATIONS

## Chapter

# 1

## 1.1 Basic Concepts

Petri nets (PN) [1, 2] were originally developed by Carl Adam Petri in 1962 to model asynchronous concurrent systems. 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 and simple methods for describing and analyzing systems that may exhibit asynchronous and concurrent activities. As a graphical tool Petri nets can be used as a visual-communication aid similar to flow charts, block diagrams, and networks. These nets can simulate the dynamic and concurrent activities of systems.

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 analyzing the model, useful information about the underlying system can be obtained. This information might reveal system properties such as deadlocks. A model can be interpreted and used to incorporate changes without running the tests on an actual system. One fundamental

---

---

idea is that the modeled systems are composed of separate, interacting components. 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. For example, in a computer system, peripheral devices, such as card readers, line printers, tape drives may all operate concurrently under the ultimate control of the computer. In a manufacturing system a number activities take place in parallel.

Conventional systems are unable to model concurrent systems successfully. On the other hand, Petri nets have demonstrated powerful approach towards the solution of such concurrency and parallelism problems. Petri nets have proved to be a very useful modeling technique for systems with some of the following characteristics :

1. Concurrency and parallelism.
2. Synchronous and asynchronous systems.
3. Distributed systems.
4. Nondeterministic and/or Stochastic behavior.
3. Conflicts for resources.
5. Event driven.

These systems are difficult to model accurately with either differential equations or queuing theory. Petri nets can provide accurate and useful models for the following reasons:

1. Petri nets capture the precedence relations and structural interactions concurrent and asynchronous events.
  2. They are logical models derived from the knowledge of system operation. Their graphical nature is a good visual aid in the modeling process.
  3. Deadlocks and conflicts can be modeled easily and concisely.
-



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

## 1.2 Introduction to Petri Nets

A Petri net is composed of four types of elements : a set of places  $P$ , a set of transitions  $T$ , an input function  $I$ , and an output function  $O$ . The input and output functions relate transitions and places. The input function is a mapping from a transition  $T_j$  to a collection of places  $I(T_j)$ , known as the input places of the transition. The output function  $O$  maps a transition  $T_j$  to a collection of places  $O(T_j)$  known as the output places of the transition. The structure of a Petri net is defined by its places, transitions, input function and output function.

A Petri net structure,  $C$ , is a quadruple,  $C = \{P, T, I, O\}$ , where  $P = \{p_1, p_2, p_3, \dots, p_n\}$  is a finite set of places,  $n \geq 0$ .  $T = \{t_1, t_2, t_3, \dots, t_m\}$  is a finite set of transitions,  $m \geq 0$ . The set of places and the set of transitions are disjoint.  $P \cap T = \emptyset$ .  $I : T \rightarrow \mathcal{P}(P)$  is the input function, a mapping from the transitions to bags of places.  $O : T \rightarrow \mathcal{P}(P)$  is the output function, a mapping from transitions to bags of places. A basic Petri net structure is described as follows :

$$C = \{P, T, I, O\}$$

$$P = \{p_1, p_2, p_3, p_4, p_5\}$$

$$T = \{t_1, t_2, t_3, t_4\}$$

$$I(t_1) = \{p_1, p_3, p_5\}$$

$$I(t_2) = \{p_2, p_3, p_5\}$$

$$I(t_3) = \{p_3\}$$

$$I(t_4) = \{p_4\}$$

$$O(t_1) = \{p_2, p_3, p_5\}$$

$$O(t_2) = \{p_5\}$$

$$O(t_3) = \{p_4\}$$

$$O(t_4) = \{p_2, p_3\}$$

If we look at the above Petri net description, it is difficult to form a concept of the actual structure of the net. A graphical representation of a Petri net graph is much more

---

useful for illustrating the concepts of a Petri net theory. A Petri net graph is a representation of Petri net structure as a bipartite multigraph. A Petri net of the above structure is shown in Fig 1.1

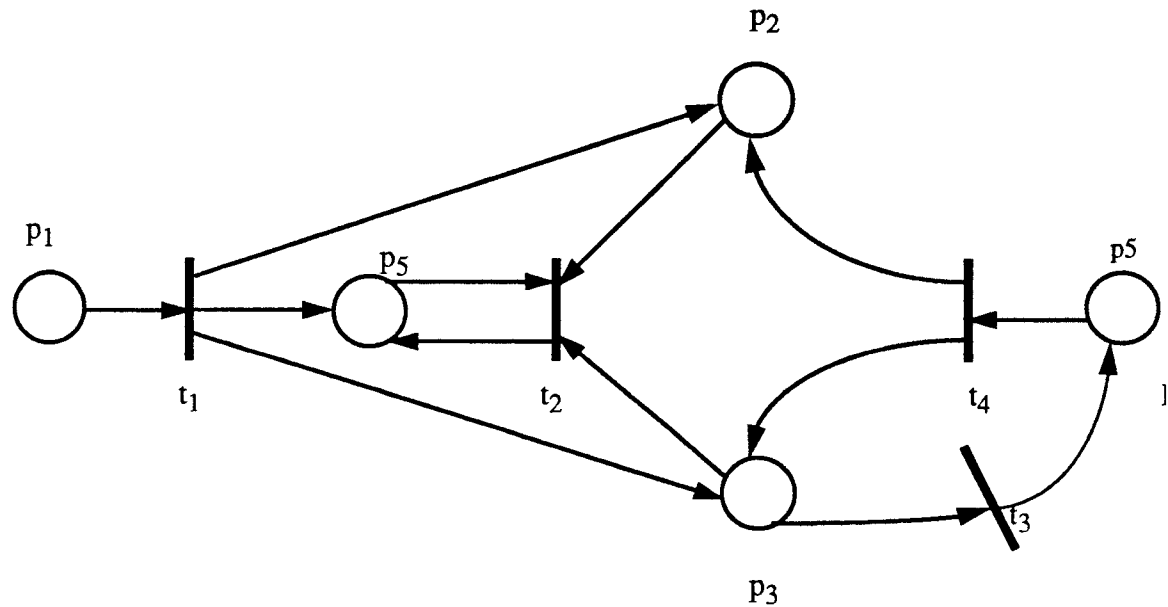







Figure 1.1 : A graphical equivalent of the above net.

### 1.2.1 Terminology and Representation

Petri nets consists of *places*, *transitions*, *tokens* and *arcs*. For graphical representation of Petri nets the following symbols are used :

- A place is represented by a circle 
- A transition is represented by a single solid bar  a hollow rectangular bar  and a thick solid bar  depending on the state and type of transition.

- 
- A token is represented by a solid small dot •
  - An arc is a directed arrow (  )
  - A *marking*  $m$  is an assignment of tokens to the places of a Petri net.

### 1.2.2 Transition Enabling and Firing

A Petri net is a particular kind of directed graph. It has an initial state called the initial marking  $M$ . A marking (state) assigns to each place a non-negative integer  $k$ , we say that  $P$  is marked with  $k$  tokens. Pictorially, we place  $k$  black dots(tokens) in place  $P$ . A marking is defined as the token distribution in each place is denoted by ' $m$ ', an  $m$ -vector, where  $m$  is the total number of places. The  $P$ th component of  $M$ , denoted by  $M(P)$ , is the number of tokens in place  $P$ .

In modeling, using the concept of conditions and events, places represent conditions, and transitions represent events. A transition(an event) has a certain number of input and output places. Their marking represent the pre-conditions and post-conditions of the event, respectively. The presence of a token in a place is interpreted as holding the truth of the conditions associated with the place. In another interpretation,  $k$  tokens are put in a place to indicate that  $k$  items or resources are available. Some typical interpretations of transitions and their input places and output places are shown in Table 1.1

**Table 1.1 :** Some Typical interpretations of Transitions and Places

Input Place	Transition	Output Place
Preconditions	Event	Postconditions
Input data	Computation step	Output data
Input signals	Signals processor	Output signals
Resources busy	Task complete	Resources released
Conditions	Clause in login	Conclusions
Parts	Assembly	Product
Buffers	Processor	Buffers

---

---

The behavior of discrete systems can be described in terms of system states and their changes. In order to simulate the dynamic behavior of a system, a state or marking in a Petri net is changed according to the following transition (firing) rules:

1. A transition is said to be enabled if each input place contains at least as many tokens as arcs from the place to the transition. For example,  $t_2$  in Figure 1.1, requires a token in each of  $p_2$ ,  $p_3$ , and  $p_5$ .
2. Firing of an enabled transition removes a token per arc from each input place and adds a token per arc to each output place. For example, if  $T_4$  in figure 1.1 fires a token is removed from  $P_4$  and a token is placed in each of  $p_2$  and  $p_3$ .

## 1.3 Applications

Petri nets have been proposed for a wide variety of applications. This is due to the generality and permissiveness inherent in Petri nets. They can be applied informally to any area or system that can be described with flow charts and that needs a means of representing parallel or concurrent activities. However, careful attention must be paid to a trade off between modeling generality and analysis capability. The more general the model, the less amenable it is to analysis. A weakness of Petri nets is the complexity problem, i.e., Petri net based models may become too large for analysis even for a modest size system.

Promising areas of applications include performance evaluation of communication systems, flexible manufacturing/industrial control systems, modeling and analysis of distributed software systems, distributed database systems, concurrent and parallel programs, discrete event systems, programmable logic and VLSI arrays, asynchronous circuits and structures, compiler and operating systems. Other areas of application are local-area networks, legal systems, human factors, neural networks, digital filters and decision models.

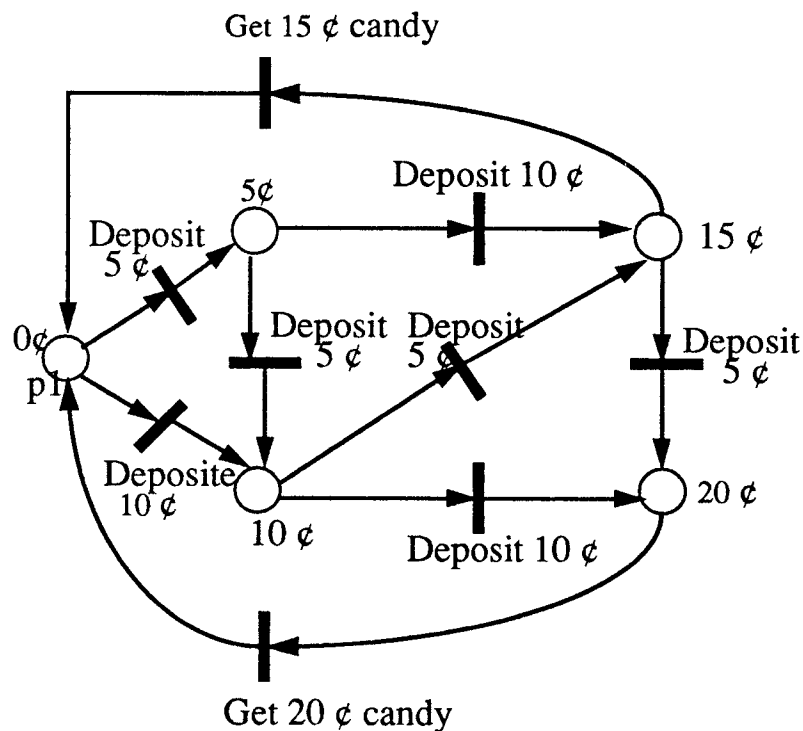
---

## 1.4 Modeling systems:

To illustrate the scope and practical importance of modeling using Petri nets the following examples are given for better understanding.

### 1.4.1 Finite State Machines

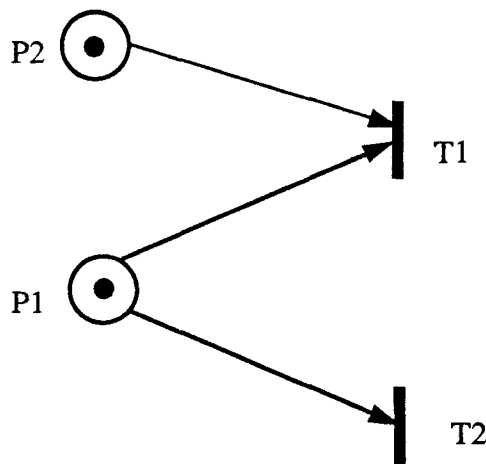
Finite-state machines or their state diagrams can be equivalently represented by a subclass of Petri nets. As an example of a finite-state machine, consider a vending machine which accepts either nickels or dimes and sells 15¢ or 20¢ candy bars. For simplicity, suppose the vending machine can hold up to 20¢. Then, the state diagram of the machine can be represented by the Petri net shown in figure 1.2, where the five states are represented by the five places labeled with 0¢, 5¢, 10¢, 15¢, 20¢, and transformations from one state to another state are shown by transitions labeled with input conditions, such as "deposit 5¢".



**Figure 1.2 :** A Petri net (a state machine) representating the state diagram of a vending

---

The initial state is indicated by initially putting a token in the place p1, with a 0¢ label in this example. Note that each transition in this net has exactly one incoming arc and exactly one outgoing arc. The subclass of Petri nets with this property is known as state machines. Any finite-state machine can be modeled with a Petri net state machine. Place P1 has two output transitions T1 (5 ¢) and T2 (10¢), as shown in Figure 1.3. This situation is referred to a conflict, decision, or choice because only one of T1 and T2 can fire. State machines allow the representation of decisions, but not the synchronization of parallel activities.



**Figure 1.3 :** A Petri net structure called a conflict choice or decision. It is a structure exhibiting non determinism

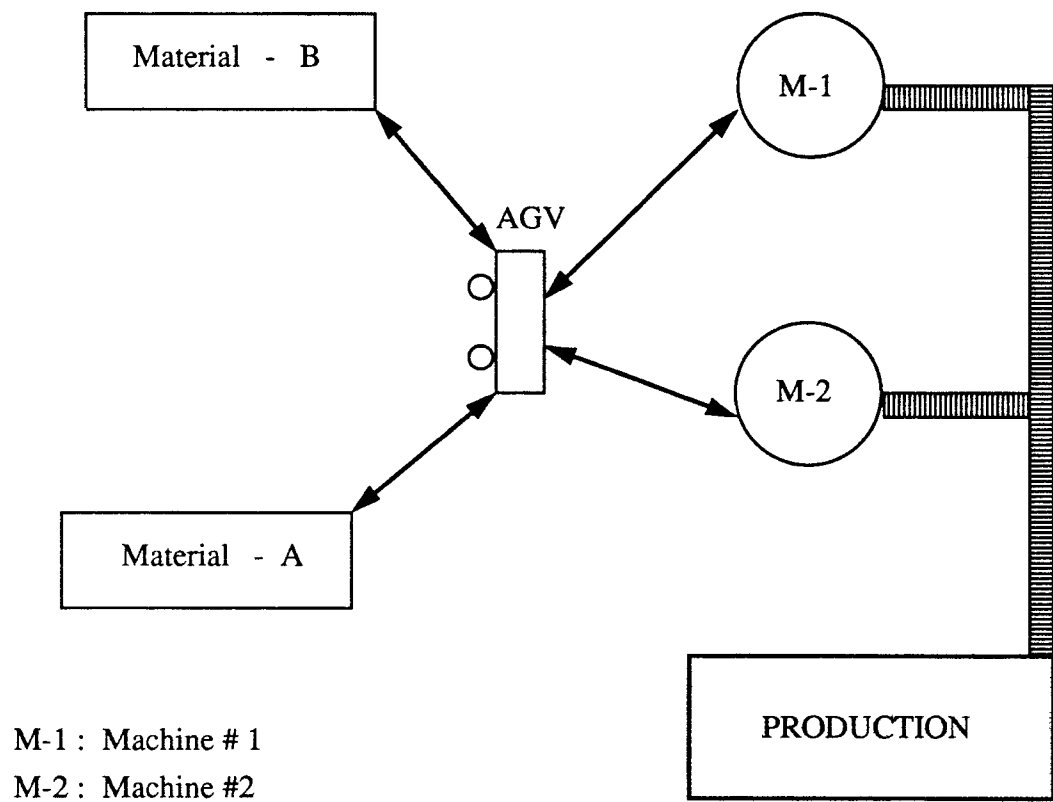
### 1.4.2 Flexible Manufacturing Systems

One of the promising areas of application has been Flexible Manufacturing Systems (FMS). The following section discusses a simple example of a Flexible Manufacturing System with focus to Petri nets.

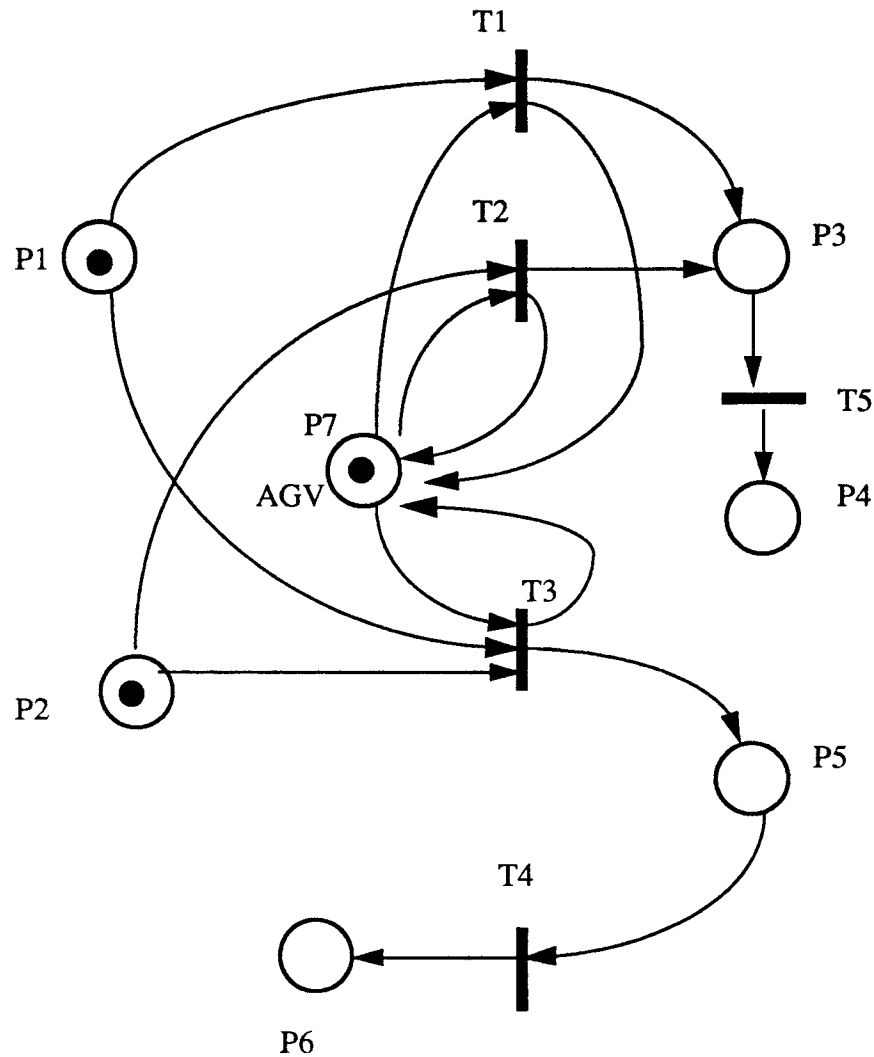
In this example two machines M1 and M2 operate on different combinations of parts arriving from the input buffer. Machine 1 processes individual parts from input buffer

---

1 or 2. The input buffers are labeled as P1 and P2. The AGV is required to transport these parts to the machines. If both buffers have parts available then only the AGV is capable of transporting the parts of machine 2 which is labeled as T4. The place P5 serves as a staging place for machine 2. Similarly place P3 serves as a staging for machine 1 which is represented by transition T5. On reaching either machines the AGV is freed and becomes available for the next transportation. This example shows how synchronism as well as flexibility is modeled using Petri nets. Figure 1.4 shows the modeled system, where as Figure 1.5 shows its Petri net representation.



**Figure 1.4:** A block diagram of a Flexible Manufacturing System



**Figure 1.5:** A Petri net model of FMS in Fig. 1.4

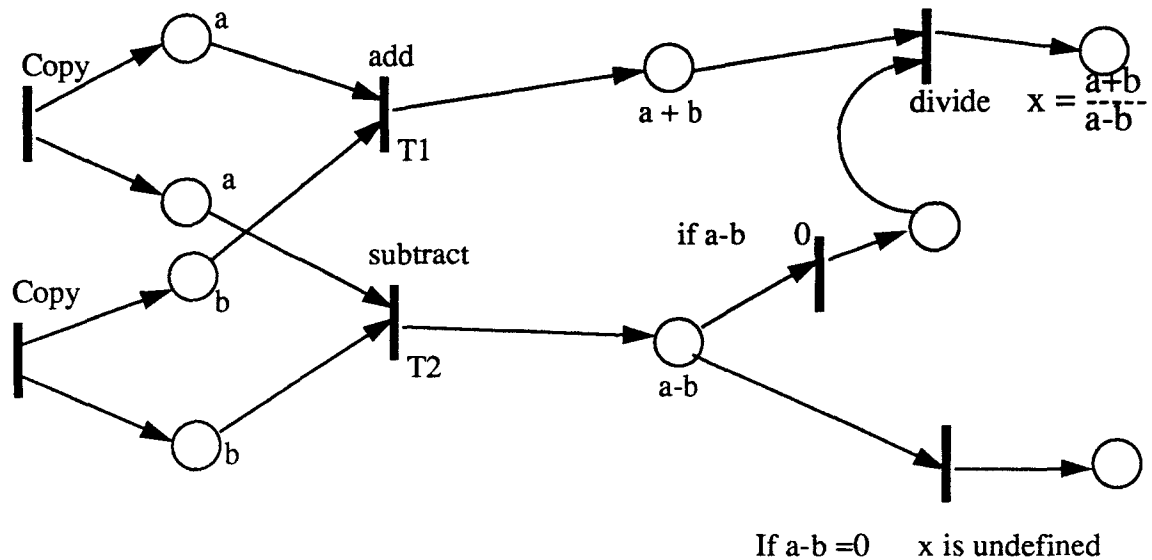
### 1.4.3 Dataflow Computation

Petri nets can be used to represent not only the flow of control but also flow of data. The net shown in Figure 1.6 is a Petri net representation of a dataflow computation. A dataflow computation is one in which instructions are enabled for execution by the arrival of their operands, and may be executed concurrently.

In the Petri net representation of a data flow computation, tokens denote the values of current data as well as the availability of data. In the net shown in Figure 1.6, the instruc-



tions represented by transitions T1 and T2 can be executed concurrently and deposit resulting date (a+b) or (a-b) in the respective output places.



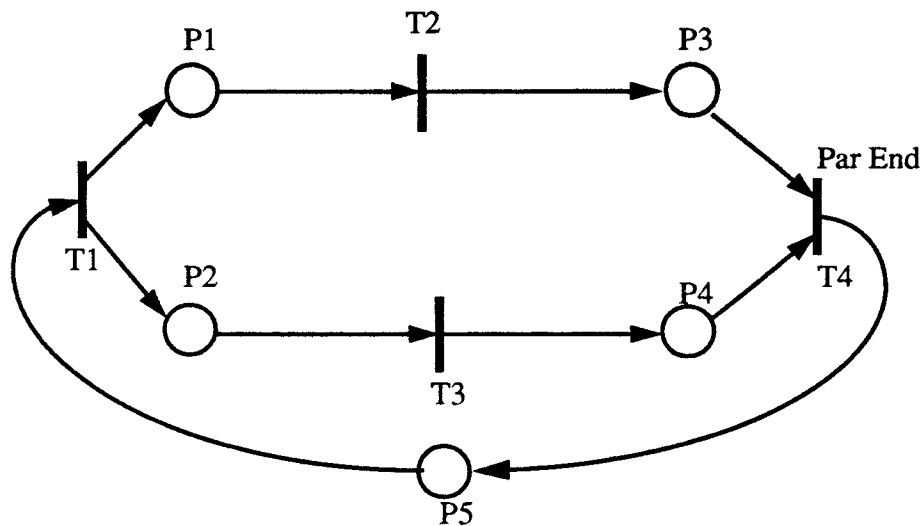
**Figure 1.6:** A Petri net showing a data flow computation for  $x = (a+b)/(a-b)$

## 1.5 Parallel Activities

Parallel activities or concurrency can be easily modeled by Petri nets. For example, in the Petri net shown in Figure 1.7, the parallel or concurrent activities represented by transitions T2 and T3 begin at the firing of transition T1 and end with firing of transition T4. In general two transitions are said to be concurrent if they are actually independent, i.e., one transition may fire before or after or in parallel with the other, as in the case of T2 and T3 in Figure 1.4.

It has been pointed out [Ref] that concurrency can be regarded as a binary relation (denoted by  $e_0$  on the set of events  $A = \{e_1, e_2, \dots\}$ ). For example one may drive a car (event  $e_1$ ) or walk (event  $e_3$ ) while singing (event  $e_2$ ), but one can not drive and walk concurrently. Note each place in the net shown in Figure 1.7 has exactly one incoming arc and

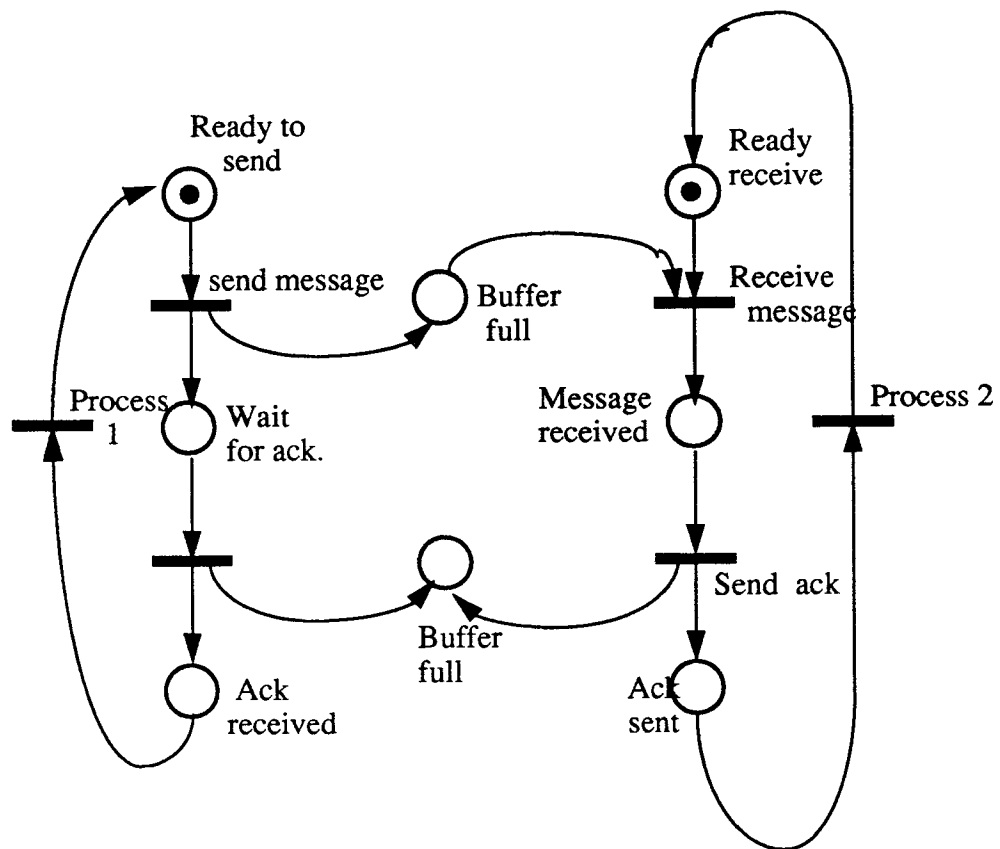
exactly one outgoing arc. The subclass of Petri nets with this property is known as marked graphs. Marked graphs allow representation of concurrency but not decisions(conflicts). Two events  $e_1$  and  $e_2$  are in conflict if either  $e_1$  or  $e_2$  can occur but not both, and they are in concurrent if both events can occur in any order without conflicts.



**Figure 1.7:** A Petri net (a marked graph) representing deterministic parallel activities

### 1.5.1 Communication Protocols

Communication protocols are another area where Petri nets can be used to represent and specify essential features of a system. The liveness (lack of deadlock) and safeness properties of a Petri net are often used as correctness criteria in communication protocols. The Petri net shown in Figure 1.8 is a very simple model of a communication protocol between two processes. Figure 1.8 shows the Petri net representation of a nondeterministic wait process where  $tr_1$ ,  $tr_2$ , or  $t_{out}$  fires if response 1, response 2, or no response is received before a specified time ( $t_{out}$ ) respectively.



**Figure 1.8 :** A simplified model of a communication protocol

---

## 1.6 Research objectives

System performance and behavior can be studied once a system is translated into a Petri net model. At present, a computer tool with the expertise to model a system with Petri nets does not exist. Once a system is translated, simulation and analysis of the net should be automated. Thus there is a need for a graphical computer tool which allows one to create a Petri net structure on screen and simulate the dynamic behavior of the drawn net according to the prescribed firing rules. Feldbrugge[17] has given an overview of available net-based tools. Most of these tools were developed on machines not in use these days and were used for some particular class of modeling problems. Later, Chio-  
la[18] and Vernon[19] presented more general graphical Petri net tools on SUN3 workstations running under UNIX<sup>1</sup> 4.2<sup>1</sup>. These tools however, are not available and modifiable. The field of modeling using Petri nets is constantly undergoing modifications resulting from research and development. As a result, it is extremely important that such tool be available in well-understood source form to enable modifications and development as the field progresses.

Initial work towards the development of a graphical Petri net tool was made by Arsalan Gilani [20]. Later Ashish Shukla [21] integrated a simulator for Petri nets with the graphical editor. This thesis is a continuation of the above work.

Time has been added to Shukla's model to enable simulation of real systems that have time delays as an important feature. New features have been added to enhance the editing and simulation capabilities of the tool. Systems with deterministic and discrete stochastic delays can be simulated and many performance and throughput parameters are derived. Extensive improvements in the user interface have made it possible to learn and use the tool much easily. The acronym TPNS for "Timed Petri Net Simulator" is used to refer to the tool described here. Integration of TPNS and CPNS<sup>2</sup> into a unified package is planned.

---

<sup>1</sup> UNIX is a trade mark of AT&T Bell labs

---

---

The justification for using the SunView<sup>TM</sup> environment<sup>3</sup>, the description of SunView capabilities which contributed towards the development of TPNS and the explanation of program specifications and salient features of the editor code are detailed in Gilani's [20] work. The basic simulator structure and description of modes of operation (i.e. Step and Run) are available as a part of Shukla's [21] work.

## 1.7 Overview of Chapters

In the following chapters concepts, previous research and current status of timed net is discussed as a part of the second chapter. This chapter also discusses the status of the current research in timed nets.

The third chapter describes the TPNS graphical interface and its features. It describes the user interface and its features. There are many suggested future developments in this regard. The user interface has been improved and new features are added to it.

The fourth chapter discusses structure of Timed Petri Net Simulator. It mainly focuses on the file system, type of modules and programs that are required to run this simulation tool. It has a section that describes the data structures used for objects in this tool. Chapter five is the user manual for this tool. It describes TPNS tool and its working in detail.

An example for Zubereks paper [8] is used in chapter six to verify the conformance of results. Application of TPNS tool to FMS is given in section 6.2. This section is completely devoted to the discussion of using the tool for performance analysis. The last chapter has the concluding remarks and discussion of future work.

---

<sup>2</sup> The work of Sanjay Desai towards the development of a simulator for Colored Petri Nets (CPNS) is carried out in parallel to this work of TPNS and is almost complete. In CPNS tokens and arcs carry information which can be used to model concurrent and interlocked systems such as flexible manufacturing systems.

<sup>3</sup> SunView is a trade mark of Sun micro systems

---

---

# TIMED PETRI NETS

## Chapter

# 2

## 2.1 Time in Petri Nets

In conventional Petri net theory no consideration is given to time [7, 25, 26], transitions fire instantaneously. Practical systems are constructed from devices which have a finite speed of operation. For example, in an FMS a finite amount of time is required for a robot to complete an activity. Almost every practical system requires a finite amount of time to complete its operation. Since conventional Petri nets do not have time parameters as part of their definition, they do not contain enough information for throughput evaluation. Time is introduced into the PNS model to have a more powerful and flexible method of evaluating system behavior, performance and the throughput of systems with timing delays.

In order to model time as part of a Petri net model, a new class of transitions is introduced which is called timed transitions [1, 3, 7, 8, 23]. A finite time delay is associated with these transitions to model timing behavior. Addition of time to Petri nets has opened a wide variety of application areas that can be modeled. Different classes of timed Petri nets and their application is discussed in the following sections.

---

## 2.2 Status of PN Models Including Time

The first two extensions of classical Petri nets involving time came around 1973 with the publication of two PhD theses. Ramachandani's thesis [1] proposed a model called Timed Petri nets (TdPNs) in which a duration or a firing time is associated to each of the transitions of a classical model. Merlin's thesis in the University of California Irvine [2, 3, 4] proposed another model called Time Petri nets (TPNs). A third extension was proposed three years later by Sifakis [5]. In this model a delay is associated to each place of a classical Petri Net. Later Sifakis, concluded that his model was equivalent to Ramachandani's model. The model proposed by Sifakis was named in this occasion Timed Place-Transition nets (TdPTNs)

The classical net obtained from a time(d) net by disregarding all timing information is called the associated classical net of a Time(d) Petri Net and will be denoted by a quadruple  $PN = (Ps, Ts, Is, Os)$  as discussed in section 1.2, where  $Ps$  is the set of places,  $Ts$  is the set of transition,  $Is$  is the transition input function such that  $I(p,t)$  is the weight of the oriented arc leading place  $P$  to transition  $T$ ;  $Os$  is the transition output function such that  $O(p,t)$  is the weight of the oriented arc leading transition  $T$  to place  $P$ . The set of places where  $I(p,t) > 0$  will be called the set of input places of  $T$  and the set of places where  $O(p,t) > 0$  will be called the set of output places of  $T$ .

### 2.2.1 Ramachandani's Timed Petri Nets

A Timed Petri Net is a couple  $(PN, \zeta)$  where  $PN$  is a classical Petri Net  $(Ps, Ts, Is, Os)$  and  $\zeta$  is the function assigning a real non negative number to each  $T_i \in Ts$  called the firing time of transition  $T_i$ . Transitions are enabled the same way as in classical Petri nets (Section 1.2.2). Firing an enabled transition  $T$  provokes a two step marking change.

1. The first step is instantaneous with the enabling of  $T$  and provokes a decrement in the marking of input places as follows
-

---


$$\forall p \in P_s, \quad m'(p) = m(p) - I(p,t)$$

Where  $m'(p)$  is the marking after the completion of first step  
 $m(p)$  is the marking before the step  
 $I(p,t)$  is the weight of the leading edge from place  $P$  to transition  $T$ .

2. The second step occurs  $\zeta(T_i)$  units of time latter and will provoke an increment on the marking:

$$\forall p \in P_s, \quad m''(p) = m'(p) + O(p,t)$$

where  $m''(p)$  is the marking of the place  $P$  obtained after the firing of  $T$   
 $O(p,t)$  is the weight of the edge leading transition  $T$  to place  $P$ .

As it can be seen, the marking after a transition firing in a Timed Petri net is the same as in its associated classical Petri net. When a transition is enabled a firing is initiated. The above firing rule shows two kinds of states: **Active** and **Inactive states**.

Active States: This state is the one where there are active transitions, i.e. transitions for which firing is incomplete [1, 10].

Inactive states : In this state no transition is active [1, 10].

### 2.2.2 Merlin's Time Petri Nets

A Time Petri net is a couple  $(PN, \theta)$ , where  $PN$  is a classical Petri net  $PN = \{P, T, I, O\}$  and  $\theta$  is a function assigning a closed interval  $[a_i, b_i]$  to each transition  $t_i \in T_s$ , called the static firing interval of transitions  $T_i$ . No restrictions to the lower and upper bound of the interval is implied, except that  $b_i$  must be greater or equal to  $a_i$ . This means that the lower bound of the static fire interval may be zero and the upper bound may be infinite.

Transitions are enabled the same way as in classical Petri nets, but firing only

---



occurs within the limits of time defined by the static firing interval beginning from the moment the transitions was enabled. The firing of a transition itself is an instantaneous action and produces exactly the same effect as in the classical model.

## 2.3 Properties of Time Extensions

As result of introduction of time to a classical Petri net model the following properties are observed:

### 2.3.1 General Model

The set of all TPNs contains both the set of all classical nets and the set of all TdPNs. In fact, classical Petri nets may be considered as special cases of TPNs where all static firing intervals are set to zero. Timed Petri nets may equally be considered as a special class of Time Petri nets. It is possible to obtain a TPN with the same behavior as a TdPN if every transition  $T_i$  in a TdPN with firing time  $x$  is substituted by the subnet consisting of two transitions and one place as shown in Figure 2.1

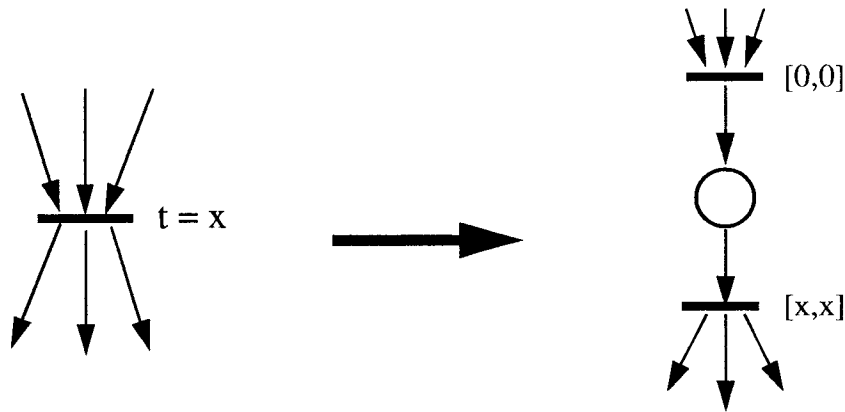


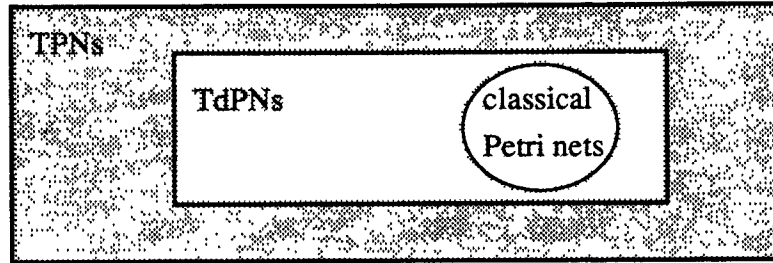
Figure 2.1: TdPNs  $\longrightarrow$  TPNs

### 2.3.2 Firing Sequences

In TdPNs transitions starts firing as soon as they become enabled. Classical Petri nets could also be considered as a special class of TdPNs in which all firing times

---

associated to the transitions are set to zero. This is because the set of all firing sequences in a TdPN with all firing times set to zero is exactly the same as for their associated classical net. So, if we define net equivalence solely on the basis of the firing sequence, it is possible to classify classical nets as special cases of TdPNs.



**Figure 2.2 :**  $TPNs \supset TdPNs \supset \text{Classical Petri nets}$

## 2.4 Timed Petri Nets

There are several variants on this model.

- A variable time can be associated with transitions. This model was introduced by P. Merlin [2] who used a fixed range of firing times for transitions to model protocols.
  - A fixed time can be associated with transitions so that tokens are removed and withheld for a fixed period of time before the tokens can appear in the output places [8, 23].
  - A fixed time can be associated with the transitions as a firing delay so that the actual removal of the tokens and appearance of the output tokens is instantaneous after the delay has expired [1].
-

---

## 2.5 Stochastic Petri Nets

The Stochastic Petri Net (SPN) considers the transition firing and the resulting marking process to be a stochastic process, usually a Markov process or semi-Markov process. There are many variants of this model.

### 2.5.1 SPN and Transition delays

- The common stochastic Petri nets associate an exponentially distributed random variable with each transition for the time to fire the transition [7], [12].
- Another model used more general distributions of time [14].
- The generalized stochastic Petri net (GSPN) adds a transition that takes zero time to fire, the immediate transition [15].
- Another variant allows the random selection of which tokens to remove and which tokens to add to the output places [16].

There are several variants depending on the rule for conflict handling. The selection of which transition to fire in a conflict may occur independently of the firing time (this is called preselection, see Section 2.6) or the firing of the transitions may be considered to continue in parallel so the transition selected is the transition with the minimum firing time.

### 2.5.2 Generalized SPN (GSPN)

GSPN, that were introduced in [14], try to cope with the problem associated with the explosion of the number of MC states by defining two types of transitions: timed

---

---

transitions (drawn as boxes) and immediate transitions (drawn as thin bars) [23] . An exponentially distributed firing time is associated with timed transitions, whereas immediate transitions are defined to fire in zero time with priority over timed transitions. It was shown that GSPN are still equivalent to Markovian models and that their solution can be obtained with less effort than it is necessary for SPN.

### 2.5.3 SPN with Deterministic and Exponential Firing Times (DSPN)

DSPNs is an extension of GSPN model, which allows firing delays of timed transitions to be either constant, or exponentially distributed random variables [23].

## 2.6 Status of Current TPNS model

In the current TPNS model each transition takes a positive amount of time to fire. When a transition is enabled a firing can be initiated by removing a token from each of the T's input places. This token remains in the transition [7, 12, 23] for the "firing time" and the transition is represented on the screen as a thick solid block. This representation provides visual effects during net execution. Firing terminates by adding a token to each of the t's output places. Each of the firings is scheduled at the moment it is enabled [8], [23]. This is expressed in greater detail later in this section.

Operation of a timed net can thus be considered as taking place in "real time" and it is assumed that it starts at time ( $\tau = 0$ ). At this moment the firings of the timed transitions enabled by the marking  $m_0$  are initiated and the tokens are removed from their input places. Then, after the time determined by the smallest firing time of the enabled transitions, the tokens are deposited in the appropriate output places creating a new marking and a new set of enabled transitions. Timed transitions can have deterministic or stochastic delays [23] depending on what is modeled.

---

---

## Preselection

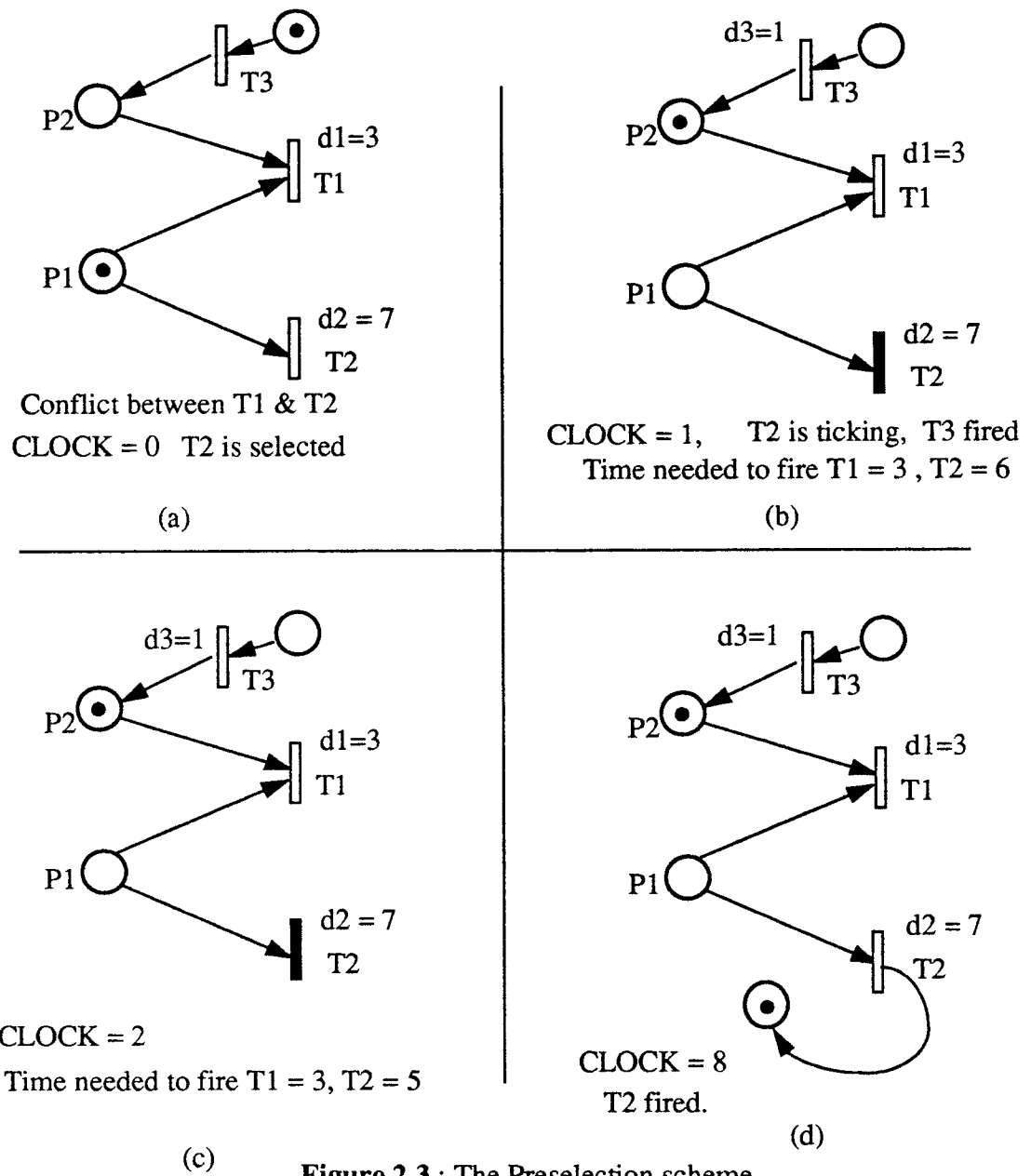
The method of preselection is used for transition firing [8, 23]. Consider the example in Figure 2.3. Transition T2 has a constant delay of 7 time units where as T1 has a lesser delay of 3 time units. With an initial marking  $m_0 = (1, 0)$  and  $CLOCK = 0$ , T1 and T2 will be in conflict. If T2 is selected as a result of conflict resolution, then the token in P1 will be absorbed by T2. This token will be held for 7 time units. If in the mean while (say at  $CLOCK = 1$ ) a token arrives in P2 from P3, T1 requires a token in P1 and P2 to get enabled. If the token held by T2 is available for T1, then T1 can fire earlier (at  $CLOCK = 4$ ). Where as if T2 retains that token then it will fire at  $CLOCK = 7$ . For this situation some researchers have allowed the token absorbed by T2 to be available for T1. Others allow the preselection scheme in which that token will be held by T2 until it fires. This pre-selection scheme has been used in TPNS tool.

To justify the use of this technique the following aspects are presented :

1. The technique will permit a fair policy of firing. If T2 is disabled due to the appearance of a token in P2, then T2 possibly may never fire.
2. Assuming these transitions to represent a process (drilling process) and a token absorbed by it as a piece of metal to be drilled, then reassignment of the token to T1 will require recording state of the token at the time of reassignment with respect to T2. This will involve maintaining lot of history information relevant to all the involved objects. The preselection method eliminates this requirement.

This technique is explained here to give a clear idea of the TPNS tool behavior in such situations. Users of this tool are subject to this rule and are required to create models with due consideration to this scheme.

---



**Figure 2.3 : The Preselection scheme**

---

# TPNS GRAPHICAL USER INTERFACE

**Chapter**  
**3**

## 3.1 Introduction to the User Interface

Most computer aided design and software application developers keep revising their user interfaces. Work is constantly done towards the improvement of the user interfaces for application software based on users feedback. A user interface may simply be defined as the interface between the human behavior and the capabilities and features of an application. For example if a user clicks a mouse button on an object, selects an option from a pull-down menu or requests loading, saving or printing action he expects a feedback about the completion, success or results of that action. A user may requests for help while he is in the middle of some activity and expects for helpful directions related to his situation. This is called case sensitive help. These features along with many others, constitute a user interface for an application software. The measure of success of a user interface is user satisfaction in terms of comfort of use and ability to learn and operate quickly. Therefore, an application can be very powerful, yet lack of a suitable user interface can result in considerable failure for that software. Developing a user interface for an application requires the addition of many features. For the PNS, the interface was in its preliminary stages. As a part of developing TPNS considerable attention is given

---

towards the improvement of this aspect.

## 3.2 Pop-up Windows

To draw immediate attention a pop-up window with a message is a very effective technique. Normally these windows suddenly appear in the middle of the screen.

### 3.2.1 Message Pop-ups

These pop-up windows can be examples to inform the user of an action and its status (i.e. failure or success). An interpretation of a result, an error message and a help message are examples. Such pop-ups are extensively used for TPNS tool. When an invalid button on the Panel is clicked, an error message appears in a pop-up window with a brief description of the invalidity and some hints (where required) on how to correct it.

### 3.2.2 Data Entry Pop-ups

These windows can be used to request the user to input some data. In this way the user is prompted for input in a very prominent way. The alternative is to prompt the user for input in the parent window, which may be hidden behind the tool window. A disadvantage of using the parent window for user input is that, in most cases, a new user may not even realize that data entry is required. Using a pop-up window for this purpose eliminates the possibility of getting stuck.

### 3.2.3 Alerts and their Application

Alerts are, pop-up windows that require mandatory user attention. When an Alert is called it displays a pop-up window with a message and one or more buttons. The only action a user can take is to click the **Continue** button. Another type of Alert has two buttons, one to **Confirm** and one to **Cancel** an action. The third type has multiple options and buttons.

---



### 3.2.4 Information Pop-ups

A good example of this kind of pop-ups is the Tag pop-up. This pop-up window contains information either about a transition or a place. The following information is available for a place.

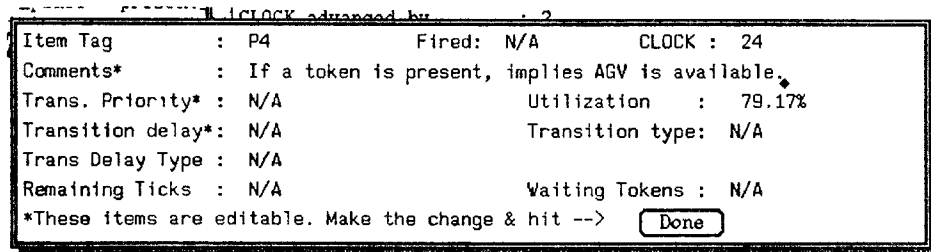


Figure 3.1: A Tag pop-up window with place information

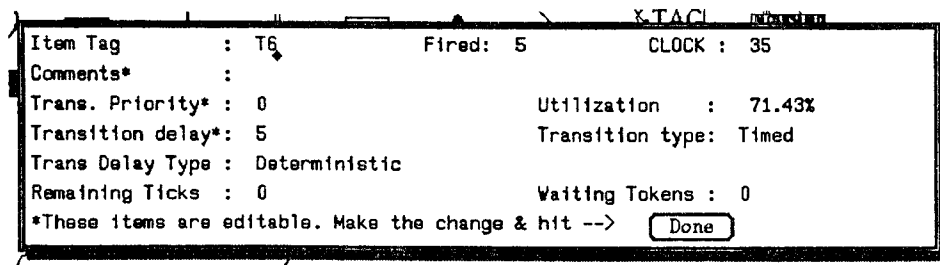


Figure 3.2: A Tag pop-up window with transition information

## 3.3 On-line Help Facility

An on-line help facility is designed for the TPNS tool to provide help on different topics and features of the tool. The help facility consists of a Panel window with a tty subwindow attached to it. The Panel consists of buttons labeled by the help topic. A sub-directory Help is created in the current P\_Nets directory. The file structure is as follows:

Source file name	tpns_help.c
Executable file	tpns_help*
All help topics	help_topic.txt

- 
- |                      |                          |
|----------------------|--------------------------|
| 1. How to Load/Save  | 2. Re-setting parameters |
| 3. Using the Eraser  | 4. Displaying files      |
| 5. Using tokens      | 6. Net utilization       |
| 7. Deleting Segments | 8. Net marking           |
| 9. Deleting Arcs     | 10. Getting started      |
| 11. Using Redraw     | 12. Using Verify         |

All the above topics provide concepts and working steps and hints where required. The help facility is designed such that it reads the text file for a particular topic and displays it on a tty subwindow, which will be discussed later in Section 5.3.10.

There has been a recent upgrade in SunView that can be a very convenient source of help for the future version of this tool. A new Panel attribute has been added to the existing list of attributes. This attribute (HELP\_DATA) is specified with every Panel object and all structures such as (canvases, tty subwindows, etc). A tag is attached to this attribute which is used to lookup for the help text related to that object or structure. This help text is stored in a specially formatted file that has these tags as markers to differentiate between help text on different topics [24].

### 3.4 SunView Environment

SunView (SUN Visual Interface Environment for Windows) has been selected for TPNS. SunView is a system to support interactive graphic based applications running within windows. The runtime system is based on a central Notifier which distributes input to the appropriate window, and a window manager to manage overlapping windows. The exchange of data between applications running in separate windows (in the same or separate processes ) is facilitated by a Selection Service. Sunview is an object oriented system. Think of sunview objects as visual building blocks which you use to assemble the users interface to your application. Different types of objects are provided, each with its particular properties; you employ whatever object you need for the task at hand. Technically, an object is a software entity presenting a functional interface [24].

---

---

# STRUCTURE OF TIMED PETRI NET SIMULATOR TOOL

**Chapter**  
**4**

## 4.1 Modular Programming

Programming is the art of converting a sequence of mathematical steps or a modeled system into computer instructions in a particular format that is executable by a computer. With time and experience programmers have learned techniques to develop programs more efficiently. Modular programming is one of these techniques. Programs written using this method are easily understood by others as well as being easy to debug and modify.

In modular programming control resides in the main module. Smaller functions inside the main program or separate programs that are included or linked with the main module perform a special or specific task. These functions or modules are usually written in such a way that they can be called from different parts of the main program or from inside other functions or programs. Therefore, generality of these functions is important. Parameters are passed to these functions or modules and results are passed back (returned) to the calling function or module. For example a function to sort an array or a linked list of structures can be called repeatedly from different locations without

---

writing code for each individual situation. Writing different modules(programs) for a particular task and including or linking it to the main module reduces the size of the main module and thus reduces the compilation time which results in ease of debugging and accelerates modification process.

The current work is a third phase in the development of a highly interactive and user friendly graphical interface for the simulation and performance analysis of Timed Petri Nets. At the end of the second phase [21] the tool was in the form of a single large file. A decision about the future file structure was required. There were the following two choices :

### Choice 1

The programming language used for this tool is C. It was suggested to break down the main module into large number of separate smaller modules based on their functions. Then by making use of the **make** utility these files could be linked. **Make** creates object modules of all files and every time it is executed, it recompiles only the modified modules. This technique would result in saving considerable compilation time. The use of this technique was postponed because of the large file size and the amount of time required for debugging and bringing the tool back to working order. The prime objective at the beginning of the third phase was to introduce time to the tool, making it possible to model, simulate and analyze real time systems and applications.

### Choice 2

As an alternate to the above method it was suggested that all possible new modifications to the current tool along with the work done in the third phase, will be added as separate program files. These files are then included in the main module as :

```
#include "filename.c"
```

This procedure does not reduce the compilation time, but eases and accelerates the debugging process.

---

---

## 4.2 Parts of the Tool

The overall structure of the tool can be divided into three basic functioning parts:

### 4.2.1 Graphical Editor

The Graphical Editor was developed as a part of the first phase [20]. This part provides drawing, editing, saving, loading and other basic editor functions. The tool runs under SunOs in SunView environment. SunView library functions are called to perform different graphical and windowing functions [20, 24].

### 4.2.2 Petri Net Simulator

The Petri Net Simulator was added to the existing tool as a part of the second phase[20]. The PN Simulator runs in two modes. The first mode is the "Step" mode, in which the net is executed for one step. A step is defined as the transition from one state to the next state by firing all enabled transitions simultaneously. The second modes is the "Run" mode in which the user can precisely control the extent of execution by specifying the number of steps to run or defining a certain termination state. A history file which describes the total population of tokens, transitions fired and net marking at every step is also created. The log file can be used later for performance and / or throughput analysis.

### 4.2.3 Timed Petri Net Simulator

The TPN Simulator is developed as a part of the third phase (this work). Now the tool is capable of modeling real systems with time as an important characteristic. Timed Petri Nets carry timing information [1, 8, 9, 12] as a part of the transitions. A deterministic or a stochastic delay is associated with each timed transition. Two new objects representing a time transition, with both horizontal and vertical orientations are added to the objects in the Panel window (Figure 5.1). The two modes of simulation, "STEP" and "Run" perform in the same manner with due consideration to timed transitions. The definition of a 'STEP' for a net with timed transitions has changed considerably and will be discussed in Section 5.3.8. In 'Run' mode the "STEP" mode is repeatedly called until the required state is reached or a deadlock occurs. During the execution of a net in

---

---

either mode a CLOCK called (the System Clock) ticks to determine the overall execution time and all relevant timing information for performance and throughput analysis.

## 4.3 Extended Editor Features

As a part of this work many features were added to enhance editing capabilities of the Graphical Editor. These features are discussed below.

### 4.3.1 Automatic Backups and Overwrite Protection

Every user is prone to mistakes. Sometimes these mistakes can be a disaster. In saving a new file or a modified version of an old file with the same old name, a user can, by mistake, overwrite very important informations. Therefore, the feature of automatic backup is added to the existing editor. When a user requests saving a file with an existing file name, he is warned about the existence of the old file with the same file name and is given an option to cancel his action. If the user still chooses to continue deliberately or not a backup of the old file is created with an extension “bak” (the user is prompted of the backup action). In this way, a disaster situation can be avoided.

### 4.3.2 Action Choices

In most cases where the user is given set of preset choices he is prompted about it using ALERTS (Section 4.3.4). These action choices have a uniformity of display. The actions are numbered (1, 2, 3, etc) and a short description of the action is followed by every number. At the bottom of the action window pop-up, buttons are labeled as . **choice #x**

By making these action choices uniform, the user gets familiar with the format of these pop-ups and chances of clicking the wrong choice button are reduced. The addition of this feature saves the user from swithcing between parent and tool window for choice selection requiring keyboard strokes. Important goals are to enable the user to rely on mouse clicks for most actions and reduce keyboard usage.

---

---

### 4.3.3 Built-in File Extension

Saving or loading a file requires the supply of a filename without any extension. To identify files that are created as a result of a simulation session the following extension are predefined and are used as standards as shown in Table 4.1..

**Table 4.1:** Output file extensions and their description

Extension	Abbreviates	Description
*.pic	Picture	This extension is automatically added for any Petri net picture file
*.log	Log	This extension is added for the history file that is created in "Run" mode. This file contains all the necessary information about net activity in each STEP.
*.vfy	Verify	This file is created in "Run" mode only and contains the initial connection matrix for places and transitions. The net connection can be obtained independently at any time by pressing the <b>Verify</b> button.
*.bak	Backup	This file is the backup version of the edited file.
*.utilize	Utilization	This file is created at the end of "Run" mode only and is used for transition and place utilization and behavior. The information can be used for performance and throughput analysis.
*.mark	Marking	This file contains the instantaneous marking of a net.

### 4.3.4 Error Pop-ups

If a user presses a wrong button or does an illegal action then, irrespective of any damage, he should be prompted of the error. The best way to do this is by popping up a small window with an error message. To ensure user's attention, a button press is required to continue. Such error messages catch user's attention and reduces the possibility of similar future errors. These pop-up windows are called **ALERTS**. When this window is displayed on the screen, the whole screen is frozen and TPNS waits for the user to press Continue.

---

---

## 4.4 Enhanced Simulator Features

The Petri Net Simulator tool is able to detect and resolve conflicts up to a certain level. In order to expand the scope of this tool the following features are added to the existing model.

### 4.4.1 Date Stamping

At any point during simulation users may want to stop the simulation process with the specified token distribution (Marking) and/or transition delays (if time transitions are present). They may wish to edit these parameters without doing any modification to the structure of the net (i.e. deleting, modifying or adding any place, transition or arc) and run simulations a number of times with modified parameters for performance analysis, and keep all results in the same log file.

In order to be able to distinguish results from different runs they must be separated. We can use the exact time of the clock and date information to be stamped to the log file before the start of each run. In the above mentioned case the users will request to append the results of every run to the existing log file (Figure 5.5). For initial marking information a verify file (\*.vfy) is also stamped with time and date informations. In this way it becomes easy for users to analyze a log file containing results of many runs at different times with an exact track of initial marking at every time. In order to avoid unnecessary log files to occupy large space on the disk, each time the users finish working with one net and requests loading or simulating another net they are asked to delete or keep the log file (if one exists for the recent Petri net file).

### 4.4.2 Conflict Detection

The PNS model was designed on the assumption that there will be only two conflicting transitions per conflict resolution. Detection of conflicts involving more than two transitions and situations with more than one conflicts involving a number of transitions in conflict was beyond the scope of their thesis research [20]. To be more general and allow more complicated multiple conflicting situations the tool has been

---



---

upgraded to handle a variety of complex situations. Now the tool is capable of detecting a number of simultaneous conflicts where every conflict may have m number of transitions in conflict. This enhancement will increase the flexibility of the tool to model concurrent systems with very complex multiple conflicting situations.

#### 4.4.3 Conflict Resolution By Random Selection

If a situation is detected with more than two transition in conflict requiring random selection for conflict resolution then due attention should be given to the fact that the random selection of one out of n transitions follows the equal opportunity rule. In case of two transitions of the same priority in conflict each transition has a probability of selection of 0.5. Similarly if the number of transitions is 5, then each should have the probability of selection of 0.2.

The PN Simulator was designed with consideration to allow two transitions, at the most, to undergo conflict [20]. Therefore, in TPN Simulator this restriction was removed and equal opportunity rule is applied for random selection of any number of transitions.

### 4.5 Description of Modules

The modules that are added to the main program are discussed in the following section. These modules are listed alphabetically as follows :

check_cycles.c	sort_enabled_trans.c
display_file.c	trans_modify.c
enable_timed.c	update_remaining_ticks.c
fire_immediate.c	update_place.c
fire_timed.c	random_selection.c
net_utilization.c	
priority_selection.c	

These modules are included in the main program as discussed in Section 4.1.

---

---

### 4.5.1 check\_cycles.c

The following module is called at the end of each STEP in “Run” mode simulation to check the initial marking. If that marking is found, the structure for cycle is updated. The fields in the cycle structure are self explanatory. If the “Run” mode is requested by the user for a net then the present marking for the net is recorded. This marking is checked for occurrence at the end of every step. If a match occurs, the time of the day from the system clock is assigned to that particular cycle along with the cycle number (which is nothing but the cycle occurrence sequence). This feature can be extended to detect any repeated marking in the net to detect cycles of a user- supplied marking. A similar feature is also available at the time of selecting a “Run” stop criterion. The user has an option of selecting a particular marking as a terminating point for a "Run" mode. This TPNS feature can be handy to detect cycles while running with a different terminating conditions.

### 4.5.2 display\_file.c

This module is called as a result of clicking a button on the Panel window. The user is asked to select from four options, namely,

1. To display the current log file.
2. To display the current connection matrix (known as the verify file).
3. To display the current utilization file.
4. To display the current marking file.

Section 5.3.12 discusses the Display button in more detail.

### 4.5.3 enable\_timed.c

This module is called to enable timed transitions. This module performs the following functions.

1. If the input places have sufficient tokens to permit firing, tokens are removed from the input places.
  2. It modifies all the enabled time transitions and their pictorial representations (Figure 4.1).
-

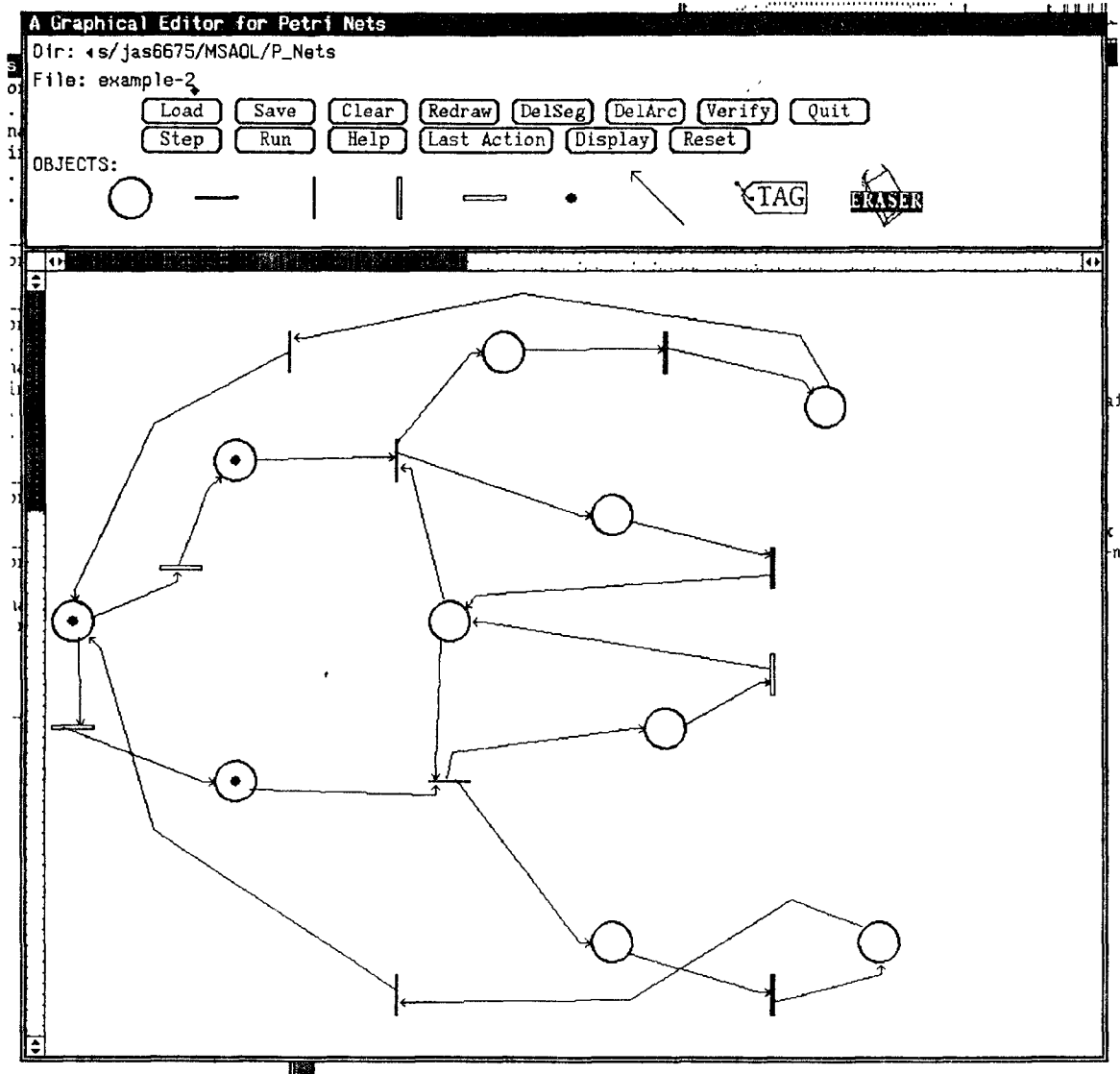


Figure 4.1: A Timed Petri net with ticking transitions

- 
3. In case of a stochastic transition, it calls the function `Get_Exponential_delay` to calculate the discrete stochastic delay. Every time a stochastic transition is enabled, a discrete stochastic delay is calculated. A random number is generated which is used to calculate the random variable (delay). The function uses the regenerative approach for producing a random number which is later used for generating a random variable. The regenerative random number method is used for simulation consistency. In simulating a net a user will be interested to run the simulation with the same set of values for random variables. His main focus will be on the net behavior with different average delays (stochastic transitions), different transition delays (deterministic transitions) and perhaps a different set of markings.

$$F_X(x) = \Pr[X \leq x] = 1 - e^{-\lambda x}$$

or the probability density function

$$f_X(x) = \lambda_i e^{-\lambda_i x}$$

The average delay is given by

$$\bar{d}_i = \int_0^{\infty} [1 - F_X(x)] dx = \int_0^{\infty} (e^{-\lambda_i x}) dx = \frac{1}{\lambda_i}$$

Where  $\lambda_i$  is the firing rate of transition  $T_i$

---

---

#### 4.5.4 fire\_immediate.c

This module is called to fire all immediate transitions. It removes tokens from all input places of enabled immediate transitions and places tokens in all output places updating the associated structure. The utilization parameters are also updated in this module.

#### 4.5.5 fire\_timed.c

This module is called with two parameters, the structure of the selected transition and the amount of delay by which the system clock has to be advanced. In order to advance the system clock, first the enabled timed transition with the minimum amount of delay is selected [8]. When the selected transition fires, the system clock is advanced by the transition delay. The sequence of execution and the steps will be similar to firing an immediate transition above. This module also modifies the pictorial representation of a fired timed transition from a black solid bar to a hollow bar to represent the release of tokens by that transition.

#### 4.5.6 net\_utilization.c

This module is called at the end of the “Run” mode. It creates a file showing the net utilization and behavior in terms of transition and place utilization. This file is created in the current directory with the notation filename.utilize. Filename is the name for the net and utilize is the standard extension added to the file.

##### (a) Transition Utilization

All transitions are tabulated against the following parameters

The number of times fired:	Number of times a transition fired.
Busy:	$\% \text{ Busy} = (\text{Total wait time of transition} * 100) / \text{run time}.$
Involved in conflict :	The number of times the transition was involved in conflict.
Success:	The number of times it won the conflict and fired.

Transition utilization can be checked in STEP mode using Tag pop-up.

---

---

### (b) Place Utilization and Behavior

All places are tabulated against the following parameters

Total number of tokens entered this place: Total tokens entered the place.

% Duration for which place was Occupied: Duration for which a place was vaccant.

A place is said to be vacant if there is no token in the place and all the timed transitions for which this place is an input place , are not enabled.

$$\text{vacant}(p_i) = \begin{cases} 1, & \text{if } m(p_i) = 0 \text{ and } T \text{ has absorbed a token } \forall t \notin P_i \\ 0, & \text{otherwise.} \end{cases}$$

where  $P_i = \{ t \mid I(p,t) \neq 0, t \in T \}$

The module `vaccant.c` is called to check if a place is vaccant at some point during simulation. This module calls the token functions to check the above described condition. If both calls to token return 0, the place is vaccant otherwise occupied. Place utilization can be checked in STEP mode using Tag pop-up.

### (c) Cycles detected

In "Run" mode the net checks by default the number of times the initial marking of the net is repeated and records the system time at that cycle (Section 4.5.1). If the initial marking is not repeated it reports that also.

### 4.5.7 `priority_selection.c`

This module checks the list of ready (enabled) transitions and deletes all those that are in conflict except the one with the highest priority (e.g. Priority 0 is higher than Priority 1 or 2). It then updates the linked list of ready transitions. Handling of the updated linked list becomes easier and simpler for the next modules and functions.

---

---

#### 4.5.8 random\_selection.c

This routine is called to resolve conflicts between two or more transitions that have equal priority. In this situation one transition has to be selected out of those in conflict. The PNS tool [21] was capable of handling two transitions in conflict. Here one of the conflicting transitions is retained in the list of ready transitions and all the losers are deleted. As a result of this update the linked list of ready transitions contains only those transitions that are enabled and non-conflicting.

#### 4.5.9 sort\_enabled\_trans.c

This module performs sorting of the linked list of ready transitions (`this_rdy_trans` & `first_rdy_trans`) which at this stage contains only timed transitions. The linked list is sorted in an ascending order on the transition delay. For stochastic transitions, the discrete transition delay is already calculated (Section 4.5.3) during the execution of `enable_timed.c`. The selected transition is the one with the minimum delay. In case of a more than one enabled time transitions with the same minimum firing delay then all of them are retained in the list and are fired at the same time in `fire_timed.c`.

#### 4.5.10 trans\_modify.c

This module is called once from `enable_timed.c` to convert the pictorially the hollow timed transition bar (representing waiting state) to a solid black bar representing a transition that has absorbed a token and is ticking. When the timed transition fires this module is called to convert the solid black bar into a hollow bar representing release of absorbed tokens.

#### 4.5.11 update\_place.c

This routine is called at the end of each step to modify each place's utilization parameters. Place occupancy is calculated on the basis of how long it was vacant. Another module "vacant.c" is called to find out if a place is vacant or not. This routine calculates the total accumulated time for which the place has remained vacant.

---

---

### 4.5.12 update\_remaining\_ticks.c

This module is called to update the timed transitions that are ticking. At every step the CLOCK is moved up by the minimum delay of all the enabled transitions. Remember that the conflicts are already resolved. The timed transition with the minimum delay is selected. At the end of each step the active clock for every ticking transition is updated by subtracting the delay of the selected transition from it. The active CLOCK for every transition is a field in the structure of transition.

### 4.5.13 vacant.c

This routine is called from the update\_place.c to find out if a place is vacant or not. This decision is taken on the basis of conditions discussed in Section 4.5.6(b). The vacant module returns 1 if a place is found vacant, otherwise 0. Based on the place's vacancy the percentage place occupancy is calculated as its compliment. The Tag pop-up window can be used at any time during simulation for place utilization. The utilization file contains these information besides some others.

## 4.6 Data structures

The TPNS tool is written using the SunView library. SunView is a notification based [20, 24] based event driven environment. Therefore, all the procedures are written keeping the underlying notifiere in mind. During the initial setup all the procedures are registered with the notifier and then the notifier invokes each of them when the related event occurs. The basic requirement to write any software is the definition of the data structures which will hold the details of different objects. For TPNS we mainly rely on the data type “structure” used in C language. Petri net places and transitions are described in linked lists. Following is description of each data structure.

---



---

### 4.6.1 Places

The data structure used to hold details of a place is as follows :

```
struct place
{
    int place_no;           /* Place number of the place */
    int no_of_tokens;       /* Number of tokens in place */
    int pl_loc_x;           /* Place location X          */
    int pl_loc_y;           /* Place location Y          */
    int tokens_in;          /* tokens entered the place in 1 step*/
    int total_tokens_in;    /* total tokens entered the place */
    int avt;                /* accumulated vaccant time */
    int vls;                /* vaccant last time */
    int nvst;               /* Not vaccat start time*/
    char tag[7];            /* Place tag                  */
    char comment[60];       /* Place comment              */
    struct place *next_place; /* pointer to next place     */
};
```

The first integer `place_no` hold the place number of the object. This number is automatically assigned to the place at the time of drawing. The fields `pl_loc_x` and `pl_loc_y` hold the positioning details of the place in the xy plane. The integer `no_of_tokens` holds the number of tokens in a place at any moment. The fields `tokens_in`, `total_tokens_in`, `avt`, `vls` and `nvst` are used for calculating the number of tokens entered a place during a run and the total vaccant time of a place. For definition of place vacancy refer to Section 4.5.6. A tag field hold the tag and the comment field hold any definition or description related to the place. The structure `*next_place` is a pointer which points towards the next place in the linked list.

---

---

## 4.6.2 Transitions

The data structure used to hold details of a transition is as follows :

```
struct transition
{
    char orient;                /* Orientation of transition */
    char trans_type;            /* Type of transition      */
    char trans_name[20];        /* Immediate or Timed trans */
    char tag[7];               /* transition tag          */
    char comment[60];          /* transition comment      */
    float utilization;          /* Transition utilization */
    int tr_loc_x;              /* transition location X   */
    int tr_loc_y;              /* transition location Y   */
    int priority;              /* transition firing priority*/
    int conflicts;             /* How many times went to conflicts*/
    int conflict_success;      /* How many times survived */
    int total_delay;           /* Paramete used for utilization */
    int total_firing;          /* No of times transition fired */
    int trans_delay;           /* transition delay */
    int remaining_ticks;       /* remaining ticks while waiting */
    int trans_no;              /* transition number */
    int avg_delay;             /* Avg. delay (Stochastic) */
    int waiting_tokens;        /* Tokens wiaiting in timed trans */
    struct arc *first_arc_in;   /* arc coming in transition */
    struct arc *first_arc_out;  /* arc going out transition */
    struct transition *next_trans; /* pointer to next transition*/
    struct transition *next_rdy_trans; /* pointer to next ready tran*/
};
```

### Character fields :

The orient field holds the orientation of the object (v: vertical, h: horizontal). The trans\_type hold the type of transition (t: timed, i: immediate). The tag field hold the tag which is assigned to a transition by the tool (Tx, where x is the transition number). The

---

---

comment field is 60 characters long and can describe function of a transition or contains remarks about it.

### Integer fields:

The `tr_loc_x` and `tr_loc_y` hold the positioning details of the transition in the xy plane. `Priority` holds the priority of a transition to fire in case of conflict. The field `conflicts` contain hold the number of times a transition was in conflict where as `conflict_success` holds the number of times it won the conflict. The field `total_delay` is used to calculate the total time a transition was in wait state (i.e. ticking). The `total_firing` field determines the number of times a transition fires. The field `trans_delay` is only used for transitions with deterministic delays. The `remaining_ticks` field is used during simulation. This field is used to keep the ticking transitions in a net up to date. The `trans_no` field holds the number assigned by the simulator to a transition. The `avg_delay` field is used by time transitions with discrete stochastic delays. The `waiting_tokens` field is used to keep track of the number of tokens absorbed by a transition. This field is set to one whenever a time transition is enabled. It is set to zero when the transition fires. The last four fields in the transition structure are pointers to the structures of the first arc coming in, the first arc going out, the next transition, and the structure of next ready transition. A float field is used for utilization. This contains the transition occupancy as a percentage of the total time for which a net is simulated.

### 4.6.3 Arcs

Arcs are stored in a data structure which is as follows :

```
struct arc
{
    struct place *arc_inout;
    struct arc *next_arc;
};
```

The `struct place *arc_inout` holds the address of the place to which this arc is going out or coming from. `Struct arc *next_arc` is a pointer to the next arc in the linked list. Besides the data structures described above which are essential to find the interrelationship

---

---

of different objects, there is another data structure which is important for editing purposes. It is as follows :

```
struct del_arc
{
    int pl_no;
    int tr_no;
    int x1;
    int y1;
    int x2;
    int y2;
    struct del_arc *other_del_arc;
    struct del_arc *next_del_arc;
};
```

The integers x1, y1 and x2, y2 holds coordinates of two ends of a line. Integers pl\_no and tr\_no hold the place number and transition number of the place and transition which are connected by this line. The struct del\_arc \*other\_arc points towards the last or first member of the structure which belongs to the same arc and struct del\_arc \*next\_del\_arc points to the next member of the linked list. This is a linked list of all the line segments drawn to connect a place to a transition.

#### 4.6.4 Cycles

Cycles are stored in a data structure which is as follows :

```
struct cycle
{
    int cycle_no;           /* The number of cycle */
    int cycle_time;         /* Time of the day at which it occurred */
    struct cycle *next_cycle; /* Pointer to the next cycle */
};
```

This technique is used to save these cycle structures because it provides dynamic storage. In this way memory is conserved from being pre-assigned using arrays. The

---

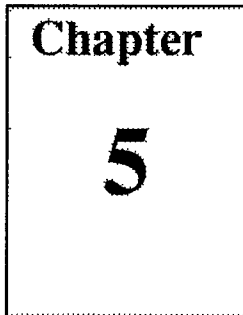
---

cycle\_no holds the number of cycle. The cycle\_time holds the time of the day at which that cycle is detected. Knowing the elapse time can be helpful for finding time between cycles. The thir field is the pinter to the next structure \*next\_cycle.

---

---

# DESCRIPTION AND WORKING OF TPNS TOOL



## 5.1 General Description

The Timed Petri Net Editor and Simulator (TPNS) runs on SUN3 and SUN4 workstations under suntools utility based on SunView providing a graphical user interface. Figure 5.1 shows the appearance of the tool on the workstation screen, a high resolution bit-mapped CRT with  $1152 \times 900$  pixels. The upper window is a panel which governs the editor. The big window below is the working space of the editor. The working space has two scrollbars (one horizontal and one vertical) to move the visible part of the main window over the whole net space. The grey bar in the scroll bar represents the visible part of the whole size, and can be moved by clicking the right or left button of the mouse at the point at which one wants the center of the visible part. In the following sections the acronyms LMB, MMB, RMB will be used to indicate the left, middle, and right mouse buttons, respectively. A header displays the name of the tool "A Graphical Timed Petri Nets Simulator"

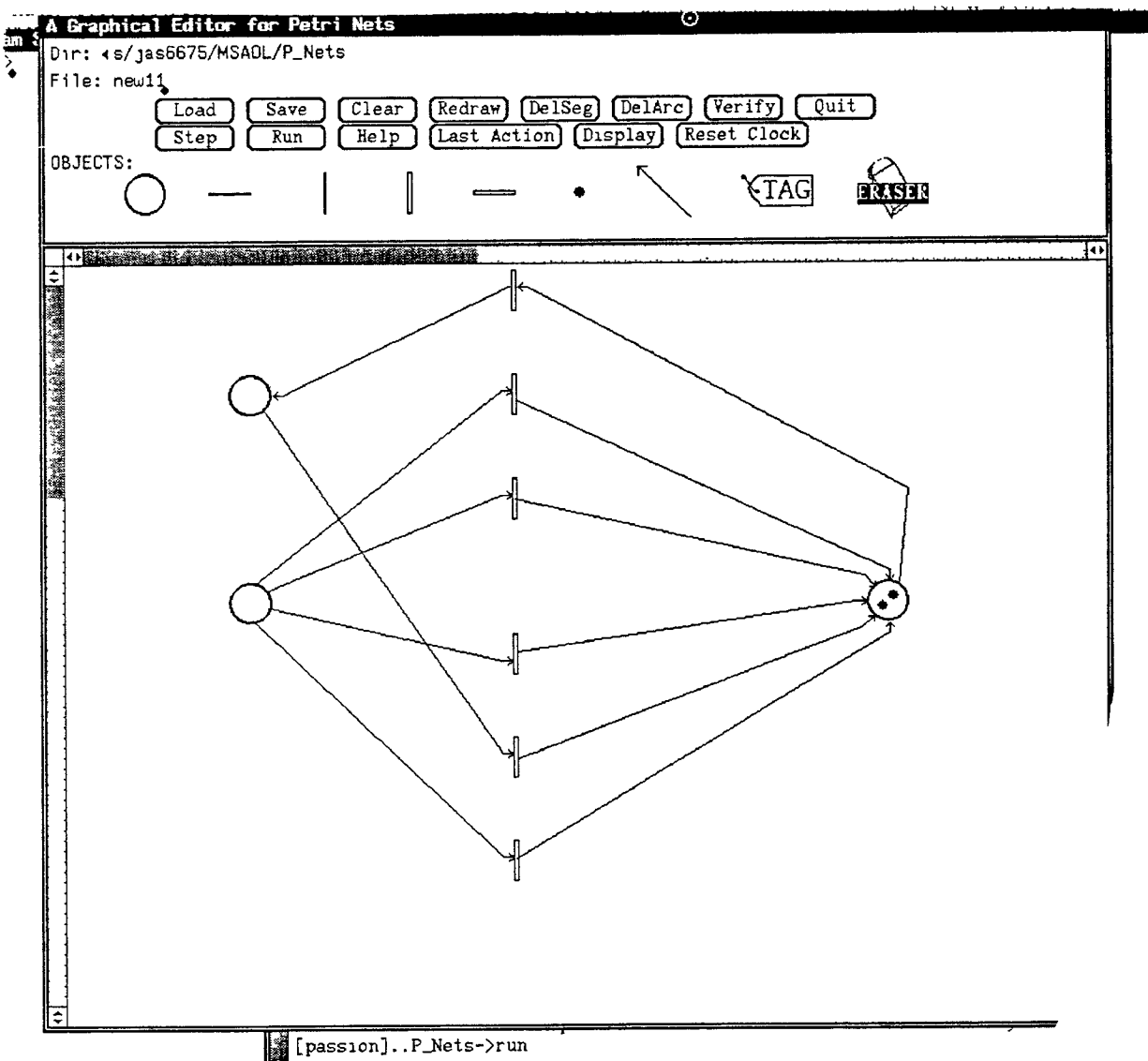


Figure 5.1: TPNS tool Panel and Canvas windows

---

## 5.2 Net Management

The Panel window contains many items. When the tool appears on the screen the `directory` item appears in the top left corner of the Panel window. This item is by default set to the current directory. The user can edit this item to specify a different path. This directory path is used for loading or saving files. This path is read by a SunView library function as follows :

```
string_variable = panel_get_value(dir_item)
```

This `string_variable` can be used to specify the location for saving or loading a file. The second item that appears below the `directory` item is the `File name` item. This is an empty string of length described by `MAX_FILENAME_LEN` defined in the beginning of the main program file (`tpns.c`). The user is required to type in the filename. This file name is read into a `string_variable` as follows :

```
string_variable = panel_get_value(fname_item)
```

## 5.3 Panel Buttons

There are fourteen buttons on the Panel for Petri nets editing and simulation and results display. These are Load, Save, Clear, Redraw, DelSeg, DelArc, Verify, Step, Run, Help, Last Action, Display, Reset and Quit. The following section discusses these buttons and their functions in more detail.

### 5.3.1 Load

When the LMB is clicked on the this button , it loads the file described in the `File name` item into the net working area. Loading a file clears any previously edited net. Loading a net is a three step process integrated into one step. The following sequence is followed:

1. The `clearall_proc` is called to clear any previous net structurere from memory and screen.
-



- 
2. Loads the file into memory using the `load_proc`.
  3. Draws the net on the canvas using `redraw_proc`.

If there is no file name specified or there already exists a file in memory the user is warned. When this button is clicked with a loaded net on the canvas, the simulator searches for the generated files (log, verify, utilization, and marking) of the most recently loaded net (if there was a net loaded before the current net). If these files exist, then the user has the choice to delete them or keep them. The purpose of this is to immediately free disk space.

### **5.3.2 Save**

When the LMB is clicked on the Save button, it saves the net drawn in the canvas window with the file name specified in the file item. If there is any previous version of that file the, user is prompted about it. If the user still continues then a backup copy of the older version is saved with the same file name and an extension of “bak”. This prevents accidental loss of important data.

### **5.3.3 Clear**

When this button is clicked, the whole net working space is cleared. Any information not saved is lost. If there exists a net on the canvas, the user is notified.

### **5.3.4 Redraw**

This option may be needed to redraw the whole net during an edit session. Clicking the LMB on this button will redraw the whole net. This can be used to redraw arc connections to a timed transition if transition type is changed or refill the fine connections at the intersection of arcs or redrawing the arrow-heads. The redraw function will refer to the loaded net in memory for redrawing a net.

---

---

### 5.3.5 DelSeg

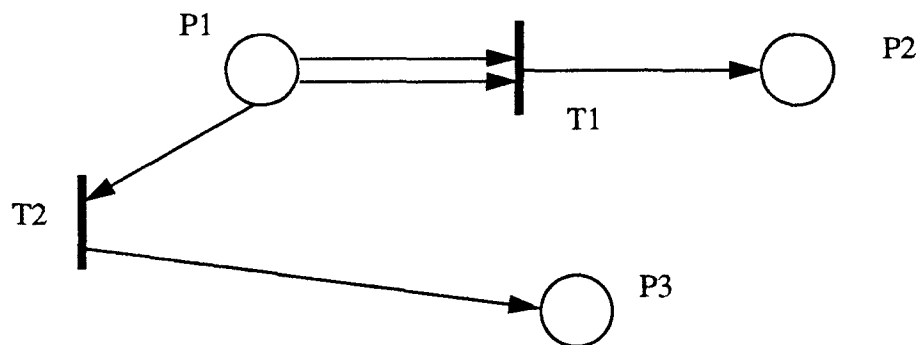
This button is used to delete the last line segment drawn. This button is primarily used for editing while drawing. Every time this button is clicked, a segment is deleted.

### 5.3.6 DelArc

This button is used to delete an entire arc. To delete an arc the LMB is clicked once at one end and then at the other end of the arc. When the first arc end is successfully selected, a message pop-up instructs the user to select the other end. Once the second end of the arc is selected a message pop-up prompts 'press the RMB'. The position of the cursor is not important while clicking the RMB. Clicking the RMB once will delete the entire selected arc.

### 5.3.7 Verify

This button is used to print the connection matrix between the transition and places in the parent window. A user may draw a net with arc connections that looks correct on the canvas, but the stored structure is incorrect. This matrix can be used to verify whether the net drawn in the canvas is interpreted correctly by the tool or not.



**Figure 5.2 :** A sample net with 3 places and 2 transitions

---

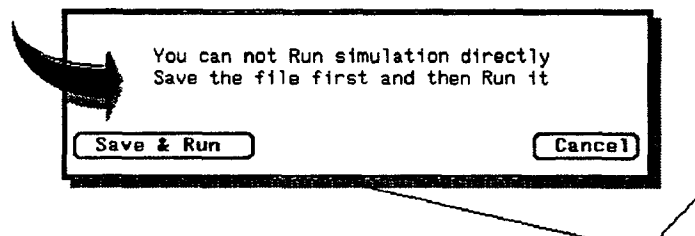
**Table 5.1 : Connection matrix for net in Figure 5.2**

Transition #	Input places	Output places
1	1	
1	1	
1		2
2	1	
2		3

This shows that transition T1 has 2 input arcs from place P1 and one output arc to place P2. Transition T2 has one input arc from place P1 and one output arc to place P3. If this table does not agree with user intentions then check for errors. If a net is loaded with erroneous connection information, an error message pops up indicating the error and its location.

### 5.3.8 Step

This button is a part of the simulator. When a net is loaded into the canvas it can be stepped. Simulation using Step mode can not be carried out immediately after drawing a net on the canvas. The drawing has to be saved first and then loaded before simulation. If the user requests to simulate a net in Step mode immediately after drawing a net, he is given a choice to save the file with a temporary name "temp.pic" ( Figure 5.3) or select the Save option from the Panel window by specifying a file name in the File name item.

**Figure 5.3: Petri Net saved with a temporary name**

---

Once the file is reloaded, simulation can be carried out. There can be only timed transitions (status =1) or immediate and timed (status = 2), or immediate transitions only (status =3) in the net. The net status is determined before execution of a step. Based on the net status the corresponding section of the step procedure is called. The Step is defined in three different ways depending on the net status.

#### **Timed transitions only (status = 1)**

If there are only timed transitions in the net, then the following sequence of steps are taken to complete one step :

1. Resolve conflicts on the basis of priority or random selection.
2. Update the list of potentially enabled transitions. The updated list contains transitions which are independent (not in conflict).
3. Enable ready timed transitions. Remove input tokens and add tokens to timed transitions. These timed transitions are displayed as solid thick rectangles.
4. Update linked list of timed transitions by sorting in an ascending manner.
5. Update the remaining ticks for the ticking transitions.
6. Fire the timed enabled transitions with least delay and place output tokens.
7. Advance CLOCK by the delay of the timed transition fired.
8. Report activity to logfile (if called from RUN procedure).

#### **Timed and immediate transitions (status = 2)**

If there are both timed and immediate transitions in a net, then the following execution sequence is followed to complete a step :

1. Resolve conflicts on the basis of priority or random selection.
  2. Update the list of potentially enabled transitions.
  3. Fire all enabled immediate transitions.
  4. If the firing of the immediate transitions enables new immediate transitions repeat steps 1 to 4.
  5. If there are no immediate transitions enabled and there exist enabled timed transitions then follow steps 3 to 7 as in (status = 1) for timed transitions only.
-

- 
6. Report activity to logfile (If called from RUN procedure).

### Immediate transitions only (status = 3)

If there are immediate transitions only in the net, then the `fire_immediate` procedure will remove tokens from input places and deposit them in the output places [21]. In this case the `CLOCK` is never advanced.

## 5.3.9 Run

This button is a part of the simulator. All the conditions for running simulation in Step mode apply to Run mode. In Run mode, the Step procedure is called repeatedly until a user specified terminating condition (Figure 5.4 (a)) is reached or a deadlock (Figure 5.4 (b)) occurs. When Run mode is selected to simulate a net, the simulator searches for any previously created files (log, verify, utilization, and marking) for the current net. If these files are found then the user is prompted with the options menu of Figure 5.5.

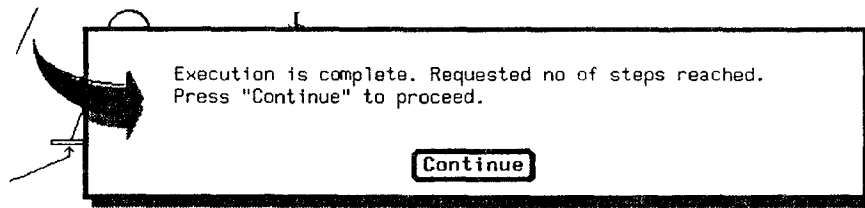


Figure 5.4 (a) : User specified termination condition reached.

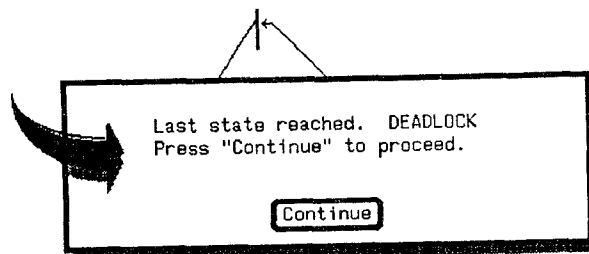
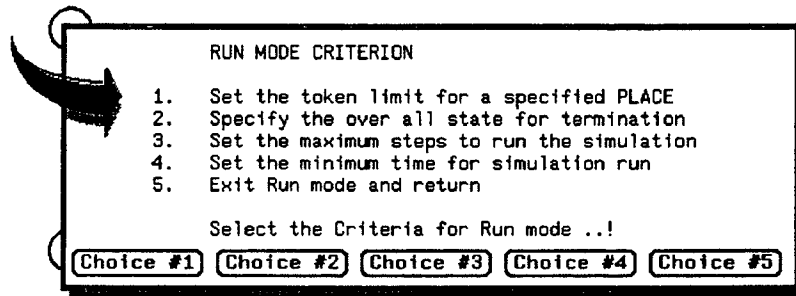


Figure 5.4 b) : A Petri Net reached a dead lock state

---



**Figure 5.5 : Run mode options**

If the termination condition is reached or a deadlock occurs a window pops up with termination message and the simulation status. There are three files that are created as a result of running simulation. These are (a) a log file, (b) a verify file (connection matrix), (c) a utilization file, and (d) a marking file. These files can be displayed after completion of net simulation using the Display button.

### 5.3.10 Help

This button is used for accessing an on-line help facility designed to provide documentation about the TPNS tool, and helping the user in its usage. This is a fast and relevant method of learning to use the TPNS tool. When the user clicks this button a pop-up window will appear with a Panel window and a tty subwindow (Figure 5.6). The Panel window contains the help topics as button tags. The user can click the button with the required help topic and the help text will be displayed in the tty subwindow (Section 3.4).

### 5.3.11 Last Action

This button can be useful during net drawing. If it is clicked, it displays in a small pop-up the last button clicked and the last object selected.

### 5.3.12 Display

This button is used to display generated files due to simulation. Clicking this button pops up a window with four the different options shown in Figure 5.7.

---

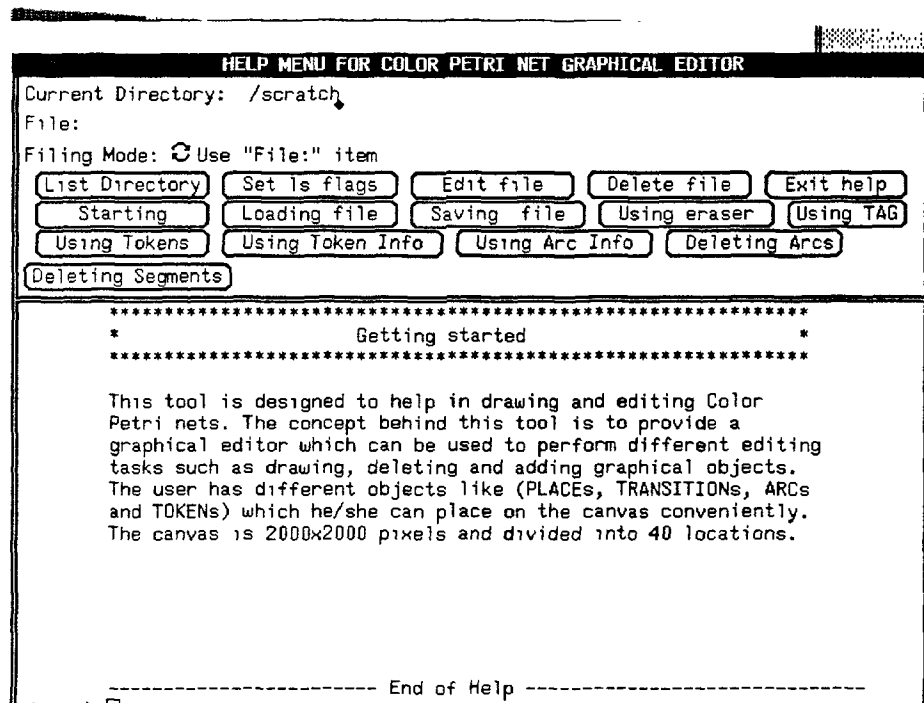
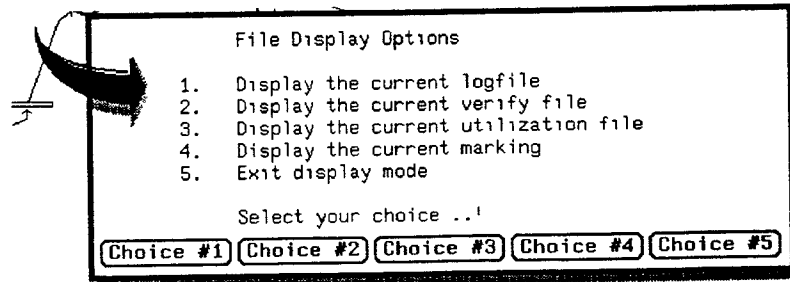


Figure 5.6: An on-line Help window



**Figure 5.7:** Display menu

The “current” is defined as the file whose name appears in the File name item in the Panel item. Once the required option is selected it is displayed in a tty subwindow.

### 5.3.13 Reset

This button is used to reset the utilization parameters associated with every place and transition. The same button is used to reset the CLOCK. When simulation is carried out in Run mode utilization parameters (Section 3.2.4) are calculated and updated. These are, for example, percentage of transition occupancy, number of times a transition was in conflict, how long a place was vacant, etc. It is left to the user to decide to allow these parameters to accumulate or to reset them at some point during simulation. A user may want to run simulation many times with a different set of marking and/or transition attributes and study the impact of certain parameters on the net behavior and utilization. It should be noted that resetting the utilization parameters resets the CLOCK automatically.

### 5.3.14 Quit

This button is used to quit the TPNS tool. It requires confirmation.

---



---

## 5.4 Panel Objects

The available objects are (from left to right) places, transitions, arcs, tokens, tags, and eraser.

### 5.4.1 Places

Places (represented by circles) are characterized by a name (tag) that defaults to  $P_x$  (where  $x$  is the number of place). These places when drawn on the canvas default to zero initial marking (i.e. zero tokens). A comment about what a place represents or its function in the net is optional (Section 5.4.5).

### 5.4.2 Transitions

Transitions (represented by horizontal or vertical bars) are characterized by a name (tag) that defaults to  $T_x$  (where  $x$  is the transition number). There are two types of transitions :

1. Immediate transitions (solid single bar)
2. Timed transitions (hollow rectangles)

There are many parameters as discussed in Section 4.5.2, associated with a transition. These can be modified through a Tag pop-up window as shown in Figure 5.8. To draw a timed transition the user has to specify the type of transition delay as deterministic or stochastic. A transition delay for deterministic or an average transition delay (Average transition delay =  $1/\lambda$ , where  $\lambda$  =firing rate) for a stochastic transition is entered at the time of drawing a net (Figure 5.9).

---

---

Item Tag : T3      Fired: 5      CLOCK : 26

Comments\* : Time reqd. by conveyor belt to transport material to WS1

Trans. Priority\* : 0      Utilization : 46.15%

Transition delay\* : 0      Transition type: Timed

Trans Delay Type : Stochastic.      Avg. delay is : 3

Remaining Ticks : 0      Waiting Tokens : 0

\*These items are editable. Make the change & hit -->

**Figure 5.8:** A Tag pop-up window

(Avg. delay = 1 / firing rate)

Average Delay is always truncated. The fractional part is truncated and only the integer part is used. For more precise results use smaller units of time.

Enter average delay (INTEGER):

**Figure 5.9:** A data entry pop-up for average delay of a stochastic transition

Integer values are used for transition delay. This eliminates the need for fractional time units and synchronizes firing with the CLOCK. The user is recommended to use a smaller time unit for more precise results. If a user selects a time transition to draw on the canvas he is forced to enter an integer value greater than zero for transition delay.

### 5.4.3 Arcs

Arcs are characterized by a type (Input or Output), a place and a transition. These arcs are used to connect places and transitions.

---

---

### **5.4.4 Tokens**

Tokens (represented by small solid circles). Their distribution in places constitutes a net's marking.

### **5.4.5 Eraser**

This object is used for editing purposes. It can be used to erase other objects such as places, transitions, and tokens).

---

# EXAMPLE APPLICATIONS

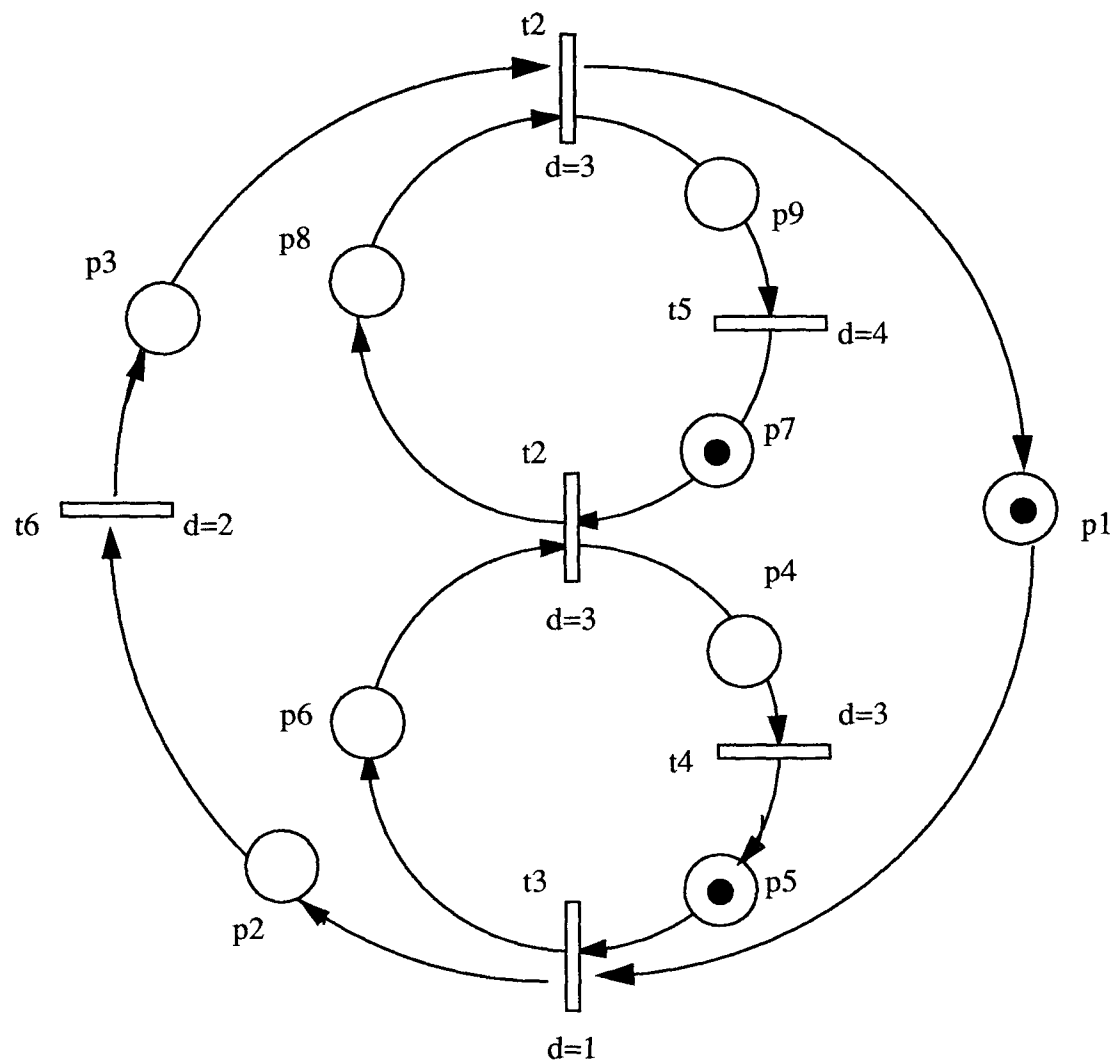
**Chapter**  
  
**6**

## 6.1 An example of Timed Net

Zuberek[8] in 1980 studied an example for Timed Petri nets. The current TPNS model conforms with Zuberek's model and satisfactory simulation results for the net of Figure 6.1 confirm this. The behavior of a certain class of timed nets can be visualized by a firing diagram represented by a set of time semiaxes, each semiaxis corresponding to a particular *transition*. Such a diagram for the timed net of Figure 6.1 is shown in Figure 6.2. The succession of firings is marked by arrows. It can be observed that the sequence of firing is cyclic and the state of the net at the step "S9" is the same as that at step "S4". It is also seen that the states of the net are subject to changes at those moments only in which firings initiate or terminate.

When the above example is drawn using TPNS and is simulated in Run mode Zuberk's results are confirmed. Figures 6.3 to 6.5 show the generated output files. This example shows the conformance of deterministic timed Petri nets in TPNS model with Zuberek's work.

---



**Figure 6.1 :** Working example of a Timed Petri net.

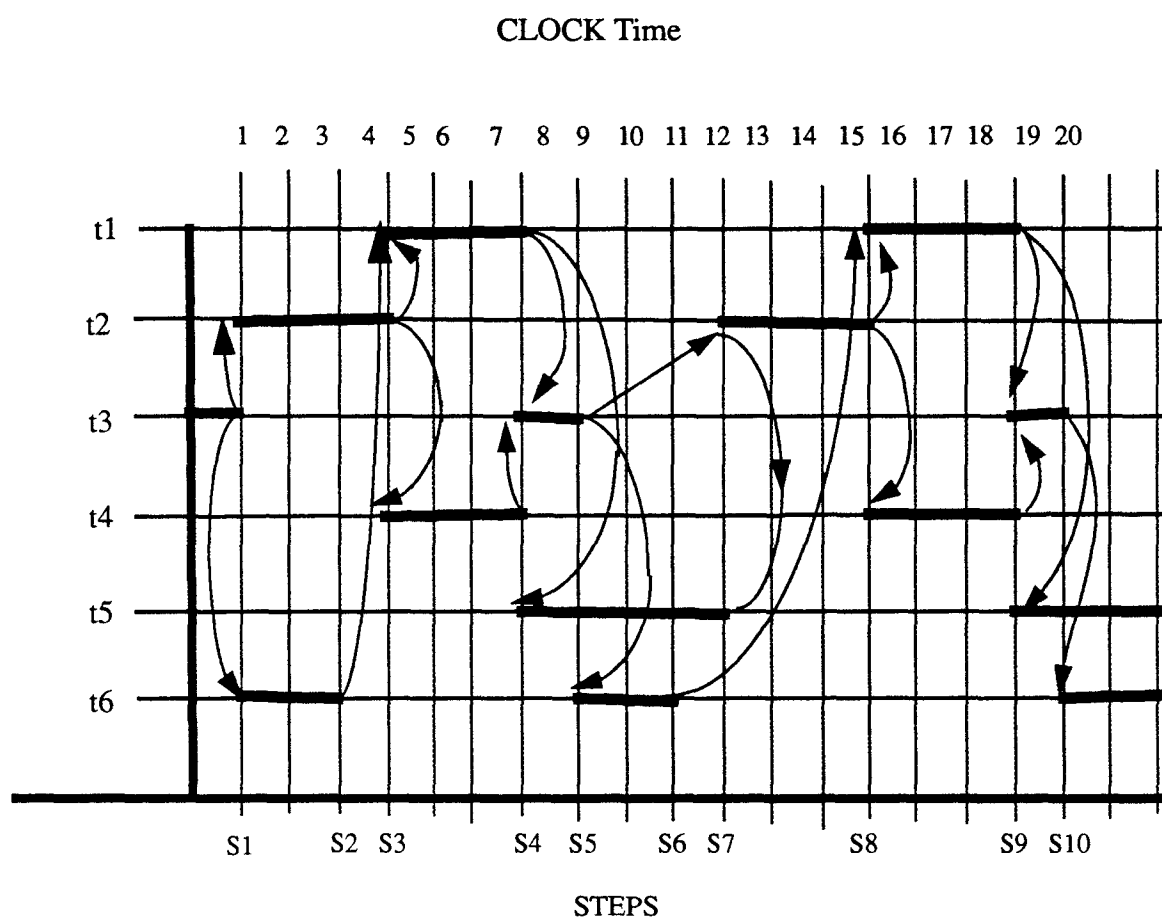


Figure 6.2. Firing diagram

```

*****
                        NET UTILIZATION AND BEHAVIOR
*****

CLOCK reached ..... : 38
Total number of steps : 20
Petri Net File name :  zub-2.utilize
Mon Dec 16 09:13:05 EST 1991
*****
                        Transition Utilization / Behavior
*****

```

Transition	Fired	Busy*	in Conflict	Success
t1	4	31.58%	0	0
t2	4	31.58%	0	0
t3	5	13.16%	0	0
t4	4	31.58%	0	0
t5	3	42.11%	0	0
t6	4	21.05%	0	0

```

*   computed as : (total transition delay X 100) / total simulation time.
*****
                        Place Utilization / Behavior
*****

```

Place	Total number of tokens entered this Place	% Duration for which Place was occupied
p1	8	84.21%
p2	5	21.05%
p3	4	65.79%
p4	4	31.58%
p5	4	13.16%
p6	5	55.26%
p7	3	31.58%
p8	4	31.58%
p9	4	34.21%

```

*   Computed as : (total place occupancy X 100) / total simulation time.
*****

Initial marking not reached during net execution.

```

**Figure 6.3 : The utilization file of Fig. 6.1**

---

```

*****
Petri Net File name :  zub-2.mark
Mon Dec 16 09:04:51 EST 1991
*****
*****
Net Marking
*****
Step p1  p2  p3  p4  p5  p6  p7  p8  p9      t1  t2  t3  t4  t5  t6
1:   0   1   0   0   0   1   1   0   0      0   0   0   0   0   0
2:   0   0   1   0   0   0   0   0   0      0   1   0   0   0   0
3:   0   0   1   1   0   0   0   1   0      0   0   0   0   0   0
4:   2   0   0   0   1   0   0   0   1      0   0   0   0   0   0
5:   1   1   0   0   0   1   0   0   0      0   0   0   0   3   0
6:   1   0   1   0   0   1   0   0   0      0   0   0   0   1   0
7:   1   0   1   0   0   1   1   0   0      0   0   0   0   0   0
8:   1   0   1   1   0   0   0   1   0      0   0   0   0   0   0
9:   3   0   0   0   1   0   0   0   1      0   0   0   0   0   0
10:  2   1   0   0   0   1   0   0   0      0   0   0   0   3   0
11:  2   0   1   0   0   1   0   0   0      0   0   0   0   1   0
12:  2   0   1   0   0   1   1   0   0      0   0   0   0   0   0
13:  2   0   1   1   0   0   0   1   0      0   0   0   0   0   0
14:  4   0   0   0   1   0   0   0   1      0   0   0   0   0   0
15:  3   1   0   0   0   1   0   0   0      0   0   0   0   3   0
16:  3   0   1   0   0   1   0   0   0      0   0   0   0   1   0
17:  3   0   1   0   0   1   1   0   0      0   0   0   0   0   0
18:  3   0   1   1   0   0   0   1   0      0   0   0   0   0   0
19:  5   0   0   0   1   0   0   0   1      0   0   0   0   0   0
20:  4   1   0   0   0   1   0   0   0      0   0   0   0   3   0

```

---

**Figure 6.4 :** The instantaneous marking file



---

```

*****
Petri Net File name :  zub-2.log
Mon Dec 16 09:04:41 EST 1991
*****
Step:   1 of 20
Timed Transition enabled : 3,
Timed Transition fired   : 3,
CLOCK advanced by..... : 1
CLOCK has reached..... : 1
Total tokens in the net  : 3
Net marking ..... : (0, 1, 0, 0, 0, 1, 1, 0, 0)
Ticking transitions .... :
*****
Step:   2 of 20
Immediate Transitions fired: None.
Timed Transition enabled : 2, 6,
Timed Transition fired   : 6,
CLOCK advanced by..... : 2
CLOCK has reached..... : 3
Total tokens in the net  : 1
Net marking ..... : (0, 0, 1, 0, 0, 0, 0, 0, 0)
Ticking transitions .... : (t2,1)
*****
Step:   3 of 20
Timed Transition enabled : None.
Timed Transition fired   : 2,
CLOCK advanced by..... : 1
CLOCK has reached..... : 4
Total tokens in the net  : 3
Net marking ..... : (0, 0, 1, 1, 0, 0, 0, 1, 0)
Ticking transitions .... :
*****
Step:   4 of 20
Immediate Transitions fired: None.
Timed Transition enabled : 1, 4,
Timed Transition fired   : 1, 4,
CLOCK advanced by..... : 3
CLOCK has reached..... : 7
Total tokens in the net  : 4
Net marking ..... : (2, 0, 0, 0, 1, 0, 0, 0, 1)
Ticking transitions .... :
*****
Step:   5 of 20
Immediate Transitions fired: None.
Timed Transition enabled : 3, 5,
Timed Transition fired   : 3,
CLOCK advanced by..... : 1
CLOCK has reached..... : 8
Total tokens in the net  : 3
Net marking ..... : (1, 1, 0, 0, 0, 1, 0, 0, 0)
Ticking transitions .... : (t5,3)
*****

```

---

---

```

Skipping steps 6 to 17 ...
*****
Step:   18 of 20
Timed Transition enabled : 2,
Timed Transition fired   : 2,
CLOCK advanced by..... : 3
CLOCK has reached..... : 34
Total tokens in the net  : 6
Net marking ..... : (3, 0, 1, 1, 0, 0, 0, 1, 0)
Ticking transitions .... :
*****
Step:   19 of 20
Immediate Transitions fired: None.
Timed Transition enabled : 1, 4,
Timed Transition fired   : 1, 4,
CLOCK advanced by..... : 3
CLOCK has reached..... : 37
Total tokens in the net  : 7
Net marking ..... : (5, 0, 0, 0, 1, 0, 0, 0, 1)
Ticking transitions .... :
*****
Step:   20 of 20
Immediate Transitions fired: None.
Timed Transition enabled : 3, 5,
Timed Transition fired   : 3,
CLOCK advanced by..... : 1
CLOCK has reached..... : 38
Total tokens in the net  : 6
Net marking ..... : (4, 1, 0, 0, 0, 1, 0, 0, 0)
Ticking transitions .... : (t5,3)
*****

```

---

**Figure 6.5:** The log file created during simulation of net in Fig. 6.1

---

## 6.2 Application of TPNS to FMS

In the following section the TPNS tool is used to simulate and analyses a real time application of Flexible Manufacturing System (FMS). This example application is a factory floor model of a flexible manufacturing system. An Automatic Guided Vehicle (AGV) is used to transport material between work stations and processors. These processors operate on the parts supplied by the AGV and transport them to the delivery locations. To increase productivity and cost ratio one AGV is used to transport material in two different directions. A central storage location is used to store the raw material. This raw material is moved to the workstations through conveyor belts. When the AGV becomes available it is used to transport the material to machine 1 and machine 2. The processed parts are moved to output places. All these activities require a finite amount of time and is modeled accordingly.

### 6.2.1 System Representation

All these activities require a finite amount of time and is modeled accordingly. The block diagram of Figure 6.6 is used to represent the underlying system and its activities. The Petri net model for this system is given in Figure 6.7. Places and transitions are represented as in Table 6.2 and 6.3.

### 6.2.2 System Operation

The supply of the raw material is carried by conveyor belts. These conveyor belts which are represented by stochastic transitions (t1 and t3) with an average delay of 3 and 4 time units. The reason for this declaration of transition delay as exponentially distributed is the fact that supply to these locations are exponentially distributed. The objective is to maximize the use of the AGV by adjusting the speed of the conveyor belt such that the material does not reach workstation 1 and 2 simultaneously. This will also eliminate frequent conflict for AGV. When a token arrives in place p1 or p3 (Workstation 1 or 2), the AGV has to be available in order to fire immediate transitions t1 or t2. The availability of AGV is represented by a token in place p4. In many instances t1 and t2 fire independently. In case of conflict, prioritized or random selection is carried out. Therefore, a con-

---

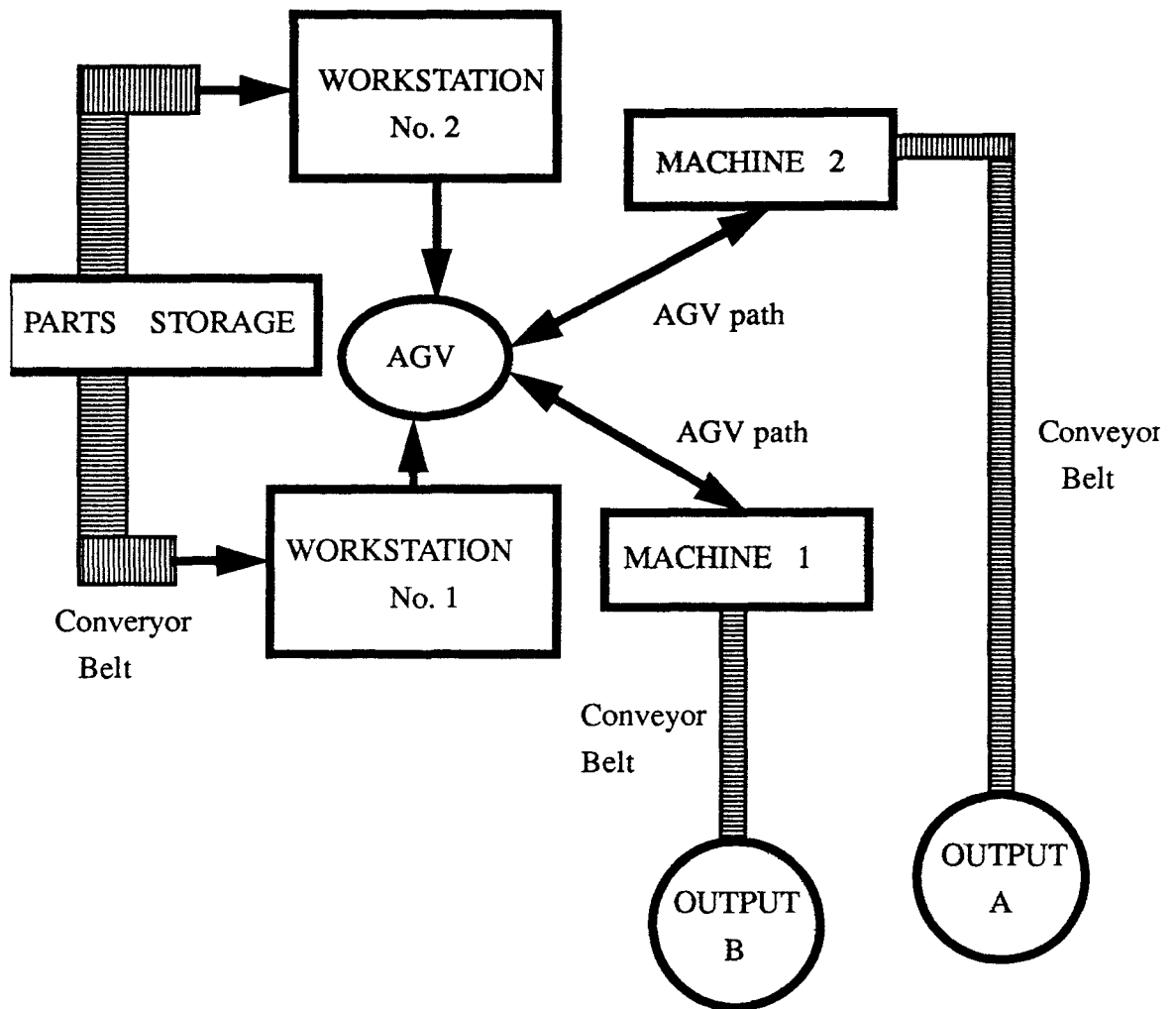
flicting situation is also modeled in this example.

**Table 6.1:** Places and their representation in FMS application of Figure 6.6 & 6.7

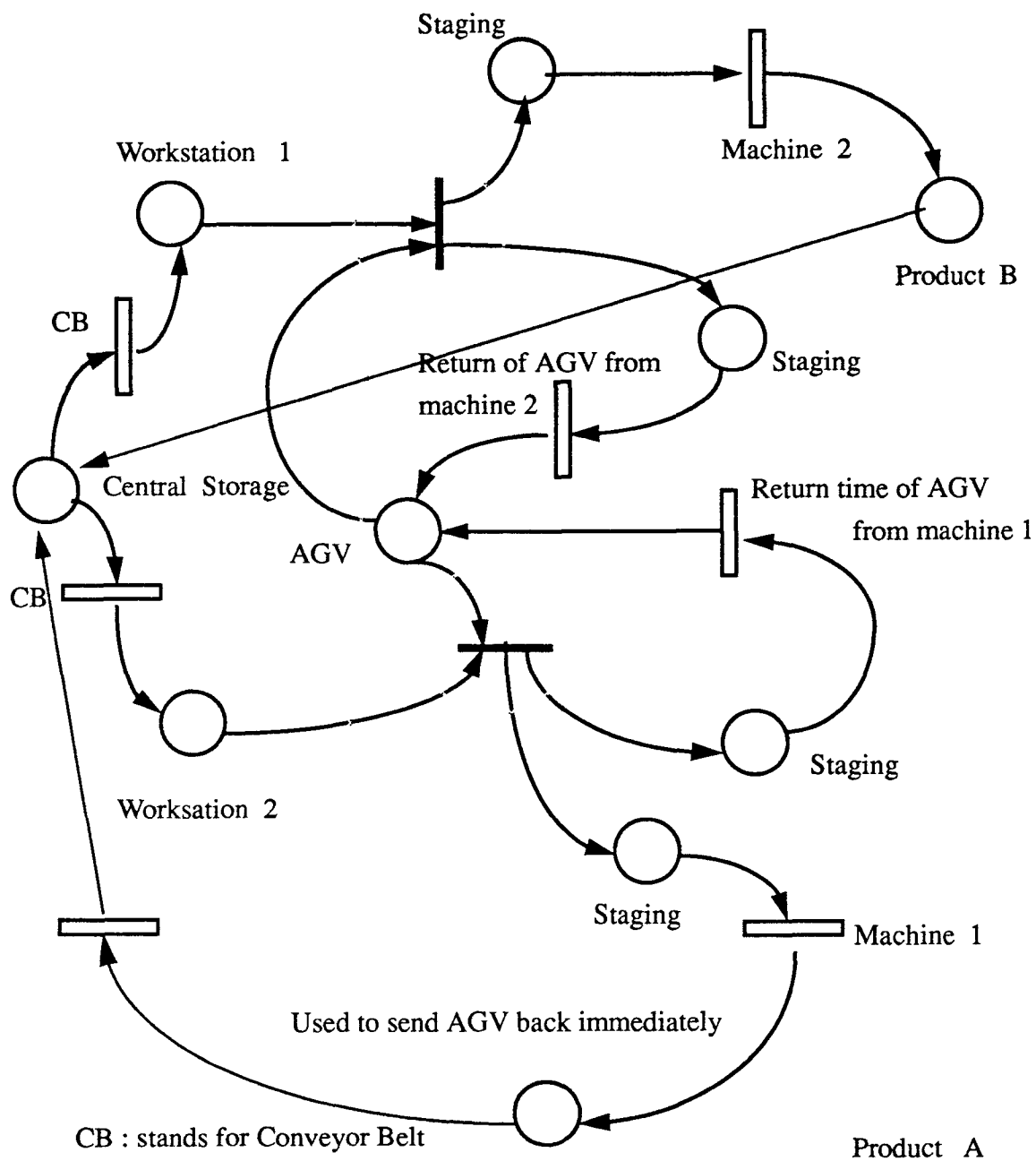
Place #	Represents
p1	Work station number 1
p2	Work station number 2
p3	Central Storage of raw material
p4	AGV available
p5	Staging place for AGV transportation
p6	Staging place for AGV transportation
p7	Staging place for Product A transportation
p8	Output location of product A
p9	Staging place for Product B transportation
p10	Output place for Product B transportation

**Table 6.2:** Transitions and their representation in FMS of Figure 6.3

Transition #	Represents	Type
t1	Used to immediately send AGV back from machine B after it has shipped the material to machine B.	Immediate
t2	Used to immediately send AGV back from machine A after it has shipped the material to machine A	Immediate
t3	Time required by Conveyor belt to transport material from storage to workstation 2	Stochastic
t4	Time required by Conveyor belt to transport material from storage to Workstation 1	Stochastic
t5	Time required by machine 2 to process material supplied by AGV	Deterministic
t6	Time required by machine 1 to process material supplied by AGV	Deterministic
t7	Time required by AGV to return to machine 2	Deterministic
t8	Time required by AGV to return to machine 1	Deterministic



**Figure 6.6 :** A block diagram of a Flexible Manufacturing System



**Figure 6.7:** Petri net model of FMS of Figure 6.6

---

When a token is present in place p4 (representing availability of an AGV) transition t1 or t2 can fire provided firing conditions are met. These transitions along with place p5 and p6 are used for staging. Place p5 and p6 are temporary places used for sending the AGV back to the workstations area immediately after shipment of material is complete.

Transitions t1 and t8 are used to model the time required for the AGV to reach the starting point to be available for the next shipment. Transitions t7 and t8 are deterministic timed transitions with delays equal to 3 and 2 time units respectively. Places p7 and p9 are used to model availability of material for processing by machine A and B, respectively. t5 and t6 are deterministic timed transitions used to model the processing of machine A and machine B. Finally the products are deposited to place p8 and p9. This cycle goes on until all the tokens in the store (p2) are exhausted or a user requested termination state is reached.

### 6.2.3 System behavior

The TPNs tool was used to analyse the above example. Simulation results are analysed to study system behavior and utilization. Figures 6.8 to 6.10 show the simulation and utilization results. Figure 6.8 shows the utilization of transitions and places during net execution. It also shows how many times a transition was in conflict, how many times it fired, the number of tokens entered a place and place occupancy. Figure 6.9 shows the markings reached during net execution at each step. Token distribution at each step is given along with the number of remaining ticks for each ticking transition is shown for debugging and step verification purposes. Figure 6.10 shows the log file created during simulation.

In order to achieve maximum utilization of both machines 1 and 2, utilization of these machines is observed for this system with respect to the return time of AGV from machine 2 keeping all other time parameters constant. Table 6.3 shows these results which were obtained after simulating the net for 2000 steps. Figure 6.1 shows a graphical representation of these results. These results lead to the conclusion that machine 1 and 2 have their optimum utilization with return time of AGV from machine 2 equal to 4.

---

---

\*\*\*\*\*

NET UTILIZATION AND BEHAVIOR

\*\*\*\*\*

CLOCK reached ..... : 36

Total number of steps : 40

Wed Dec 11 08:29:50 EST 1991

\*\*\*\*\*

Transition Utilization / Behavior

Transition	Fired	% Busy	In conflict	Success
t1	8	20.00% ***	9	4
t2	6	15.00% ***	9	5
t3	10	38.00**	0	0
t4	9	100.00%	0	0
t5	4	97.22%	0	0
t6	4	69.44%	0	0
t7	8	66.67%	0	0
t8	5	33.33%	0	0

\* computed as : (total transition delay X 100) / total simulation time.

\*\* transition is still ticking, the value given is the total delay.

\*\*\* immediate transition. computed as: (total firing w.r.t total steps)

Place Utilization / Behavior

Place	Total number of tokens entered this Place	% Duration for which Place was occupied
p1	10	72.22%
p2	0	100.00%
p3	9	55.56%
p4	13	2.78%
p5	8	30.56%
p6	6	5.56%
p7	6	41.67%
p8	4	83.33%
p9	8	86.11%
p10	4	72.22%

\* Computed as : (total place occupancy X 100) / total simulation time.

Initial marking not reached during net execution.

**Figure 6.8 :** Utilization file created at the end of simulation in Run mode

---



```

*****
Wed Dec 11 08:28:54 EST 1991
*****
*****
Net Marking
*****
Step p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 t1 t2 t3 t4 t5 t6 t6
1: 0 36 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
2: 0 36 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
3: 1 35 0 1 0 0 0 0 0 0 0 0 0 0 3 0 0
4: 0 35 0 0 1 0 0 0 1 0 0 0 0 0 3 0 0
5: 1 34 1 0 0 0 0 0 0 0 0 0 0 5 1 1 0
6: 1 32 2 0 0 0 0 0 0 0 0 0 0 5 1 1 0
7: 1 31 2 1 0 0 0 1 0 0 0 0 2 16 4 0 0
8: 1 31 1 0 0 1 1 1 0 0 0 0 2 16 4 0 0
9: 2 31 1 1 0 0 0 1 0 0 0 0 0 14 2 3 0
10: 1 31 1 0 1 0 0 1 1 0 0 0 0 14 2 3 0
11: 1 30 1 0 0 0 0 1 1 1 0 0 1 12 0 1 1
12: 2 30 1 1 0 0 0 2 0 1 0 0 0 11 6 0 0
13: 2 30 0 0 0 1 1 2 0 1 0 0 0 11 6 0 0
14: 2 29 0 1 0 0 0 2 0 1 0 0 6 9 4 3 0
15: 1 29 0 0 1 0 0 2 1 1 0 0 6 9 4 3 0
16: 1 29 0 1 0 0 0 3 1 1 0 0 3 6 1 0 0
17: 0 29 0 0 1 0 0 3 2 1 0 0 3 6 1 0 0
18: 0 29 0 0 0 0 0 3 2 2 0 0 2 5 0 0 2
19: 1 29 0 1 0 0 0 3 1 2 0 0 0 3 5 0 0
20: 0 29 0 0 1 0 0 3 2 2 0 0 0 3 5 0 0
21: 1 28 0 0 0 0 0 3 2 2 0 0 0 3 5 0 3
22: 1 27 1 1 0 0 0 3 2 2 0 0 2 0 2 0 0
23: 0 27 1 0 1 0 0 3 3 2 0 0 2 0 2 0 0
24: 1 26 1 0 0 0 0 3 3 3 0 0 0 1 0 0 1
25: 1 25 2 1 0 0 0 3 2 3 0 0 1 0 6 0 0
26: 1 25 1 0 0 1 1 3 2 3 0 0 1 0 6 0 0
27: 2 24 1 0 0 0 0 3 2 3 0 0 0 4 5 4 0
28: 2 23 1 1 0 0 0 3 2 3 0 0 2 3 4 3 0
29: 1 23 1 0 1 0 0 3 3 3 0 0 2 3 4 3 0
30: 2 23 1 0 0 0 0 3 3 3 0 0 0 1 2 1 1
31: 3 22 2 1 0 0 0 4 3 3 0 0 0 0 1 0 0
32: 2 22 2 0 1 0 0 4 4 3 0 0 0 0 1 0 0
33: 2 20 2 0 0 0 0 4 4 4 0 0 7 3 0 0 2
34: 2 20 2 1 0 0 0 4 3 4 0 0 5 1 5 0 0
35: 2 20 1 0 0 1 1 4 3 4 0 0 5 1 5 0 0
36: 2 20 2 0 0 0 0 4 3 4 0 0 4 0 4 4 0
37: 2 19 3 0 0 0 0 4 3 4 0 0 4 0 4 4 0
38: 2 18 3 1 0 0 0 4 3 4 0 0 3 1 3 3 0
39: 2 18 2 0 0 1 1 4 3 4 0 0 3 1 3 3 0
40: 2 18 3 0 0 0 1 4 3 4 0 0 2 0 2 2 0

```

**Figure 6.9 :** A marking file that contains instantaneous state information at each step

---

```

*****
Petri Net File name :  example-1.log
Wed Dec 11 08:28:38 EST 1991
*****

Step:   1 of 40
Immediate Transitions fired:  None.
Timed Transition enabled :  3,  4,
Timed Transition fired   :  4,
CLOCK advanced by..... :  1
CLOCK has reached..... :  1
Total tokens in the net  : 38
Net marking ..... : (0, 36, 1, 1, 0, 0, 0, 0, 0, 0)
Ticking transitions .... : (t3,2)
*****

Step:   2 of 40
Immediate Transitions fired:  2
Total tokens in the net   : 38
Net marking ..... : (0, 36, 0, 0, 0, 1, 1, 0, 0, 0)
Ticking transitions .... : (t3,2)
*****

Step:   3 of 40
Immediate Transitions fired:  None.
Timed Transition enabled :  4,  6,  8,
Timed Transition fired   :  3,  8,
CLOCK advanced by..... :  2
CLOCK has reached..... :  3
Total tokens in the net  : 37
Net marking ..... : (1, 35, 0, 1, 0, 0, 0, 0, 0, 0)
Ticking transitions .... : (t4,2) (t6,3)
*****

Step:   4 of 40
Immediate Transitions fired:  1
Total tokens in the net   : 37
Net marking ..... : (0, 35, 0, 0, 1, 0, 0, 0, 1, 0)
Ticking transitions .... : (t4,2) (t6,3)
*****

Step:   5 of 40
Immediate Transitions fired:  None.
Timed Transition enabled :  3,  5,  7,
Timed Transition fired   :  3,  4,
CLOCK advanced by..... :  2
CLOCK has reached..... :  5
Total tokens in the net  : 36
Net marking ..... : (1, 34, 1, 0, 0, 0, 0, 0, 0, 0)
Ticking transitions .... : (t5,5) (t6,1) (t7,1)
*****
*****

skip steps 30 to 37 ...

```

---

```

*****

```

---

```

*****
Step:   38 of 40
Immediate Transitions fired:  None.
Timed Transition enabled : 4,
Timed Transition fired   : 8,
CLOCK advanced by..... : 1
CLOCK has reached..... : 35
Total tokens in the net  : 35
Net marking ..... : (2, 18, 3, 1, 0, 0, 0, 4, 3, 4)
Ticking transitions .... : (t3,3) (t4,1) (t5,3) (t6,3)
*****
Step:   40 of 40
Immediate Transitions fired:  None.
Timed Transition enabled : 8,
Timed Transition fired   : 4,
CLOCK advanced by..... : 1
CLOCK has reached..... : 36
Total tokens in the net  : 35
Net marking ..... : (2, 18, 3, 0, 0, 0, 1, 4, 3, 4)
Ticking transitions .... : (t3,2) (t5,2) (t6,2) (t8,1)
*****

```

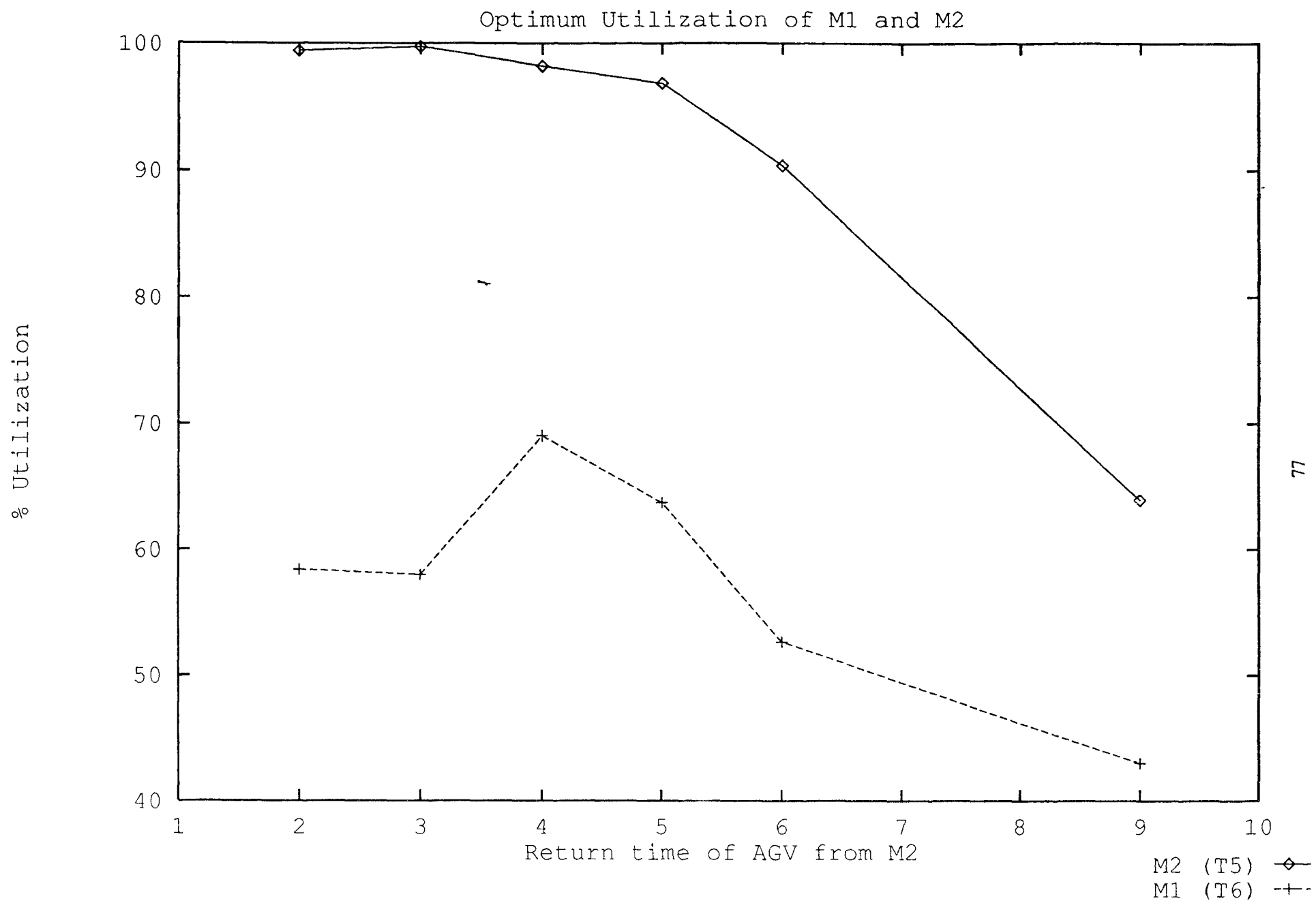
**Figure 6.10** :Parts of the log file created during simulation.

## Performance

**Table 6.3:** Simulation results after running net of Fig. 6.1 for 2000 steps.

Time required to return form machine 2	Time required to return form machine 1 (t8)	Utilization of Machine 2 (t5)	Utilization of Machine 1( t6)
2	2	99.40 %	58.39 %
3	2	99.74 %	58.00 %
4	2	98.20 %	69.01 %
5	2	96.83 %	63.70 %
6	2	90.37 %	52.63 %
9	2	63.92 %	43.02 %

**Figure 6.11:** Graphical representation of maximizing utilization of M1 & M2



---

## CONCLUSION

**Chapter**  
**7**

### 7.1 Summary

The development of TPNS is an important extension of PNS [21]. The tool is more user friendly, with its pop-up windows enhancing the graphical user interface. Extensions to the editor include automatic file backups, overwrite protection, standardized file extensions, date stamps, message and error pop-ups. An on-line help facility is designed to help understand the TPNS environment, related concepts and working of the tool.

The addition of time to PNS tool has expanded its application scope. The tool is capable of modeling systems that use time as an important characteristic. A CLOCK keeps track of the time elapse since execution started. TPNS also can handle nets with complex conflict situations such as simultaneous multiple conflicts. This has enhanced the power of the tool to model a variety of real systems.

The PNS history file is modified to provide comprehensive information about each step. New output files are produced during simulation in Run mode to enable system throughput evaluation and utilization. A marking file is created in Run mode which

---

---

contains *place* and *transition* status for a quick review of instantaneous states during net simulation.

## **7.2 Future Work**

There are many features that can be added to the tool to make it more powerful and versatile. These are discussed here.

### **7.2.1 Text Processing Program**

In Run mode the tool produces output files in text format. These contain detailed activity of the net. A text processing program which can analyze this information to produce aggregate performance measures can be added the tool.

### **7.2.2 Extensions to the Stochastic Model**

Currently the tool can handle systems with stochastic delays that have exponential distribution. This distribution is selected by default. As an extension other distribution functions such as normal and uniform can be added. The current work has provided the basic structure for such extensions.

### **7.2.3 Extensions to Conflict Handling**

The current TPNS model resolves conflicts on the basis of priority or random selection. In random selection every transition has an equal opportunity to fire. As an extension to this equal opportunity rule, the user could specify selection probabilities for transitions in conflict.

### **7.2.4 Integration of TPNS and CPNS**

The integration of TPNS with CPNS [22], can lead to a tool capable of modeling a large variety of systems.

### **7.2.5 Enhancement of On-line help**

Currently a separate program is used for the on-line help facility. This separate file is kept under a `Help` subdirectory under the current directory of TPNS, and Help is called

---

---

using C language's system call `system()`. A SunView help facility as discussed in section 3.4, would permit context dependent help.

---

# References

- [1] C. Ramachandani, "Analysis of Asynchronous Concurrent Systems by Time Petri Nets, PhD. Thesis, " MIT., July 1973.
- [2] P. Merlin, "A study of the recoverability of Computer Systems," PhD. Thesis, Computer Science Dept. University of California IRVINE, 1974.
- [3] P. Merlin, "Methodology for the design and implementation of Communication Protocols, *IEEE Transactions on Communications COM-24*, No. 6, June 1976, pp. 614-6621.
- [4] P. Merlin, Farber, D.J. , " Implication of a Theoretical Study", *IEEE Transactions on communications*," Vol COM-24, No. 9, September 1976, pp. 1036-1043.
- [5] J. Sifakis, "Use of Petri Nets in performance evaluation in measuring, modeling and evaluating computer systems," North Holland, 1977, pp. 75-93.
- [6] M. Mansche, B. Berthomieu , "Time Petri Nets for analyzing and verifying time dependent Protocols," 3rd International workshop on Protocol verification, Zurich, 1983, pp.161-172.
- [7] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proc. of IEEE on Software Engineering*, November 1989, pp. 541-580.
- [8] W. M. Zuberek, "Timed Petri Nets and preliminary performance evaluation," 7th Annual Symposium on Computer Architectures, May 1980.
- [9] W. M. Zuberek, "Application of Timed Petri Nets to Analysis of Multiprocessor Realization of Digital Filters," Proceedings of the 25th Symposium of Circuits and Systems, Houghton, Michigan, August 1982.
- [10] Miguel Menasche, "PARADE: An automated tool for the analysis of Time(d) Petri Nets".



- [11] P. Merlin, D.J. Farber, "Recoverability of Communication Protocols - Implication of a Theoretical study," *IEEE Transactions on Communications*, September 1976, pp.1036-1043.
- [12] M. K. Molloy, "Performance Modelling using Stochastic Petri Nets," *IEEE Transactions on Computers*, September 1982, C-31, No. 9, pp.913-917.
- [13] J. B. Dugan, K. S. Trivedi, R. M. Geist, V.F. Nicola, "Extended Stochastic Petri Nets: Applications and Analysis," *Proceedings of Performance 84 North-Holland*, 1984, pp.506-519.
- [14] A. M. Marsan, G. Balbo, G. Conte, "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems," *ACM Transactions on Computer Systems*, Vol 2, No. 2, May 1984, pp. 93-122.
- [15] P. J. Hass, G.S. Shedler, "Regenerative Stochastic Petri Nets," *Performance Evaluation* Vol. 6, No. 3, September 1986, pp. 189-204.
- [16] M. K. Molloy, "Petri Nets Modeling: The past, the Present, and the Future," *IEEE transactions*, 1989.
- [17] F.Feldbrugge, "Petri Net Tools," Phillips data Systems, Netherlands.
- [18] G.Chiola, "A Graphical Petri Net Tool for Performance Evaluation," *Proc. International Workshop on Modeling Techniques and Performance Evaluation*, France, March 1987.
- [19] M .A. Holliday, M. K. Vernon, "A Generalized Timed Petri Net Model for Performance Analysis," *IEEE Transactions on Software Engineering*, Vol. SE-14, no. 12, December, 1987, pp.1297-1310.
- [20] A.Gilani, "A Graphical Editor for Petri Nets," Master's Thesis, Electrical Engineering, Departement, New Jersey Institute of Technology, 1989.
- [21] Ashish Shukla, "A Petri Net Simulation Tool," Master's Thesis, Electrical Engineering Department, New Jersey Institute of Technology, 1990.

- [22] Sanjay Desai, "Colored Petri Net Simulator," Master's Thesis, Electrical Engineering Department, New Jersey Institute of Technology, 1991.
- [23] M .A. Marsan and G.Chiola, "On Petri nets with Deterministic and Exponential Transition Firing Times," in Lecture Notes in Computer Science, vol. 266, pp. 132-145, 1987.
- [24] Sun Microsystems, Inc. -SunView' Programmers Guid, 1990.
- [25] J. L. Peterson, Petri Net Theory and the Modeling of Systems, Englewood Cliffs, New Jersey: Prentice-Hall, 1981
- [26] T. Agerwala, Putting Petri Nets to work, IEEE Computer, pp. 85-94, December 1979.

## **Appendix -A**

### **(List of Program Files used in TPNS)**

The following is the list of program files used for TPNS. This list is arranged alphabetically.

- check\_cycles.c
- check\_priorities.c
- display\_file.c
- enable\_timed.c
- fire\_immediate.c
- fire\_timed.c
- net\_utilization.c
- print\_marking.c
- priority\_selection.c
- random\_selection.c
- sort\_enabled\_trans.c
- tpns.c
- trans\_modify.c
- update\_place.c
- update\_remaining\_ticks.c
- vacant.c

Description and purpose of every file is given in Section 4.5.

## Appendix -B

### (List and use of Image Files used in TPNS)

timed\_vt.image  
 timed\_ht.iamege  
 timed\_vt\_fill.image  
 timed\_ht\_fill.image  
 place.image  
 htransition.image  
 marker.image  
 dot.image  
 arc.image  
 tag.image  
 clear.image  
 eraser.image  
 vtransition.image  
 petri.image

The list of Panel objects are as follows :

Description and purpose of every file is given as follows :

#### 1. **timed\_vt.iamege**

The name abbreviates (timed transitions with vertical orientation). These are represented as follows :



#### **timed\_ht.image**

the name abbreviates (timed transitions with horizontal orientation). These are represented as follows :



**timed\_vt\_fill.image**

The name abbreviates (timed transitions with vertical orientation which is filled). This type of transition represents a timed transition that has absorbed tokens and is ticking. These are represented as follows :

**timed\_ht\_fill.image**

The name abbreviates (timed transitions with horizontal orientation which is filled). This type of transition represents a timed transition that has absorbed tokens and is ticking. These are represented as follows :

**place.image**

This is a circle which represents an image of a place. Pictorially this is shown as follows :

**htransition.image**

The name abbreviates (transition with horizontal orientation). This type of transition represents an immediate transition with zero firing time. These are represented as follows:

**vtransition.image**

The name abbreviates (transition with vertical orientation). This type of transition represents an immediate transition with zero firing time. These are represented as follows:

**marker.image**

The name is used for tokens which mark a net. This is represented as :

**eraser.image**

This is the eraser used to erase places, transitions (using LMB) and tokens in places (using RMB). This is represented as :

