ABSTRACT

A Graphical Tool for Timed Petri Nets
Using Object Oriented Programming

by
Baopu Liu

The objective of this effort is to develop a computer tool for drawing, editing and simulating Timed Petri Nets using object oriented programming. The developed C++ based Timed Petri Net Simulation Tool, TimedPNT, is capable of simulating discrete systems with both deterministic and stochastic delays. Preselection is used for conflict resolution. Performance and utilization results are automatically collected. XView™ Toolkit is used for building the TimedPNT's interactive graphical interface in compliance with AT&T's OPENLOOK® standard on a SunSparc™ IPX running SunOS™ 4.1.2. Compliance with the X Window standard makes the developed tool portable to other X Window based systems.

A GRAPHICAL TOOL FOR TIMED PETRI NETS USING OBJECT ORIENTED PROGRAMMING

by
Baopu Liu

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering

Department of Electrical and Computer Engineering

January 1994

APPROVAL PAGE

A GRAPHICAL TOOL FOR TIMED PETRI NETS USING OBJECT ORIENTED PROGRAMMING

Baopu Liu

Dr. Anthony Robbi, Thesis Advisor — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Mengchu Zhou, Committee Member — Date
Assistant Professor of Electrical and Computer Engineering, NJIT

Dr. David Wang, Committee Member — Date
Assistant Professor of Computer and Information Sciences, NJIT

BIOGRAPHICAL SKETCH

Author:	Baopu Liu
Degree:	Master of Science in Electrical Engineering
Date:	January 1994

Undergraduate and Graduate Education:

- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, USA, 1994.
- Bachelor of Science in Physical Chemistry,
  Beijing University of Science and Technology, Beijing, P.R. China, 1989

Major:	Electrical Engineering

To my father Bao Tong
who devotes his life
to a better China

ACKNOWLEDGEMENT

The author would like to express his sincere gratitude to his advisor, Professor Anthony D. Robbi, for his guidance, support, kindness, encouragement and friendship throughout the process of producing this thesis.

Thanks to Professor Mengchu Zhou for his invaluable support.

Thanks to Professor David Wang for serving as a member of the thesis committee.

Special thanks to my companion, Renee Chiang, who has been a source of love and indispensible support.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Computer Tools for Petri Net Modeling

The computer tool is essentially a software tool that automates the Petri net creation and simulation processes. A graphical user interface is often used to assist in the building of the model, ease the editing of models, and visually represent the system states. Simulation results are tracked and logged for use in analysis. A computer tool also simplifies the process of modifying the model, and carries out simulations with large numbers of steps to generate reliable performance results when there are stochastic components in the model.

Modeling a system requires that the modeler understand extensively both the system to be modeled and how to appropriately represent the system. The computer tool is designed to assist the expert, and the automation process should only be used *after* the Petri net model has been carefully implemented for the particular system.

1.2 Development of Petri Net Simulation Tool at NJIT

In 1989, Gilani implemented a Graphical Editor for Petri nets for his Masters Thesis at NJIT.[1] Shukla continued the work in 1990 by integrating a simulator for Petri nets with the graphical editor and called the tool PNS (Petri Net Simulator).[2] Both used the C programming language under the UNIX operating system and the window-based SunView environment on the SUN/3 workstation.

The Timed Petri Net Simulator (TPNS) was developed at NJIT in 1991 by Siddiqi, who added to the work done previously by Gilani and Shukla by including the capability of modeling systems with deterministic and stochastic delays.[3] The TPNS collected performance and utilization parameters for use in analysis. Furthermore, the TPNS resolved multiple transitions in conflict, while the PNS could only resolve conflicts between two transitions.

The above work laid a foundation at NJIT for developing a useful Petri Net simulation tool, but more work was necessary to avoid the bugs embedded in the software. The modular programming technique (C programming language based) used by Gilani and built upon by Shukla and Siddiqi presented difficulties in further development and debugging of the existing source code.

As a result, Juneja used an object oriented approach to develop a new basic Petri Net simulation tool.[4] The software was written using C++ programming language under the UNIX operating system and the X-View environment on the SUN/3 workstation. This tool included the capabilities of drawing and editing a Petri net and simulating any discrete event dynamic system. An important feature is the portability achieved by using the XWindow standard, which permits the tool to run on any system supporting it.

1.3 Development of Timed Petri Net Tool (TimedPNT)

The objective of this thesis is to continue work started by Juneja by extending his basic Petri Net tool into a Timed Petri Net Tool. This object-oriented Timed Petri Net Tool (TimedPNT) has the capability of modelling systems with deterministic and stochastic delays, and resolving multiple transition conflicts. The TimedPNT can be used to model concurrent and interlocked systems with time considerations. The OOP technology employed in this tool allows for efficient debugging, and the future addition of features can be accomplished without introducing bugs into the existing code.

CHAPTER 2

AN OVERVIEW OF PETRI NET

The Petri net is a discrete system model. The information gained from this mathematical representation of the system can be used to assist in the design and/or modification of the modeled system. Although initially created to model computer systems, Petri nets can also be used to model other types of systems, such as flexible manufacturing systems or traffic control systems.[5]

The advantage of using a Petri net as opposed to other modeling techniques (with finite difference equations or queueing theory, for example) is the ability of a Petri net to model systems which:

- have concurrent and parallel activities
- have both synchronous and asynchronous activities
- have conflicts for resources
- are event driven
- are distributed

Petri nets' mathematical functions can provide useful qualitative information about systems with the above characteristics. Furthermore, the graphical representation of Petri nets makes the model visually appealing and easy to understand and use. A weakness of Petri net methodology is that it can become too complex to enumerate all its states, a weakness shared with finite state machines.[3]

An example of a system with concurrent, parallel activities is a computer system with many peripheral devices such as printers, modems, disk drives, or audio equipment, which may be operating at the same time under the control of one operating system. An example of a system that has many of the characteristics listed above, and would therefore be a good candidate for modeling by Petri nets, is a manufacturing floor, where different work stations are performing their respective tasks simultaneously, then interacting by passing finished products to another station. [5]

2.1 History of Petri Nets

The basic theoretical concepts of Petri net are from Carl Adam Petri's 1962 doctoral dissertation "Kommunikation mit Automaten," (Communication with Automata), in which Petri formulated the basis for a theory of communication between asynchronous components of a computer system.[5]

A.W. Holt and others of Applied Data Research (ADR) expanded on Petri's work in their Information System Theory Project by developing notation and representation of Petri nets and by showing how Petri nets could be applied to the modeling and analysis of systems of concurrent components.

At the Massachusetts Institute of Technology (M.I.T.), the Computation Structures Group under Professor Jack B. Dennis did much to further the development of Petri net theory. Their research, publications, and conferences held in 1970 and 1975 helped to spread knowledge and application of Petri nets. Interest in Petri nets also spread in Europe. Now, Petri net theory is taught to many electrical and computer engineering students.

2.2 Basic Concepts and Definitions

Petri Net structure, as explained by Peterson, is based on *bag* theory, which is an extension of set theory, where a bag is like a set, but contains elements which may appear multiple times within the bag.[5]

A Petri net structure, C, has four components:

P	-	a set of *places*
T	-	a set of *transitions*
I	-	an *input* function
O	-	an *output* function

The notation is as follows:

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

$P = \{p_1, p_2, p_3, \ldots, p_n\}$, where $n \leq 0$

$T = \{t_1, t_2, t_3, \ldots, t_m\}$, where $m \geq 0$

P and T are disjoint: $P \cap T = 0$.

I maps transitions to bags of places: I: $T \rightarrow P$

O maps transitions to bags of places: O: $T \rightarrow P$

where $I(t_j)$ is the bag of places that are inputs of the transition t_j, and $O(t_j)$ is the bag of places that are outputs of the transition t_j.

A Petri net structure can be described as in the example below:

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

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

$T = \{t_1, t_2, t_3)$

$I(t_1) = \{p_1, p_2\}$ $O(t_1) = \{p_3, p_4\}$

$I(t_2) = \{p_4\}$ $O(t_2) = \{p_2\}$

$I(t_3) = \{p_3\}$ $O(t_3) = \{p_1, p_4\}$

This mathematical description is useful, but the actual structure is difficult to understand without a graphical representation. Figure 2.1 illustrates clearly the structure of the same net that is described by the notation above.

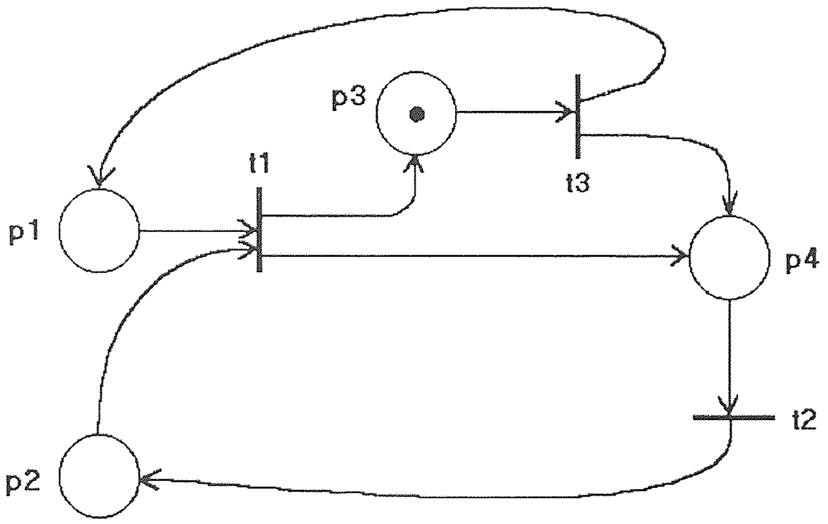


Figure 2.1 Graphical Representation of a Petri Net Structure

2.3 Graphical Representation of Petri Net

The graphical representation of Petri net provides a user-friendly interface to building the model. The elements used are:

○	= a *place*
|	= an *immediate transition* (no firing time)
▯	= a *timed transition*
▮	= a *timed transition holding token(s)*
●	= a *token*
↗	= an *arc* illustrating input and output mapping
⑤	= a *number of tokens* associated with a place

2.4 Simulation of the Dynamic Behavior of Systems

The state of a Petri net model is defined by its *marking*, μ. A marking assigns to each place a non-negative integer representing the number of tokens in that place. μ is a vector of dimension n, where n is the number of places in the model. $\mu(i)$ is the number of tokens in p_i.

The dynamic behavior of a system is simulated by the "firing" of transitions. The changing states are described by the changing of markings.

A transition is said to be *enabled* if each of its input places contains at least as many tokens as arcs from that place to the transition.

When an enabled transition is *fired*, one token per arc is removed from each input place, and one token per arc is added to each output place.

To take the above Petri net as an example (see Figure 2.1), transition t_1 is <u>enabled</u> when p_1 and p_2 both contain tokens. Upon <u>firing</u> t_1, one token will be removed from p_1 and one token from p_2, while one token will be placed in each of the places p_3 and p_4. In Figure 2.1 as shown, t_3 is enabled.

2.5 Additional Petri Net Concepts

A Petri net is considered *live* if at least one transition is enabled, which ensures that the system is still dynamic (has a new state). If a state is reached in which no transition is enabled, the system has reached a *deadlock*, since no changes can occur. These are useful descriptions of systems since most systems need to avoid deadlock situations.

Concurrency and *conflict* are used to describe a relationship between two events. Two transitions are in conflict if either can fire by itself, but not both because they share one or more input places. They are said to be in concurrence if the two transitions may fire singly or simultaneously, in any order without competing for tokens.

2.6 Modeling with Petri Nets

The proper application of Petri nets is a field of study in itself. Inappropriately representing a system would yield a model with meaningless results, no matter how powerful the representation. It is important to understand the characteristics and meaning of the Petri net model before applying it.

In the application of Petri net models, transition firings basically represent events, while places represent states. The input place marking is the precondition, which if met (represented by the existence of sufficient tokens), enables the transition. The output place markings are the postconditions, which are met after an event occurs (transition is fired).

Petri nets have been used to model a diversity of systems since the basic elements are general. Application developments of Petri nets have yielded the creation of a variety of subclasses of Petri nets.

One type of Petri net is the *state machine*, where every transition has exactly one in-coming arc and one out-going arc. This type of model only allows for the representation of decisions, but does not represent parallel activities. Another subclass is the *marked* graph, where every place has exactly one in-coming arc and one out-going arc. This type of Petri net represents concurrent parallel activities, but not decisions (resolution of conflicts).

Different applications of Petri nets have their specific uses of these concepts. In a manufacturing example, transitions may represent assembly, input place markings represent component parts, and output place markings represent the finished product. Tokens would represent the availability of parts or of a machine.

In a data computation model, the transition is the computation, the input place markings are the input data, and the output places are the output data. Tokens represent both the availability and value of data variables.

Petri net modeling is especially useful in modeling communication protocols for their liveness and performance efficiency. The marking would represent the state of readiness (e.g. ACK recv'd, READY received), and message or packet queues. Transitions would represent events (e.g. Send Message, Receive Message).

2.7 Timed Petri Nets

A "pure" Petri net does not include the time factor, so transitions are fired immediately, as soon as they are enabled. But, in reality, operations often need a finite amount of time to be completed, and throughputs, cycle times, and delays are important properties of systems which can not be measured without representing time in the model.[7]

Several different types of <u>Timed Petri Nets Simulators</u> (TPNS) have been developed.

- A *deterministic* delay (fixed firing time) can be associated with each transition. In 1973, Ramachandani presented this concept and applied it to model the performance of computer systems.[8] In his model, after the transition is enabled, there is a fixed delay, before the instantaneous removal of tokens from the input places and addition of tokens at the output places. Sifakis applied the same concept to a delay associated with each place, so that a token is not available for firing for a fixed time after it arrives at the place. This was later determined to be equivalent to Ramachandani's concept.[9]

- A *variable* time within a fixed range can be assigned to each transition. Merlin's Ph.D. dissertation in 1974 developed this concept which delayed the firing of a transition for a time, within a fixed range, from the moment it was enabled.[10]

- Another deterministic delay concept used by Zuberek also assigned a fixed time to transitions. The difference from Ramachandani's is that the tokens are removed from the input tokens immediately when the transition is enabled, but are *withheld* for the fixed period of time before tokens are added to the output places.[11] This implies that a decision to fire a timed transition is irreversible, i.e. the tokens involved cannot be committed elsewhere while waiting for the actual firing.

2.8 Stochastic Petri Nets

The Stochastic Petri Net (SPN) does not assign a fixed time to the firing of transitions, but uses a stochastic delay, usually making the process equivalent to a Markov process. There are a variety of types of Stochastic Petri nets as well.

- The common *SPN*, as described by Molloy, associates an *exponentially distributed random* variable to the firing time of each transition.[12]

- *Other distributions* of time for firing are used in other models.[13]

- *Generalized SPN (GSPN)* includes two kinds of transitions: the conventional zero firing time transition (drawn as a thin bar), as well as the transition with exponentially distributed time delay (drawn as a box). [14]

- Other variations randomly select which tokens to remove from the input places and which tokens to add to the output places. [15]

- Time can also be assigned to how long a token resides in a *place*.

- Combinations and extensions of the above have also been used, including an extension of the GSPN, where a timed transition can be either a deterministic or exponentially distributed random variable.

The resolution of conflicts in each of the above cases can be associated with the time or independent of it. *Preselection* is the method that chooses which transition to fire independently of the firing time. The selection of which transition to fire can also be considered in *parallel* to the firing time, so the transition with the minimum firing time is chosen.

CHAPTER 3

A TIMED PETRI NET SIMULATION TOOL

3.1 TimedPNT Concepts

Before a complete description of the design of the TimedPNT is given, the following concepts should be clarified.

3.1.1 Graphical Objects

NET:

A *Net* is a timed Petri net model of a particular system, and is named by the user. Its graphical representation, properties, and system state are stored in a file by that name plus an extension of "**.net**".

TRANSITION:

- *Immediate Transitions*, graphically represented by a solid thin bar, are the classical transitions that fire without delay.
- *Timed Transitions*, graphically represented by a hollow bar, is a transition that includes the property of delay, which may be deterministic or stochastic.
- *Deterministic Delay* is a fixed delay, designated by the user and associated with a timed transition.

- *Stochastic Delay* can be chosen for a timed transition instead of a deterministic delay. This property is a random delay with an average number which is designated by the user. (See 3.2.1)
- *Priority* of each transition may be designated by the user in order to determine firing sequence in the case of conflicts between transitions.
- *Transition Numbers* are associated with each transition of a particular net for identity purposes.

PLACE, ARC and TOKEN:

- *Places, Arcs* and *Tokens* are presented as per convention in traditional Petri Net.

Transitions, Places, Arcs and Tokens are basic objects that can be drawn by the user to form the starting state (marking) of a net.

3.1.2 Property Objects

TAG:

Tags are pop-up windows used to display or modify the characteristics and parameters of basic objects. In TimedPNT, there are four types of Tags: Immediate Transition Tag, Timed Transition Tag, Place Tag and Arc Tag.

- **Immediate Transition Tag** includes a *Comment* item, which allows the user to record a description of this immediate transition, a *Number* item, which displays the transition number in that net for identity purposes, a *Priority* item, which permits the user to define a level of priority for the transition when it is in conflict with other transitions, a *State* item, which displays the transition state (READY or NOT READY) when the Tag is called, and a *Type* item, which shows the transition type is IMMEDIATE.

- **Timed Transition Tag** not only includes Comment item, Number item, Priority item, State item and Type item, but also contain a *Delay* item, from which the user can specify the delay type (DETERMINISTIC or STOCHASTIC), and a *Delay/Avg.Delay* item, which allows the user to specify a delay time (in the case of deterministic delay) or an average delay time (for stochastic delay).

- **Place Tag** contains a *Comment* item, a *Number* item, and a *Stop Marking* item, which designates an end condition of a number of tokens (See 3.2.5 Simulation End Conditions). In addition, there is a *Tokens* item, which shows how many tokens are currently present in this place.

- **Arc Tag** contains a *Comment* item, a *Weight* item, which lets an arc represent multiple identical arcs, a *No. of Segments* item, which gives information on how many segments this arc is broken up into, and a *Type* item which shows if the arc is an INPUT arc or an OUTPUT arc.

VERIFY:

Verify is a procedure which can be called upon by the user any time after the drawing of a net is completed. This procedure gives the user a tabular structural description of the current net. It is useful for ensuring the graphical drawing is correctly interpreted by the tool for simulations.

LOG:

Log automatically creates a file containing a complete set of marking information during simulation (unless the Log Mode is OFF, see 5.7.1). Other details of the simulation, such as transitions enabled and conflict resolution are also contained in this file.

REPORT:

"Report" is a file automatically generated after simulation. This file contains utilization parameters of all places and transitions. (See 3.2.8)

3.2 TimedPNT Characteristics and Policies

The TimedPNT inherits basic behaviors and characteristics of the conventional Petri Net. As an extended Petri Net, some important characteristics and policy decisions have been made during the development of the TimedPNT. The characteristics and policies which differ from a conventional Petri Net are highlighted below.

3.2.1 Transition Delay

Besides the basic net objects (such as Places, Arcs, Transitions, and Tokens) in conventional Petri Net, the TimedPNT contains two different types of transitions, *immediate* and *timed*.

For *immediate* transitions, there is no associated delay. For *timed* transitions, the user must specify the delay type as *deterministic* delay or *stochastic* delay. For deterministic delays, a fixed delay time must be chosen. In the case of the stochastic delay, an average delay time must be specified by the user; during simulation, an actual delay is calculated based on this average, each time the transition is enabled.

In this process, for simulation consistency, a regenerative random number is generated each time as the delay random variable.

The cumulative distribution function is:

$$F_X(x) = P\,r\,[X \leq x] = 1\text{-}e^{-\lambda x}$$

where x is the average firing rate

The probability density function is:

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

The average delay is given by:

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

3.2.2 Firing a Transition

Firing an Immediate Transition -

The following steps occur simultaneously in the firing of an immediate transition:

- Remove tokens from all input places of enabled transition.
- Place tokens in all output places.
- Update the associated place or transition data structures.

Firing a Timed Transition -

When a timed transition is fired, the firing procedure occurs in three stages (all bulleted items occur simultaneously):

1. Preparation
   - Remove tokens from the transition's input places
   - Set the transition "ticking," which means graphically the transition becomes a thick solid bar (representing token being held for a period of time)
   - Set transition state to NOT READY.
2. Delay
   - Continue to delay until transition ticking time is up.

3. Complete firing
   - Transition releases tokens and becomes a hollow bar.
   - Place tokens in appropriate output place(s).

3.2.3 Preselection

In TimedPNT, *preselection* has been chosen as the conflict handling policy. Preselection means that the selection of a transition to fire in a conflict situation occurs **independently** of the transition firing time (delay) or of transition type. Furthermore, in case of conflict, all transitions with the same priority are chosen randomly based on an equal opportunity rule, and only one transition can be chosen.

Figures 3-1 and 3-2 show two examples of conflict resolution by the preselection scheme, illustrating that neither the transition type nor the delay time influences the choice of transition. The modeler can assign priority levels to force a particular transition to fire in conflict situations.

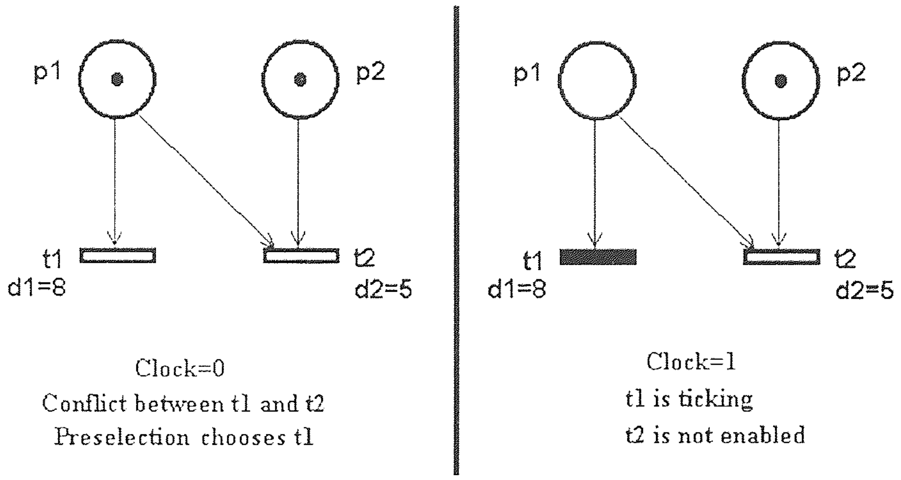


Figure 3.1 Preselection Chooses a Transition Regardless of Delay Time

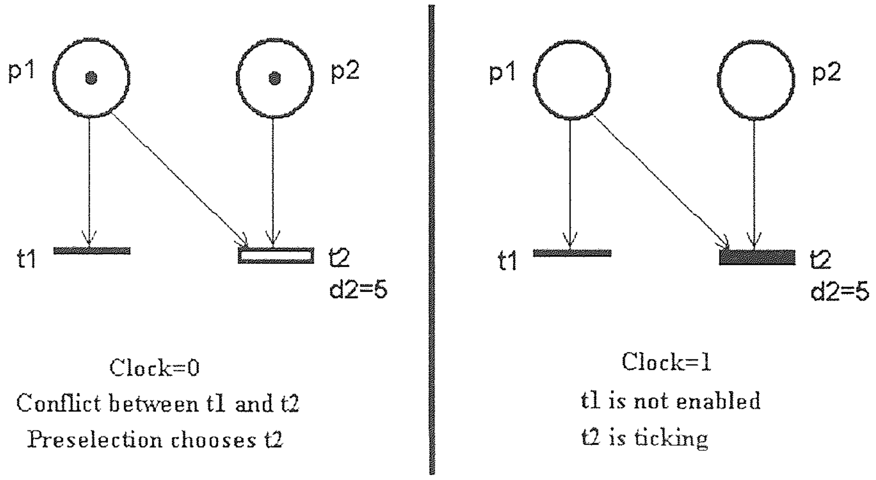


Figure 3.2 Preselection Chooses a Transition Regardless of Transition Type

3.2.4 Time in TimedPNT Simulation

A *system clock* has been built into the TimedPNT. This system clock is only concerned with the time during which tokens remain in timed transitions. In other words, an assumption has been made that token removals (from input places to transitions) and token deposits (to output places) occur immediately (i.e. system clock is not advanced). For the sake of simulation efficiency, the system clock is advanced by the minimum token remaining time of timed transitions, instead of by a single unit of time. No computing time is ever involved.

3.2.5 Simulation End Conditions

A process of simulation can be terminated by *End Conditions*. There are two types of end conditions:

1. System reaches a state of *deadlock*.

2. One of the following *user pre-defined* end conditions is reached:
    a. A desired number of run steps have been finished (see STEP and RUN).
    b. A minimum simulated clock time has been reached.
    c. A place marking has been reached (specified by user in the Stop Marking item in the associated place tags):
        1. In at least one **place**, or
        2. For *all places* specified in the **net**.

Before simulation begins, the user can specify more than one end condition (a, b, and/or c). However, if using place markings (c), the user must choose one of the two types, **place** or **net**. At run time, the simulation will be terminated by the following end conditions:

1. By place marking (c), if Stop Marking item in any place tag has been assigned a value by the user greater than -1 (i.e. default = -1 means not specified).
2. By clock time (default = 0 means not specified), if no place marking has been assigned.
3. By step number (default = 100), if neither of the above has been assigned, or if the step number is reached before the place marking (as a safeguard to prevent the program from running indefinitely).

3.2.6 Step

In TimedPNT, *net status* is the classification of a particular net. If a net consists of immediate transitions only, then the net status is defined as *IM*. If a net consists of both immediate and timed transitions, the net status is *MIX*. Similarly, if a net consists of timed transitions only, the net status is *TI*.

The simulation process can be carried out in either *Step* mode or *Run* mode. In TimedPNT, a STEP is defined in three ways depending on net status:

1. <u>Net Status = IM</u> (Immediate Transitions only)

   One STEP is:

   1. Form a preliminary list of ready (enabled) transitions.

2. If there are any enabled transitions, continue;

   otherwise, prompt Deadlock message and terminate STEP.

3. Resolve conflicts, if any, and form a final list of ready transitions.

4. Fire all transitions on the final list. (See 3.2.2 Firing an Immediate Transitions)

5. Visually update the net.

6. Record the new net marking.

2. <u>Net Status = TI</u> (Timed Transitions only)

One STEP is:

1. Form a preliminary list of ready (enabled) transitions.

2. If there are any enabled or ticking transitions, continue;

   otherwise, prompt Deadlock message and terminate STEP.

3. Resolve conflicts, if any, and form a final list of ready transitions.

4. Remove tokens from input places.

5. Set those transitions to "ticking" and form a list of remaining ticking time of each ticking transition.

6. Sort the list by remaining time with the transition with the smallest remaining time on top.

7. Visually update the net to show transitions ticking (filled in box) and token removals.

8. Fire the timed transition(s) with the minimum remaining time. (See 3.2.2 Firing a Timed Transition)

9. Advance system clock by the minimum remaining time of step 8 above.

10. Visually update the net to show tokens deposited into output places.

11. Record the new net marking.

3. Net Status = MIX (Immediate and Timed Transitions)

One STEP is:

1. Form a preliminary list of ready (enabled) transitions.

2. If there are any enabled transitions, continue;

   otherwise, prompt Deadlock message and terminate STEP.

3. Resolve conflicts, if any, and form a final list of ready transitions.

4. Check if there are any immediate transitions in the final list. If so, fire all of them and terminate this STEP;

   otherwise, continue.

5. After exhausting all enabled immediate transitions, proceed with enabled timed transitions as items 5 through 12 of Net Status = TI above.

The system clock is not advanced in a STEP where Immediate transitions are fired. Repeated STEPS may occur in order to exhaust all enabled immediate transitions, while the system clock remains unchanged. Only after all immediate transitions have been exhausted, and enabled timed transitions are fired will the system clock be advanced.

3.2.7 Run

RUN is another simulation mode. In Run mode, STEP is called repeatedly according to the net status. A Run procedure will terminate if either:

1. A user specified end condition, such as how many steps the user wants to take, has been reached (see 3.2.5).
2. A deadlock occurs.

For simulation efficiency, the net is not visually updated in each step, until an end condition is reached.

3.2.8 Utilization

Net utilization parameters are calculated and recorded in a Utilization Report.

Transition Utilization

All transitions are tabulated according to the following parameters:

- *Number of times fired*

  If a transition is ticking when simulation ends, it is not considered to have fired.

- *% Busy*

  For immediate transitions:

  % Busy = (# of times fired)/(# of total steps)

For timed transitions:

% Busy = (total clock time holding tokens)/(total run time)

Place Utilization

All places are tabulated according to the following parameters:

- *Total tokens entered*

  Initial markings are included.

- *Duration occupied*

  A place is either vacant or occupied. It is considered occupied if there is at least one token in the place; or if at least one of the timed transitions for which this place is an input place, is ticking.

CHAPTER 4

DESIGN OF THE TimedPNT

4.1 XView^{TM}

Siddiqi's Timed Petri Net Simulator (TPNS) was a SunView^{TM} based application, while the current TimedPNT is an XView^{TM} based application using XView^{TM} Toolkit.[16]

The XView^{TM} is an object-oriented user interface toolkit designed for XWindow System^{TM} Version 11 (X11). As currently configured on workstations, the XView^{TM} toolkit runs on the X side of the X11/NeWS^{TM} Server.

4.1.1 XView^{TM} System Architecture

The XView^{TM} system architecture differs significantly from the SunView^{TM} architecture. The XView^{TM} toolkit is implemented for a server-based window system (X11), while the SunView^{TM} toolkit was implemented for a kernel-based window system (SunWindows^{TM}).

The basic difference is that kernel-based systems are usually hardware and operating system specific; on the contrary, server-based systems use a protocol that is independent of hardware and operating systems. Server-based window systems can run on one machine and display their output in windows on another machine anywhere on the network, regardless of machine architecture, operating systems, display resolutions, and color capabilities.

Figure 4-1 illustrates how a remote workstation could run the TimedPNT over a network, using the XView Toolkit. This is quite an improvement over the kernel-based system.

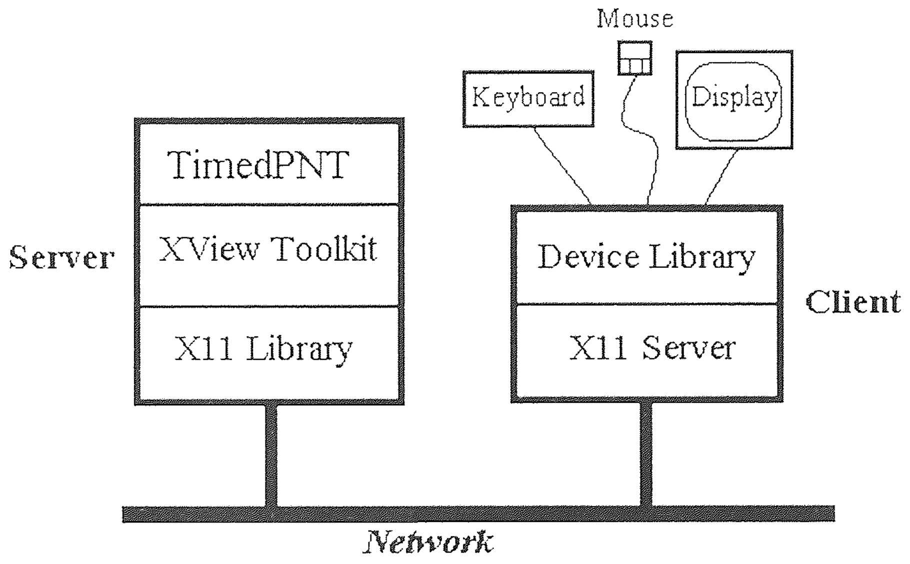


Figure 4.1 TimedPNT Running on a Network

4.1.2 TimedPNT Architecture as an XView^{TM} Application

The XView^{TM} Toolkit provides a framework for assembling pre-built user interface components with application-specific code. In the XView^{TM} environment, variable-length, attribute-value lists are used to specify various TimedPNT objects to be created, such as windows, menus, and scrollbars. Since the usual behavior for each object is already defined, only deviations from the default behavior needs to be specified. After specifying objects, call-back procedures for each object are defined. From those procedures, the toolkit calls to notify the application of events or user actions. Finally, the TimedPNT's source code is compiled with the XView^{TM} library.

4.2 The Human Interface of the TimedPNT

The TimedPNT is an interactive graphics-based application. Its human interface is formed by various XView^{TM} Visual Objects.

4.2.1 TimedPNT Window Objects

Window objects include frames and subwindows. Frames contain non-overlapping subwindows within their borders. The following subwindows are used in TimedPNT:

Canvas Subwindows: The XView^{TM} Toolkit provides application programmers with a drawing surface, called the *paint window*, on which the net can be drawn in TimedPNT. The paint window is clipped to another window, the *view window*, so only the part of the paint window underneath the view window shows through. The view window appears, together with scrollbars, in the canvas subwindow. As a result, a net can be drawn on an area larger than the visible window where the drawing appears.

Panels and Menus: The *panels* or *menus* are used as notifiers to distribute events to the individual panel and menu item objects. (See Panel item)

Text Subwindows and Scrollbars: The *text subwindows* are only used to display information, such as log and verify files, in TimedPNT. (However, in the XView^{TM} environment, the text subwindow is a full-featured text editor.) The *scrollbars* are responsible for keeping the elevator and proportional indicator up to date, but it is the responsibility of the window attached to the scrollbar to update the window's contents.

4.2.2 Other Visual Objects

Other visual objects used in TimedPNT are Panel item, Notice, Cursors and Icons.

Panel items are objects in a control panel that allow interaction between the user and the TimedPNT. Panel items can be moved, displayed and undisplayed. There are several pre-defined types of items, including buttons, choice items, etc.

Notice is a box on the screen that informs the user of some condition that requires attention, such as when deadlock is reached, or when an erroneous action has been taken by the user. It usually has one or more buttons which the user can press to confirm, deny, or continue.

Cursor is a mark on the screen representing the user's point of attention. It is controlled by the mouse.

Icon is a small window representing the TimedPNT when its frame is in a closed (idle) state.

4.2.3 Functionalities of the TimedPNT interface

The TimedPNT interface is designed to perform the following functions:

1. Drawing and editing a net.
2. Saving and loading a net.
3. Carrying out simulations either in STEP or RUN mode.
4. Viewing simulation results.

4.3 Controller Object

The *controller* is the most important object in a simulation. The user unknowingly calls it to get the net information and to do the necessary computation during a simulation.

Some of the important procedures and functions are:

- **Form_List**: It forms a list of enabled transitions in the net at the time it is called. This list is known as the rdy_trans list (ready transitions list).
- **Resolve_Conflict**: It resolves conflicts based on an equal opportunity rule when transitions in conflict have the same priority, and forms a final list of ready transitions, which is also called rdy_list.

- **Sort_List**: It sorts the ticking transition list, list_rm, according to each transition's remaining time, in ascending order.
- **Check_Condition**: It is responsible for detecting if an end condition is reached.
- **Is_Conflict**: It detects when conflicts occur.
- **Execute_Im**: It calls the procedure to fire immediate transitions.
- **Execute_Ti_1 and Execute_Ti_2**: They together fire timed transitions.

Some important variables are shown in the Table 4-1.

Table 4.1 Controller Variables

controller	
Variable	**Description**
clock	system clock
flg	flag for immediate transition ready
list_rdy	array of ready (enabled) transitions
list_rm	array of tick and ready transitions
mode	log off/on
net_handle	pointer to the current net
rdy_trans	number of ready transitions
remain_no	number of transitions ticking
run	run count during simulation
size_list	size of array
status	current net status (IM/MIX/TI)
steps	simulation mode

The following is the data structure of *controller*:

```
#include "net.h"
# include <rand48.h>
# include <math.h>

const int OFF = 1;
const int ON = 0;

// net status constants
const int IM = 1;
const int MIX = 2;
const int TI = 3;

class controller {
net *net_handle; // pointer to the current net
int size_list; //size of array;
transition **list_rdy; // array of ready transitions
transition **list_rm; // array of ticking + ready transitions
public:
controller(char *);
int rdy_trans; // no of rdy transitions
int steps; // simulation mode
int stop_watch; // user specified time to end simulation
int stop_way; // stop simulation by PLACE/NET marking
int mode; // log off/on
int run; // run count during simulation
int status; //current net (im only/mixed/ti only)
int clock; // system clock
int remain_no; // number of trans in ticking
int flg; // flag for immediate transition ready

void Stretch_Array(void);
void Destroy_All(void);
net *Get_Net_Handle(void);
void Set_Net_Handle(net *);
void Form_List(void);
void Resolve_Conflict(void);
void Sort_List(void);
int Check_Condition(void);
int Is_Conflict(transition *, transition *);
void Execute_Im(void);
void Execute_Ti_1(void);
void Execute_Ti_2(void);
};
```

4.4 Net Object

Net object is the manager of a net and its visual objects, such as places, transitions, input and output arcs.

In the TimedPNT, different net models are managed by the *Net* object. It saves a net with a user-given filename, and performs load operations as the user requests. In a particular net, the *Net* object identifies each transition, place and arc, and manages the addition or deletion of any of those objects.

The data structure of the *net* object is:

```
#include "transition.h"

const int PLACE = 1;
const int TRANSITION = 2;
const int ERROR_A = -2;
const int ERROR_B = -3;
const int ERROR_C = -4;


class net {
public:
int place_no,transition_no,input_no,output_no;
int place_num, trans_num;
char file_name[100];
int size_place,size_transition;
transition **trans_list;
place **place_list;

net();
net(char *);

int Add_Place(int,int);
int Add_Transition(char,char,int,int,int,int,int);
int Add_Arc(int,int,int,seg_array*);
int Remove_Place(int);
int Remove_Transition(int);
int Remove_Arc(int,int,int);

char *Get_File(void);
int Set_File(char *);
void Stretch_Array(int);

place *Get_Place(int,int);
place *Get_User_Pl_No(int);

int Get_Pl_No(int,int);
transition *Get_Transition(int,int);
int Get_Trans_No(int,int);
arc *Get_Arc(int,int,int);
};
```

4.5 Basic Object

The *basic* object manages the important attributes of both transitions and places. These attributes are: ID number of the transition or place, its coordinates on the canvas, and comments documented by the user.

The following is the data structure of the *basic* object:

```
#include <iostream.h>
#include <stdio.h>
#include <string.h>

class basic_object {
// attribute for the basic object
int no; // the number assigned to a place/transition
int loc_x,loc_y; // the x,y coordinates of the object on the canvas
char comment[50];

// member functions are public
public:

//constructor and destructor
basic_object();
~basic_object();

// member functions are public
int Get_No(void);
void Set_No(int);

int Getloc_x(void);
void Setloc_x(int);

int Getloc_y(void);
void Setloc_y(int);

char *Get_Comment();
void Set_Comment(char *);
};
```

4.6 Transition Object

The *transition* object is a class derived from the *basic* object. In C++ programming, a derived class may inherit the attributes and functions of the base class. Including those attributes contained in the *basic* object, the *transition* object contains a complete set of information of each transition. The additional information includes:

- transition type (immediate or timed)
- state (ready to fire, or not ready)
- orientation (vertical or horizontal)
- priority

- delay type (deterministic or stochastic)
- delay time (fixed delay for deterministic or average delay for stochastic)
- ticking remaining time
- ticking status (yes or no)

The *transition* object also determines if a transition is enabled and performs transition data structure update when it is fired.

The data structure of *transition* object is as follows:

```
#include "arc.h"
#include <rand48.h>
#include <math.h>

const int READY = 1;
const int NOT_READY = 0;
const int YES = 1;
const int NO = 0;
const int FIXED=0;
const int STOCHASTIC=1;
const int NODELAY=2;

class transition : public basic_object {
// attributes for class transition

int state; // state of transition  ready/not ready
int priority; // priority of transition
int delaytype; //fixed/stochastic
int fixedelay; //fixed delay (d) or avg delay (s) of trans
int stochasticdelay; // calculated stochastic delay of trans
int ticking; // status of ticking  yes/no
int tick_remain; // tick remaining time
int total_fire; // total fire times
int total_delay_time; // total delay time for net utilization calculation
char orientation; // horizontal/vertical
char ttype; //immediate/timed

public:
int input_no, output_no; // number of input and output arc
int size_input,size_output; // size of arrays of pointers to input&output
arcs
arc **input;// arrays of pointers to input & output arcs
arc **output;

transition();
transition(char,char,int,int,int,int,int,int);  //constructor  for  trans
class
char Get_Orientation(void); // returns the orientation
void Set_Orientation(char); // sets the orientation
char Get_its_Type(void); // returns the transition type (immediate/timed)
void Set_its_Type(char); // sets the type of transition (im/ti)
int Get_Delay_Type(void); // return the type of delay(s/f) if it's timed
void Set_Delay_Type(int); //sets delay type
int Get_Ticking(void); // return ticking status (YES/NO)
void Set_Ticking(int); //sets ticking status (YES/NO)
int Get_Fixed_Delay(void); //returns fixed delay
void Set_Fixed_Delay(int); // sets fixed delay
int Get_Sto_Delay(void); // calculate and return stochastic delay of trans

int Get_Remaining(void); //returns fixed delay
void Set_Remaining(int, int); // sets fixed delay

int Get_Total_Fire(void); //returns total fired times
void Set_Total_Fire(void); // sets total fired times
void Reset_Total_Fire(void); // reset total fired times

int Get_Total_Time(void); //returns total delay time
void Set_Total_Time(int); // sets total delay time
void Reset_Total_Time(void); // resets total delay time

int Get_State(void); // returns the state of transtion
void Set_State(int); // sets the state of transition
int Get_Priority(void); // returns the priority of transition
void Set_Priority(int); // sets the priority of transition
void Stretch_Array(int); // stretches the input/output array
int Add_Arc(int,int,arc *); // adds an arc to the input/output array
void Remove_Arc(int,int); // removes an arc from the input/output array
int Is_Enabled(void); // returns whether the transition is enabled or not
void Fire_From(void); // fires the transition from its input places
void Fire_To(void); // fires the transition to its output places
};
```

4.7 Place Object

The place object is also a derived class of the *basic* object. In addition to information in the *basic* object, the *place* object has the token information of a place, such as the number of tokens currently being held, and the breakpoint attribute, which is the execution end condition specified by the user. The *place* object performs operations to add or remove tokens to/from a place.

Its data structure is:

```
#include "basic_object.h"

const int SUCCESS = 0;
const int FAIL = -1;
```

```
class place : public basic_object {
// attributes of class place
int no_of_tokens; // no of tokens in a place
int total_tok_no; // total no of tokens entered in a place
int vacant_status; // YES (0) / NO (1)
int vacant_time; // total vacant time of a place
int breakpt; // user defined token no to end simulation

public:
place();
place(int,int,int); // constructor for class place

void Add_Tokens(int);
int Remove_Tokens(int);
int Get_Tokens(void);
void Set_Tokens(int);
int Get_Breakpt(void);
void Set_Breakpt(int);
int Get_Total_Tokno(void);
void Set_Total_Tokno(int);
void Reset_Total_Tokno(void);
int Get_Vacant(void);
void Set_Vacant(int);
void Reset_Vacant(void);
int Is_Break(void);
int Get_V_Status(void);
void Set_V_Status(int);
};
```

4.8 Arc and Related Objects

4.8.1 Arc Object

The attributes of *arc* object are: type of arc (input, output or inhibit), weight (the multipicity of arcs between a place and transitions of a given type) and information about places to which arcs are connected.

The data structures of *arc* object is:

```
#include "place.h"
#include "seg_array.h"
const int INHIBIT = 2;
const int INPUT = 1;
const int OUTPUT = 0;

class arc {
// attribute of class arc
int type; // type of arc input/output
int place_no; // no of associated place
int wt; // weight of arc
place *place_handle; // pointer to the associated place
seg_array *seg_handle;
char comment[50];
public:
arc();
arc(int,int,place *,seg_array *); // constructor with intialization

char *Get_Comment();
void Set_Comment(char *);
int Get_Type(void); // returns type of arc
void Set_Type(int); // sets type of arc
int Get_Place(void); // returns associated place no
void Set_Place(int,place *); // sets place no
int Get_Wt(void); //returns wt of arc
void Set_Wt(int); // sets wt of arc
place *Get_Handle(void); // returns the place handle
seg_array *Get_Seg_Array(void); // returns pointer to segment array
void Set_Seg_Array(seg_array *); // sets the current segment array
};
```

4.8.2 Segment Array Object

An arc is comprised of an array of straight lines. These lines are called segments. The *segment array* object is the object that manages those segments.

The data structure is:

```
#include "segment.h"
#include <stdio.h>

class seg_array {

public:
int size_array;
int no_of_segments;
segment **s_array; // array of pointers to segment
seg_array();

void Stretch_Array(void);
void Add_Segment(int, int, int, int);
void Remove_Segment(void);
};
```

4.8.3 Segment Object

The *segment* object contains the coordinates of the starting and ending points of segments.

The data structure is:

```
class segment {
public:
```

```
int x1;
int y1;
int x2;
int y2;

segment();
segment(int, int, int, int); // constructor for class segment

};
```

CHAPTER 5

TimedPNT USER'S MANUAL

5.1 Preparation for Running TimedPNT

Before running TimedPNT, the user must first:

- have thought thoroughly about how to accurately represent the system to be modeled
- prepared a draft of the net to be run (recommended)
- have TimedPNT properly installed
- have OpenWindows^{TM} running
- if running TimedPNT locally, type at a command tool window prompt in the TimedPNT working directory:

  ```
  % tpnt
  ```

- if running TimedPNT on a network:

  - "rlogin" or "telnet" to the remote host from the UNIX command prompt
  - after establishing a connection with the host, type at the host command prompt:

    ```
    telnet> setenv DISPLAY local_name:0
    ```

  - return to local command prompt (in a different window), and type:

    ```
    % xhost +
    ```

  - returning again to the host command prompt, type:

    ```
    telnet> tpnt
    ```

5.2 The Working Environment of The TimedPNT

OpenWindows^{TM} is the user environment of the TimedPNT. It provides XView^{TM} window pane management service, with a consistent screen layout. To a TimedPNT end user, it offers an intuitive working environment. In order to run the TimedPNT, a thorough working knowledge of OpenWindows^{TM} is recommended. (See OpenWindows^{TM} User's Guide)

Using The Mouse

In the operation of the TimedPNT, the mouse is the most important device for the user to control the program operation.

In this manual, the following terms are used to describe actions performed with the mouse.

- *Press* a mouse button and hold it.
- *Release* a mouse button to initiate an action.
- *Click* a mouse button by pressing and releasing it before moving the pointer.
- *Double-click* a mouse button by clicking twice quickly without moving the pointer.
- *Move* the pointer by sliding the mouse with no buttons pressed.
- *Drag* the pointer by sliding the mouse with one button pressed.

In the TimedPNT, the Left Mouse Button (LMB), called SELECT, is always used to select or manipulate controls in the operation. The Right Mouse Button (RMB), also called MENU, is used consistently to display and choose from pull-down menus.

5.3 Starting The TimedPNT

After typing "tpnt", the TimedPNT base window will show on the OpenWindow^{TM} workspace. The title bar of the tool shows: "**TimedPNT:**", followed by the name of the current working directory.

The basic controls for the TimedPNT are displayed in a control area, immediately below the title, which consists of five standard menu buttons, **File**, **Draw**, **Edit**, **Simulate** and **Utilities**; and a text field following the prompt "**File:**" which is filled in with "no_name" as a default, but should be changed by the user. In Figure 5-2, the TimedPNT window is shown with all of its menu sub-windows pinned under their respective menu buttons. All items will be explained in detail in the following text.

The large window under the control area is the TimedPNT canvas, on which a paint window is clipped to a view window (see 3.2.1). The paint window, which is where a net is drawn, is considerably larger than the view window, which reveals only a portion of the paint window at any time. Two scrollbars are used to move the paint window around the areas of the view window.

5.4 Drawing and Editing a Net

Before running a net simulation, the net has to be drawn or loaded onto the canvas. In the process of drawing, the user may also need to edit. In TimedPNT, all net drawing and editing features are incorporated in the pull down menus and invoked by the menu buttons, **Draw** and **Edit**.

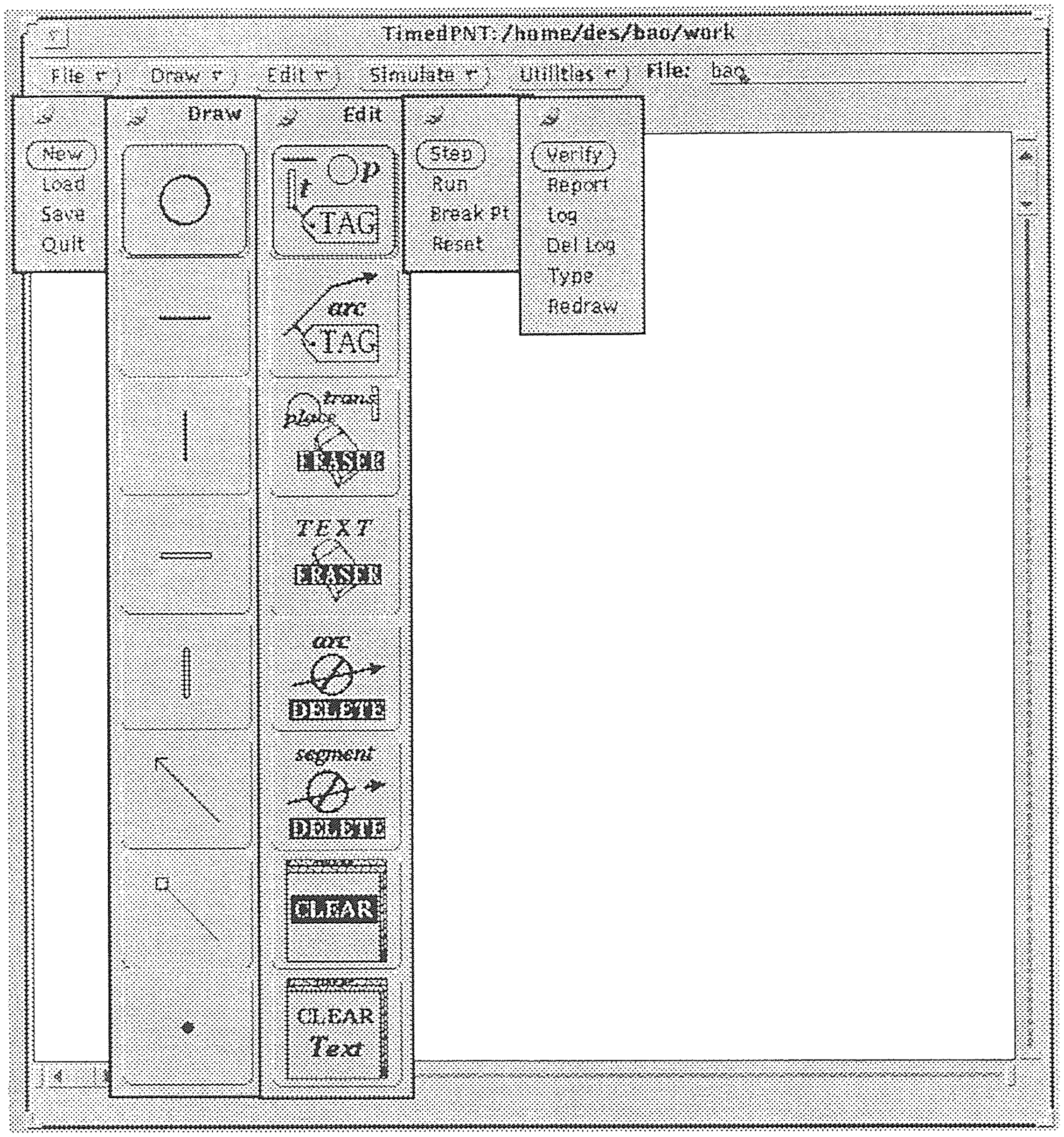


Figure 5.1 The TimedPNT Window and Menu Sub-windows

5.4.1 Using the Menus

To display the pull down menus, the user needs to **point to the menu button and press the RMB (MENU)**. This operation is consistent in the OpenWindows™ application environment. For the sake of simplicity, in the proceeding text, this operation will be referred to as "display menu". Similarly, "choose menu item" will be used to describe the actions shown in Table 5.1. There are actually four different ways to "choose menu item", as shown in the table, but they perform the same function.

When the user chooses a pull down window item, the window is automatically dismissed after one item selection. But when building a net, the user often needs to repeatedly choose items from different windows. For convenience, it is suggested that all TimedPNT sub-windows be pinned and moved to a different location on the pane. Figure 5-4 shows the sub-windows for all the menus pinned outside the canvas area. By pinning and relocating both the Draw and Edit sub-windows, the user has an open canvas area in which to build a net. Also, the menu button does not need to be used repeatedly to select the item: the item buttons can be selected directly.

5.4.2 Drawing A Net

Before drawing a net, it is necessary to first name the net. The user needs to move the pointer to the text field after the prompt "**File:**" in the control area, click the LMB (SELECT) and then change the default "no_name" by typing in an appropriate name. Then display File Menu, and choose New item. This creates a new data structure under the user specified name.

Table 5.1 Different Methods of Choosing Menu Item

	Function	Mouse	Description of Action
1	choose default menu item	click LMB	move pointer over the menu button and click the LMB (SELECT) to choose the default item under this menu (i.e. the first item button)
2	display menu	press RMB	move pointer over the menu button and press the RMB (MENU)
	choose menu item	drag - release	continue holding down RMB, drag the pointer over the desired item in the menu, then release the RMB.
3	display menu	click RMB	move pointer over the menu button and click the RMB (MENU)
	choose menu item	move - click LMB	move pointer over desired item in the menu and click LMB (SELECT)
4	display menu	click RMB	move pointer over the menu button and click the RMB (MENU)
	pin sub-window	click LMB - press LMB - drag - release	pin sub-window; drag window to desired area in pane. ***
	choose menu item (repeatedly)	move - click LMB	move pointer over the menu button and click the LMB (SELECT).

*** To **pin** a sub-window, the user needs to point to the pushpin in the upper left corner and click LMB (SELECT). To move any window, the user needs to press the LMB (SELECT) on the title area of the window, drag to a new location, then release the LMB. The window will then be relocated. To close the sub-window, **un-pin** it by moving the point over the pin and clicking the LMB.

Drawing a net is simple. Once the Draw window is displayed, there are seven object items that can be chosen from the window to draw the net: place, horizontal immediate transition, vertical immediate transition, horizontal timed transition, vertical timed transition, arc, inhibit arc and token.

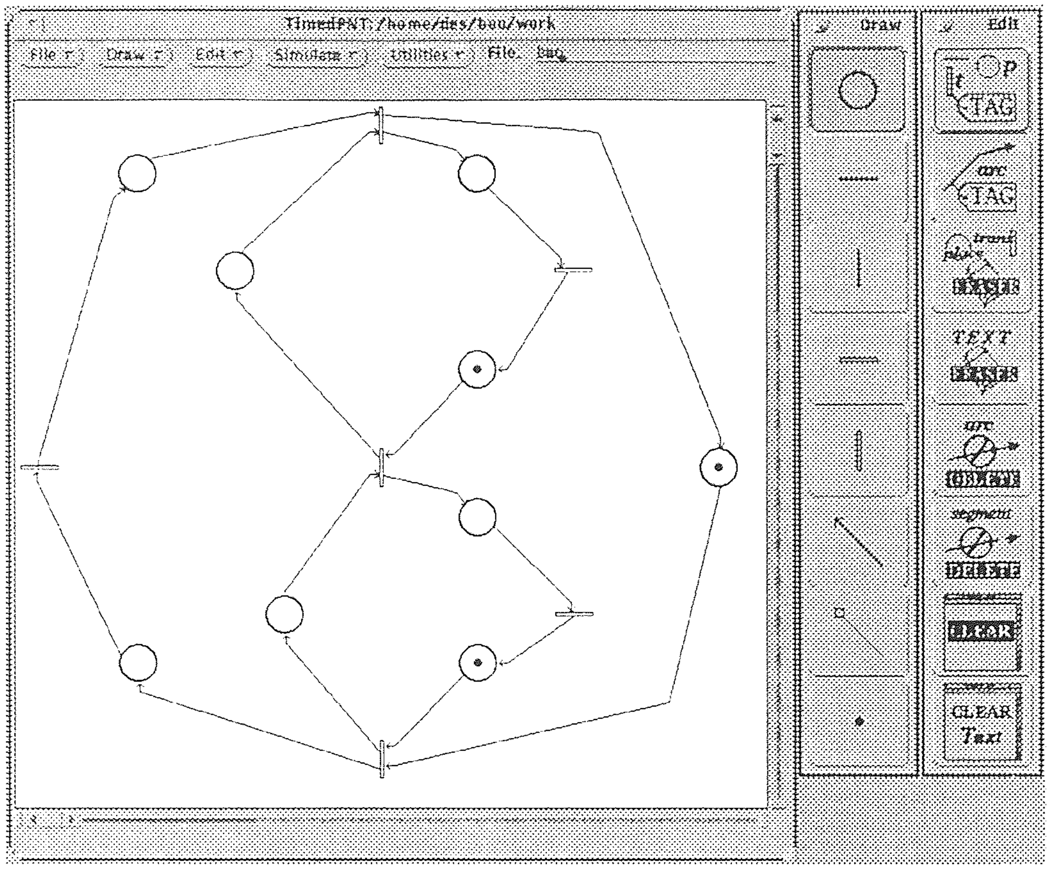


Figure 5.2 Pinning Menu Sub-windows To The Side

<u>Places and Transitions:</u>

To draw a place or a transition, first choose the appropriate menu item; then, move the pointer to the desired location on the canvas and click the LMB (SELECT) to place the object.

<u>Tokens:</u>

Similar to drawing a place or a transition, the user must first choose the token object from the menu, then move the pointer to the desired place and click the LMB

(SELECT). No token can be put outside of a place. The appearance of tokens present in a place is represented by dots for up to four tokens. If the token number is greater than four, the number will appear in the place instead of dots.

Arc:

In TimedPNT, an arc begins with an arrow (for a normal arc) pointing to an object (a place or transition), and is followed by connected straight lines, called segments, to connect it to the end object (a transition or place).

To draw an arc, the user needs to move the pointer to the desired location where an arc arrow begins, click the LMB (SELECT) to "put down" the arrow on the canvas, then move the pointer to the far end of the first segment and click the RMB (MENU), and so on, until the end object is reached. Once an end object is reached, a notice message will show on the canvas informing the user that the arc has been completed. The following rules apply to drawing arcs:

- For the same pair of place and transition, no multiple arcs of the same type (e.g. INPUT, INHIBIT or OUTPUT) can be visually drawn. The multiplicity of the same type of arc between the same pair of place and transition is represented by Weight in the arc tag (See Weight item in arc tag).

- Arcs can not be drawn between two places or between two transitions; if the user attempts this, the arc is declared as illegal by a warning message and automatically removed.

- An arc can be drawn crossing places, transitions and other arcs if necessary.

- An inhibit arc is drawn in the same way as a normal arc.

A convenient characteristic of TimedPNT's drawing features is that the user can draw many objects of a particular type without repeatedly choosing the object from the Draw Menu. For example, after choosing the place object, the user need only press LMB in different locations to draw numerous places. Therefore, a planned net can be easily drawn, choosing only a few times from the menu items.

5.4.3 Editing A Net

Erasing elements

To erase a place or transition:

In the TimedPNT Edit Menu, the third item is the **Eraser**. To erase a place or a transition, choose the Eraser item from the Edit Menu, move the pointer to a place or a transition on the canvas and click the LMB (SELECT). As a consequence of the removal action, all arcs connected to the removed place or transition are also removed.

To erase tokens:

The same Eraser item used for removing places and transitions is used to remove tokens from a place. The operational difference is that the user must click on the RMB instead of the LMB on the place from which the token is to be removed.

To delete an arc:

The fifth and the sixth items in the Edit Menu are **Arc Delete** and **Segment Delete**, respectively. The Arc Delete is used to delete a completed arc (i.e. after the message is displayed to inform the user that the arc is considered completed). To perform the deletion task:

1. choose the Arc Delete item,
2. move the pointer to the head of the arc (arrow for a normal arc, or a small box for an inhibit arc),
3. click the LMB (a message will inform you that this was done successfully),
4. click the RMB (MENU) on the respective end object (a place or a transition).

The Segment Delete item is used to delete unwanted segments while in the process of drawing an arc (before the arc is completed). The last drawn segment is removed when the LMB is clicked on the Segment Delete button. The clicking can be successively repeated until all the segments of the incomplete arc are removed.

To Clear the Net

The **Clear** item in the Edit Menu is used to clear the canvas and start a new drawing. A warning message will inform the user to save current work if it has not been saved. All dynamically allocated data structures are destroyed in this action.

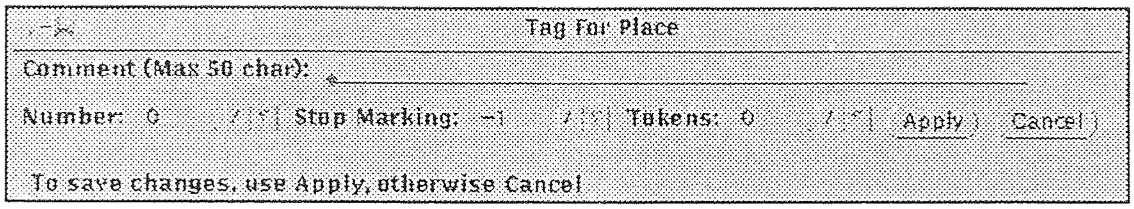
<u>Assigning Porperties</u>

Tags are used to designate the parameters for places, transitions and arcs. There are four types of Tags in TimedPNT: Place Tag, Immediate Transition Tag, Timed Transition Tag and Arc Tag. The first two items of the Edit Menu are used to invoke these tags.

In TimedPNT, a Tag is a pop-up sub-window, which contains window items. The following are the Tag window items which a Tag may have.

- *Title*: shows a tag identity.
- *Editable Text Field*: displays comments which the user can type in from the keyboard.
- *Non-Editable Text Field*: displays some properties of an object (which can not be edited by the user.
- *Numeric Field*: displays the current value of a parameter which can be changed by the user either by typing from the keyboad or clicking the LMB on its increment or decrement button.
- *Choice Button*: gives the user a choice on a particular property. NOTE: Cancelling (below) does NOT cancel choice button changes. To change an unwanted choice, simply click the correct choice button again.
- *Apply and Cancel Buttons*: gives the user the editing choice of applying or cancelling the editing which has been made. This does NOT apply to choices made by Choice Buttons.

All place and transition Tags can be accessed through the Tag item in the Edit Menu. Arc Tags are accessed by the Arc Tag item in the Edit Menu.


Tag For Place

Comment (Max 50 char):

Number: 0 Stop Marking: -1 Tokens: 0

To save changes, use Apply, otherwise Cancel

Figure 5.3 Place Tag Pop-up Window

Place Tag:

To access a Place Tag (See Figure 5.3), the user must choose the Tag item from the Edit Menu. A Place Tag contains the following items: Comment, Number, Stop Marking and Tokens. (See 3.1.2)

- **Document the Comment field**:

  Properly documenting each object is useful in net analysis. It is strongly recommended.

- **Assigning The Place Number**:

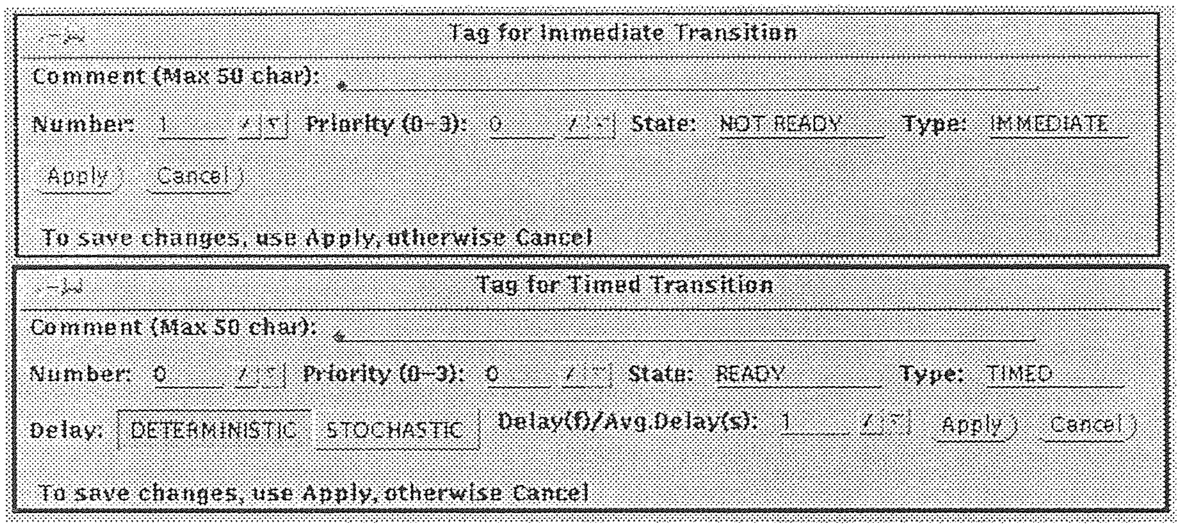
  When a net is drawn, each place has automatically been assigned a Place Number (ID Number) for its indentity. The numbering starts at 0 and is incremented successively in the order in which each place is added. If the user wants to rearrange this order, this number may be edited, but no two places should be given the same number. Otherwise, an internal erorr will occur during the net simulation without warning.

- **Assigning The Stop Marking**:

  The default value of the Stop Marking field is "-1", which means "DON'T CARE". Zero can be assigned as Stop Marking for a place, which means that an end condition has been reached as soon as the place is empty. (This is true unless the simulation is running on the "STOP BY NET MARKING" mode and the end condition(s) of other places have not yet been met. See 3.2.5 Simulation End Conditions).

- **Assigning Tokens**:

  The Tokens field offers a more efficient way to put in or take out tokens than to use the Draw Menu item. For a large number of tokens, it is the only way (unless you really want to click the LMB, say, 5000 times).


Tag for Immediate Transition

Comment (Max 50 char):

Number: 1 Priority (0-3): 0 State: NOT READY Type: IMMEDIATE

To save changes, use Apply, otherwise Cancel

Tag for Timed Transition

Comment (Max 50 char):

Number: 0 Priority (0-3): 0 State: READY Type: TIMED

Delay: DETERMINISTIC STOCHASTIC Delay(f)/Avg.Delay(s): 1

To save changes, use Apply, otherwise Cancel

Figure 5.4 Immediate and Timed Transition Pop-up Windows

Immediate Transition Tag:

The Immediate Transition Tag, Figure 5.4 (upper), is accessed in the same way as the Place Tag. The window items are: Comment, Number, Priority, State and Type. (See 3.1.2)

- **Assigning The Transition number**:

  Like the Place Number, the Transition ID Number is also automatically assigned for each transition from 0 to the maximum number of transitions, regardless of transition type (Immediate or Timed). If the user assigns the same number to more than one transition, errors will occur when running the net simulation.

- **Assigning The Transition Priority**:

  The default priority is 0. There are four priority levels: 0, 1, 2, and 3; where 0, the default value, is the highest, and 3 is the lowest.

- **State and Type field**:

  The State and Type fields are non-editable text fields which display the current state (READY or NOT READY) and the type of transition (IMMEDIATE or TIMED).

Timed Transition Tag:

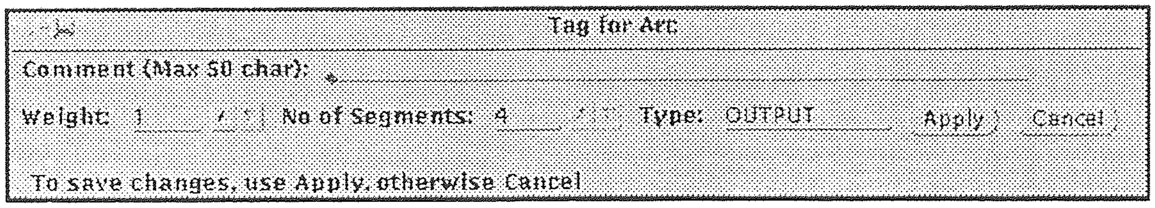
The difference between a Timed Transition Tag, Figure 5.4 (lower), and an Immediate Transition Tag is that the Timed Transition Tag contains two more items: Delay Type and Delay/Avg.Delay. The Delay Type item is a pair of Choice Buttons and The Delay/Avg.Delay is a Numeric Field. (See 3.1.2)

- **Setting up the Delay Type**:

  The default type for a Timed Transition is DETERMINISTIC. To change it, the user can simply click the LMB (SELECT) on the STOCHASTIC button. A notice message will be displayed. To change back, simply click the LMB again on the DETERMINISTIC button.

- **Assigning Delay or Average Delay**:

  The default value is 1. The value assigned to this field represents a fixed delay if the transition type is DETERMINISTIC, and represents an average delay if the transition type is STOCHASTIC.



To save changes, use Apply, otherwise Cancel

Figure 5.5 Arc Tag Pop-up Window

Arc Tag:

The Arc Tag, Figure 5.5, is accessed through the Arc Tag item in the Edit Menu. The user has to choose the Arc Tag item first, then click the LMB (SELECT) on the beginning of the arc (i.e. the arrow of a normal arc), then is prompted to click the RMB (MENU) on the end object of that arc. An Arc Tag contains Comment, Weight, No of Segment and Type items. (See 3.1.2)

The **Weight field** is the only field in the Arc Tag that can be edited by the user. (See 5.4.2, Arc)

5.5 Adding Text Onto The Canvas

After a net has been drawn, it is useful to add text on the canvas for the sake of clarification (e.g. to mark a place as "p2" or "B Status"). To write text on the canvas:

- Choose the Type item from the Utilities Menu
- Click the LMB at the desired starting location on the canvas
- Start typing.

Since the canvas is a graphical editing area, it does not support text editing features; therefore, no "Backspace" can be accepted. To edit existing text on canvas, the user has to erase the text and re-type. The Text Eraser item from the Edit Menu is used to erase text within a user defined rectangular region on the canvas. To perform this task, choose the Text Eraser, then click the LMB on one of the corners of the rectangular, then click again on the diagonally opposite corner, enclosing the text to be erased.

The user also can clear all existing text from the canvas by choosing the Clear Text item from the Edit Menu. WARNING: This clears ALL existing text.

5.6 File Functions

The File Menu is usually used at the beginning and end of sessions, to load previously drawn nets, and to save current changes.

- *Save File*

  Choose the Save item in the File Menu to save the current net, including its properties and current marking. The net should be saved before running a simulation, in order to save the initial marking.

- *Load File*

  Choose the Load item in the File Menu to load a previously drawn net, including all assigned properties.

5.7 Running a Simulation

Once a net has been completed, simulation can begin.

5.7.1 Specifying Simulation End Conditions

To specify simulation end conditions, choose the Break Pt. item from the Simulate Menu. This brings up a pop-up sub-window, Figure 5.6, which contains the following items: Steps, Clock, Stop Marking, and Log. (See 3.2.5 for Simulation End Conditions)

Stop Marking:

When using place markings as an end condition for simulation, the user must specify the place markings (token number) in the Stop Marking item(s) in the place tag(s), as well as choose a **mode** from the Simulate Break Pt. sub-window (Simulation End Conditions).

The Stop Marking item in the Simulation Mode sub-window gives the user a choice between two modes:

1. Stop by **Place** Marking (default)

2. Stop by **Net** Marking

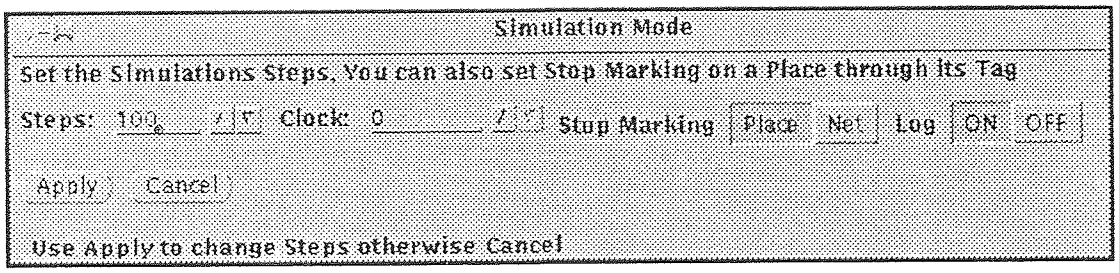
In case 1, the simulation will terminate as soon as any place has reached the user-specified stop marking for it. In case 2, the simulation will terminate only after all places have reached their user-specified stop markings.

If no **place** has been assigned a Stop Marking (other than -1), choosing the **Net** Stop Marking mode will cause the simulation to reach its end condition the moment it starts.

If no **place** has been assigned a Stop Marking (other than -1), and the **Place** Stop Marking mode is chosen (as default), then a Clock or Steps end condition will be used instead, according to Section 3.2.5.

Log Mode:

The Log choice buttons give the user the choice of turning the log file generation ON or OFF. This is a switch that remains ON or OFF until explicitly modified. This is true for both STEP and RUN modes. (Turn OFF for quicker run times.)



Simulation Mode

Set the Simulations Steps. You can also set Stop Marking on a Place through its Tag

Steps: 100 Clock: 0 Stop Marking [✓ Place] Net Log [✓ ON] OFF

Use Apply to change Steps otherwise Cancel

Figure 5.6 Simulation End Conditions Pop-up Window

5.7.2 Running the Simulation

Running the simulation can be carried out in one of two modes: STEP or RUN. (See 3.2.6 and 3.2.7)

To STEP, the user may either click LMB on the Simulation Menu button (Step is chosen as default), or choose the Step item from the Simulation Menu.

To RUN, choose the Run item from the Simulation Menu, after all end conditions have been carefully chosen.

5.7.3 Resetting Parameters

In RUN mode, parameters are automatically reset before simulation begins. In STEP mode, however, parameters are not reset, so it may be useful to manually reset. To do this, choose the Reset item from the Simulation Menu to perform the following:

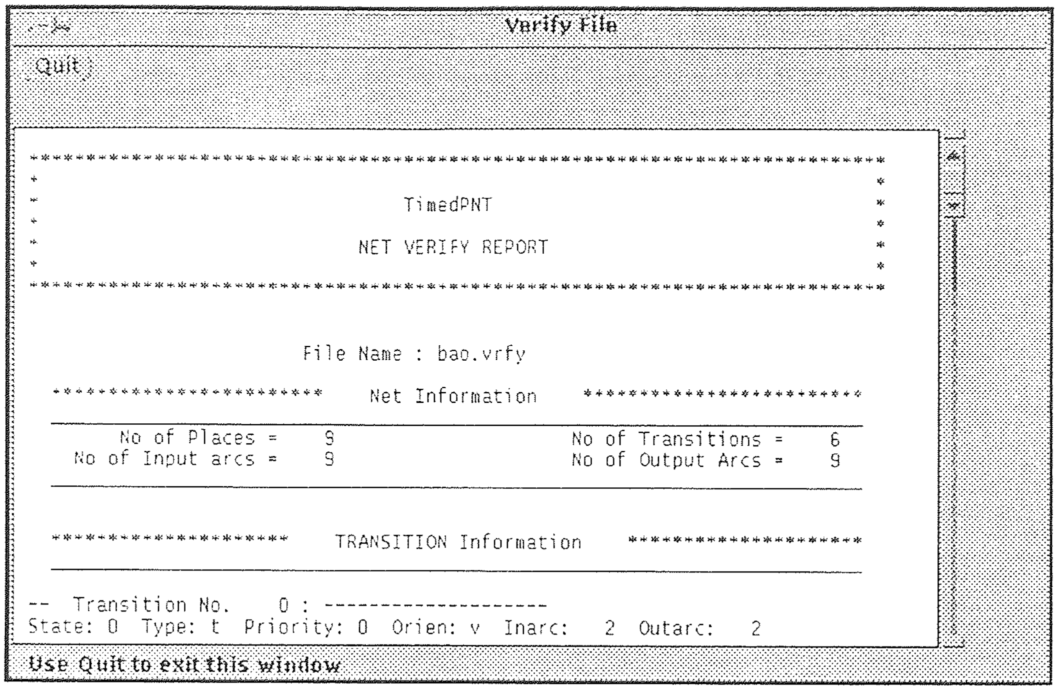
- Reset System Clock to 0
- Reset Utilization parameters

5.8 Reports

There are three types of reports generated by TimedPNT, as described below. These reports are saved in ASCII text files, and can be printed or read while running TimedPNT.

- *Verify Report* (*filename*.vrfy)

  This report verifies the structural properties of the net (See 3.1.2). To see this report, choose the Verify item under the Utilities Menu. Figure 5-11 shows a Verify pop-up window.

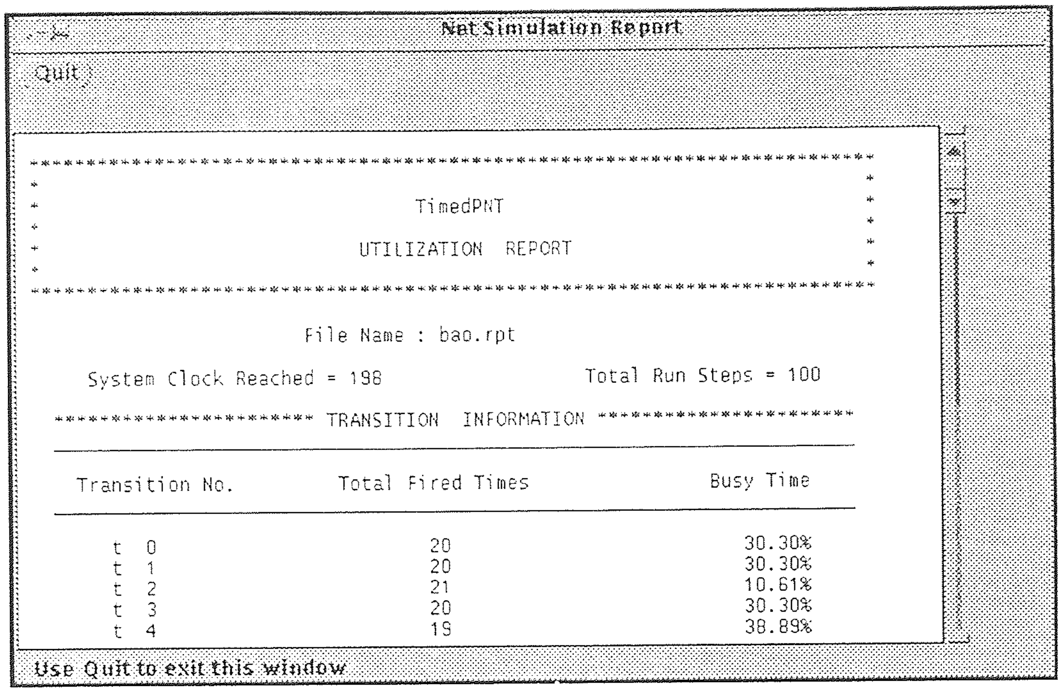


```
*****************************************************************************
*                                                                           *
*                               TimedPNT                                    *
*                                                                           *
*                          NET VERIFY REPORT                                *
*                                                                           *
*****************************************************************************

                        File Name : bao.vrfy

  ***********************     Net Information     ***********************
  ----------------------------------------------------------------------
       No of Places =     9                 No of Transitions =    6
   No of Input arcs =     9                 No of Output Arcs =    9
  ----------------------------------------------------------------------

  ********************      TRANSITION Information      ********************
  ----------------------------------------------------------------------
-- Transition No.     0 : --------------------
State: 0  Type: t  Priority: 0  Orien: v  Inarc:   2  Outarc:   2
```

Figure 5.7 Verify Report Pop-up Window

- *Utilization Report* (*filename*.rpt)

This report contains the utilization information of transitions and places after a simulation has been completed. In RUN mode simulations, this report is automatically generated and will appear in a pop-up window upon reaching the end of a simulation. Before another RUN mode simulation is performed, this file should be renamed; otherwise, it will automatically be deleted when a new report is generated. In STEP mode simulation, the information is accumulated with each step, and the user can generate an updated report at any point by choosing the Report item under the Utilities Menu. (Old reports should be renamed.)

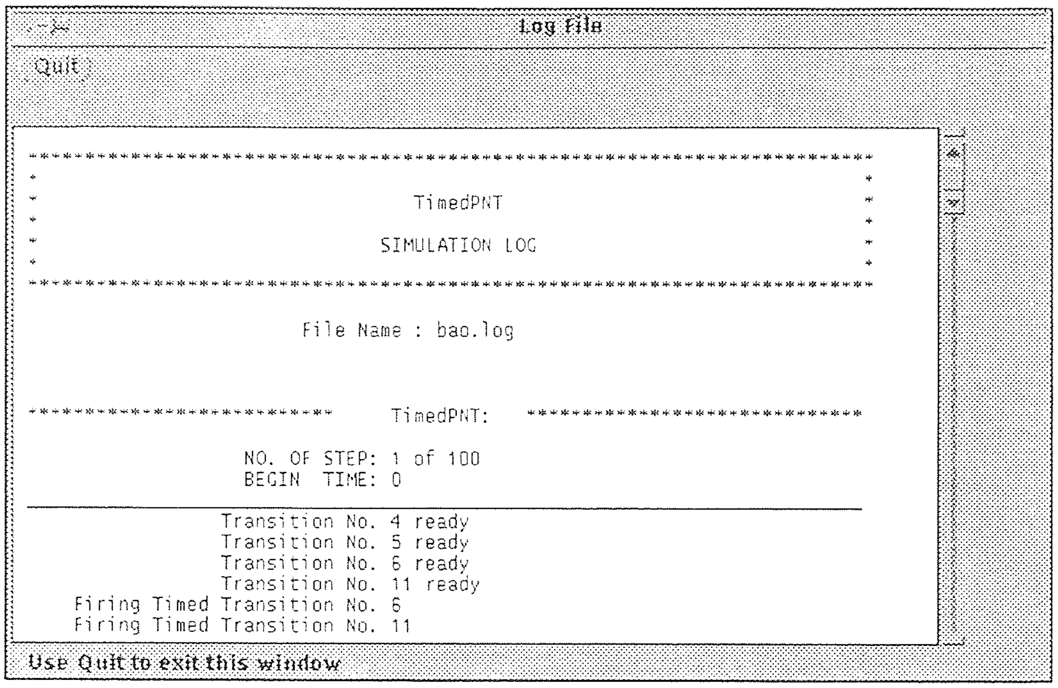


```
*****************************************************************
*                                                               *
*                          TimedPNT                             *
*                                                               *
*                     UTILIZATION  REPORT                       *
*                                                               *
*****************************************************************

                     File Name : bao.rpt

   System Clock Reached = 198              Total Run Steps = 100

  ********************** TRANSITION  INFORMATION **********************
  ____________________________________________________________________

   Transition No.          Total Fired Times                Busy Time
  ____________________________________________________________________

       t  0                       20                         30.30%
       t  1                       20                         30.30%
       t  2                       21                         10.61%
       t  3                       20                         30.30%
       t  4                       19                         38.89%
```

Figure 5.8 Utilization Report Pop-up Window

- *Log Report* (*filename*.log)

The Log report keeps track of marking and transition firing information (see 3.1.2). This file is automatically created with RUN or STEP mode simulations, and accumulates information with each step (unless Log Mode is OFF: see 5.7.1). This file will not be deleted or rewritten unless the user does so by choosing the Del Log item from the Utilities Menu. Simulations with large numbers of steps will create large log files. Therefore, it is recommended that unnecessary log files be deleted, and that Del Log be chosen before any new simulations are run.



```
*************************************************************************
*                                                                       *
*                              TimedPNT                                 *
*                                                                       *
*                           SIMULATION LOG                              *
*                                                                       *
*************************************************************************

                       File Name : bao.log


*************************     TimedPNT:     ****************************

                  NO. OF STEP: 1 of 100
                  BEGIN  TIME: 0
_________________________________________________________________________
                 Transition No. 4 ready
                 Transition No. 5 ready
                 Transition No. 6 ready
                 Transition No. 11 ready
    Firing Timed Transition No. 6
    Firing Timed Transition No. 11
```

Figure 5.9 Log Report Pop-up Window

5.9 Other Functions

- *Redraw*

  Redraw is used to redisplay the net drawing if the screen develops temporary display anomalies, for example, when erasing text or deleting arcs. This command can be invoked by choosing the Redraw item under the Utilities Menu.

- *Quit*

  To exit the TimedPNT, the user should always use the Quit item under the Files Menu, instead of the Quit button in the OpenWindows^{TM} window menu. This ensures that if the net has not been saved, the user will be prompted and given a chance to do so.

CHAPTER 6

CONCLUSION

The TimedPNT is one step forward in the development of the Petri net modeling tool in NJIT. As result of this effort:

- The TimedPNT is capable of simulating systems with both deterministic and stochastic delays.
- Performance and utilization results are automatically collected.
- OOP is employeed to ease future modification without corrupting existing code.
- The XView^{TM} based interface makes it possible for the TimedPNT to run remotely on a network.
- Compliance with the X11 standard makes the TimedPNT portable to other X11 based systems

For a more advanced Petri net simulation tool, future work might include the following:

- The TimedPNT can also be further extended into a Colored Petri Net Simulation Tool, to increase the modeling capabilities, considerably.

- Extension to conflict handling:
  - Capablity for specifing a selection probability for a particular transition in conflict.
  - Rather than using preselection, the transition with the least delay is selected to fire in a conflict situation.
- Extension to Stochastic Model:
  - Other than having a exponential distribution, other cumulative distributions can also be incorporated, for example, uniform distribution and bounded distributions.[9]

APPENDIX A

LIST OF TimedPNT FILES

(Total: 51 files, 268 Kbytes)

arc.C	940
arc.h	928
arc.image	1855
arctag.image	2190
arrow.image	1997
basic_object.C	572
basic_object.h	558
clear.image	1855
clear_net.image	2190
clear_text.image	2190
controller.C	14657
controller.h	1160
del_arc.image	2190
del_seg.image	2190
erase.image	2190
eraser.image	1855
htransition.image	1855
htransition1.image	1933
inh_arc.image	1855
makefile	1127
marker.image	1855
marker1.image	1933
net.C	6586
net.h	796
ootpns.C	140147
petri.image	1855
place.C	1576
place.h	816

```
place.image               1855
place1.image              1933
seg_array.C                852
seg_array.h                263
segment.C                  135
segment.h                  134
tag.image                 2190
test1.text                  20
text_control.C            1832
text_control.h             340
text_erase.image          2190
text_string.C              149
text_string.h              102
timed_ht.image            1933
timed_ht_fill.image       1933
timed_petri.image         1997
timed_vt.image            1933
timed_vt_fill.image       1933
tpnt                                (Executable)
transition.C              6498
transition.h              2812
vtransition.image         1855
vtransition1.image        1933
```

APPENDIX B

VERIFY FILE SAMPLE

```
**************************************************************************
*                                                                        *
*                              TimedPNT                                  *
*                                                                        *
*                          NET VERIFY REPORT                             *
*                                                                        *
**************************************************************************


                      File Name : bao.vrfy

 ************************   Net Information   ************************
 ____________________________________________________________________
      No of Places =   18                  No of Transitions =   11
  No of Input arcs =   18                  No of Output Arcs =   18
 ____________________________________________________________________


 ********************   TRANSITION Information   ********************
 ____________________________________________________________________

--  Transition No.    0 : --------------------
State: 0  Type: t  Priority: 0  Orien: v  Inarc:   5  Outarc:   5
  Connected Arcs:
 arc type: 1        pl_no:    2        weight: 1        seg_no. :   4
 arc type: 1        pl_no:    7        weight: 1        seg_no. :   4
 arc type: 1        pl_no:   11        weight: 1        seg_no. :   4
 arc type: 1        pl_no:   14        weight: 1        seg_no. :   5
 arc type: 1        pl_no:   17        weight: 1        seg_no. :   4
 arc type: 0        pl_no:    0        weight: 2        seg_no. :   5
 arc type: 0        pl_no:    8        weight: 1        seg_no. :   4
 arc type: 0        pl_no:    9        weight: 1        seg_no. :   6
 arc type: 0        pl_no:   12        weight: 1        seg_no. :   6
 arc type: 0        pl_no:   15        weight: 1        seg_no. :   5

--  Transition No.    1 : --------------------
State: 0  Type: t  Priority: 0  Orien: v  Inarc:   2  Outarc:   2
  Connected Arcs:
 arc type: 1        pl_no:    5        weight: 1        seg_no. :   4
 arc type: 1        pl_no:    6        weight: 1        seg_no. :   4
 arc type: 0        pl_no:    3        weight: 1        seg_no. :   4
 arc type: 0        pl_no:    7        weight: 1        seg_no. :   4

--  Transition No.    2 : --------------------
State: 0  Type: t  Priority: 0  Orien: v  Inarc:   3  Outarc:   3
  Connected Arcs:
 arc type: 1        pl_no:    0        weight: 1        seg_no. :   5
 arc type: 1        pl_no:    4        weight: 1        seg_no. :   4
 arc type: 1        pl_no:    9        weight: 1        seg_no. :   6
 arc type: 0        pl_no:    1        weight: 1        seg_no. :   4
 arc type: 0        pl_no:    5        weight: 1        seg_no. :   4
 arc type: 0        pl_no:   10        weight: 1        seg_no. :   4
```

```
-- Transition No.    3 : --------------------
State: 0  Type: t  Priority: 0  Orien: h  Inarc:    1  Outarc:    1
  Connected Arcs:
 arc type: 1        pl_no:    3        weight: 1        seg_no. :   4
 arc type: 0        pl_no:    4        weight: 1        seg_no. :   4

-- Transition No.    4 : --------------------
State: 0  Type: t  Priority: 0  Orien: h  Inarc:    1  Outarc:    1
  Connected Arcs:
 arc type: 1        pl_no:    8        weight: 1        seg_no. :   4
 arc type: 0        pl_no:    6        weight: 1        seg_no. :   4

-- Transition No.    5 : --------------------
State: 0  Type: t  Priority: 0  Orien: h  Inarc:    1  Outarc:    2
  Connected Arcs:
 arc type: 1        pl_no:    1        weight: 1        seg_no. :   4
 arc type: 0        pl_no:    2        weight: 1        seg_no. :   4
 arc type: 0        pl_no:   14        weight: 1        seg_no. :   4

-- Transition No.    6 : --------------------
State: 0  Type: t  Priority: 0  Orien: h  Inarc:    1  Outarc:    1
  Connected Arcs:
 arc type: 1        pl_no:   10        weight: 1        seg_no. :   4
 arc type: 0        pl_no:   11        weight: 1        seg_no. :   4

-- Transition No.    7 : --------------------
State: 0  Type: t  Priority: 0  Orien: h  Inarc:    1  Outarc:    1
  Connected Arcs:
 arc type: 1        pl_no:   13        weight: 1        seg_no. :   4
 arc type: 0        pl_no:   14        weight: 1        seg_no. :   4

-- Transition No.    8 : --------------------
State: 0  Type: i  Priority: 0  Orien: v  Inarc:    1  Outarc:    1
  Connected Arcs:
 arc type: 1        pl_no:   12        weight: 1        seg_no. :   5
 arc type: 0        pl_no:   13        weight: 1        seg_no. :   4

-- Transition No.    9 : --------------------
State: 0  Type: t  Priority: 0  Orien: v  Inarc:    1  Outarc:    0
  Connected Arcs:
 arc type: 1        pl_no:   15        weight: 1        seg_no. :   5

-- Transition No.   11 : --------------------
State: 0  Type: t  Priority: 0  Orien: h  Inarc:    1  Outarc:    1
  Connected Arcs:
 arc type: 1        pl_no:   16        weight: 1        seg_no. :   4
 arc type: 0        pl_no:   17        weight: 1        seg_no. :   4
________________________________________________________________________

**********************     Place Information    **********************
________________________________________________________________________
       Place No.                                    No. of Tokens

________________________________________________________________________
       p   0                                          16
       p   1                                           0
       p   2                                           1
       p   3                                           0
       p   4                                           1
       p   5                                           0
       p   6                                           0
       p   7                                           1
       p   8                                           0
```

p 9	0
p 10	0
p 11	1
p 12	0
p 13	0
p 14	18
p 15	0
p 16	0
p 17	0

APPENDIX C

LOG FILE SAMPLE

```
**************************************************************************
*                                                                        *
*                             TimedPNT                                   *
*                                                                        *
*                          SIMULATION LOG                                *
*                                                                        *
**************************************************************************

                       File Name : bao.log


*************************       TimedPNT:    ******************************

                 NO. OF STEP: 1 of 100
                 BEGIN  TIME: 0
__________________________________________________________________________
                Transition No. 4 ready
                Transition No. 5 ready
                Transition No. 6 ready
                Transition No. 11 ready
    Firing Timed Transition No. 6
    Firing Timed Transition No. 11
    Firing Timed Transition No. 5
    Firing Timed Transition No. 4
       System Clock Advanced by 1
           System Clock Reached 1

Marking of Net is :
Place No.   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15

Marking:   15   0   0   0   0   1   0   0   0   0   0   1   0   0  16   0

**************************************************************************


*************************       TimedPNT:    ******************************

                 NO. OF STEP: 2 of 100
                 BEGIN  TIME: 1
__________________________________________________________________________
                Transition No. 4 ready
                Transition No. 5 ready
    Firing Timed Transition No. 5
    Firing Timed Transition No. 4
       System Clock Advanced by 1
           System Clock Reached 2

Marking of Net is :
Place No.   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15

Marking:   15   0   1   0   0   1   0   0   0   0   0   1   0   0  17   0

**************************************************************************
```

```
**************************   TimedPNT:   ******************************

                   NO. OF STEP: 3 of 100
                   BEGIN  TIME: 2
________________________________________________________________________
                 Transition No. 4 ready
    Firing Timed Transition No. 4
       System Clock Advanced by 1
           System Clock Reached 3

Marking of Net is :
Place No.   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15

Marking:   15   0   1   0   0   1   1   0   0   0   0   1   0   0  17   0

************************************************************************


*************************   TimedPNT:   ******************************

                   NO. OF STEP: 4 of 100
                   BEGIN  TIME: 3
________________________________________________________________________
                 Transition No. 1 ready
    Firing Timed Transition No. 1
       System Clock Advanced by 3
           System Clock Reached 6

Marking of Net is :
Place No.   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15

Marking:   15   0   1   1   0   0   0   1   0   0   0   1   0   0  17   0

************************************************************************

*************************   TimedPNT:   ******************************

                   NO. OF STEP: 5 of 100
                   BEGIN  TIME: 6
________________________________________________________________________
                 Transition No. 0 ready
                 Transition No. 3 ready
    Firing Timed Transition No. 0
    Firing Timed Transition No. 3
       System Clock Advanced by 3
           System Clock Reached 9

Marking of Net is :
Place No.   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15

Marking:   17   0   0   0   1   0   0   0   1   1   0   0   1   0  16   1

************************************************************************


*************************   TimedPNT:   ******************************

                   NO. OF STEP: 6 of 100
                   BEGIN  TIME: 9
________________________________________________________________________
                 Transition No. 2 ready
                 Transition No. 4 ready
                 Transition No. 8 ready
                 Transition No. 9 ready
Firing Immediate Transition No. 8
Marking of Net is :
Place No.    0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15

Marking:    17   0   0   0   1   0   0   0   1   1   0   0   0   1  16   1

*************************************************************************


**************************     TimedPNT:    ******************************

                   NO. OF STEP: 7 of 100
                   BEGIN  TIME: 9
_________________________________________________________________________
                 Transition No. 2 ready
                 Transition No. 4 ready
                 Transition No. 7 ready
                 Transition No. 9 ready
    Firing Timed Transition No. 2
    Firing Timed Transition No. 7
    Firing Timed Transition No. 9
    Firing Timed Transition No. 4
       System Clock Advanced by 1
           System Clock Reached 10

Marking of Net is :
Place No.    0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15

Marking:    16   1   0   0   0   1   0   0   0   0   1   0   0   0  17   0

*************************************************************************
```

APPENDIX D

UTILIZATION REPORT FILE SAMPLE

```
**********************************************************************
*                                                                    *
*                           TimedPNT                                 *
*                                                                    *
*                      UTILIZATION  REPORT                           *
*                                                                    *
**********************************************************************

                      File Name : bao.rpt

    System Clock Reached = 141                  Total Run Steps = 100

  ********************** TRANSITION  INFORMATION **********************
  ____________________________________________________________________

   Transition No.         Total Fired Times                 Busy Time
  ____________________________________________________________________

      t  0                       14                            29.79%
      t  1                       14                            29.79%
      t  2                       15                            10.64%
      t  3                       14                            29.79%
      t  4                       14                            40.42%
      t  5                       14                            19.86%
      t  6                       14                             9.93%
      t  7                       15                            10.64%
      t  8                       15                            15.00%
  ____________________________________________________________________


  ************************* PLACE  INFORMATION ************************
  ____________________________________________________________________

     Place No.          Total Tokens Entered            Occupied Time
  ____________________________________________________________________

       p  0                       30                          100.00%
       p  1                       15                           19.86%
       p  2                       14                           69.51%
       p  3                       14                           29.79%
       p  4                       15                           10.64%
       p  5                       15                           59.58%
       p  6                       14                           29.79%
       p  7                       14                           29.79%
       p  8                       15                           40.43%
       p  9                       15                           10.64%
       p 10                       15                            9.93%
       p 11                       14                           79.44%
       p 12                       15                            0.00%
       p 13                       15                           10.64%
       p 14                       30                          100.00%
```

REFERENCES

1 Gilani, A. 1989. "A Graphical Editor for Petri Nets." Master's Thesis, Electrical and Computer Engineering Department, New Jersey Institute of Technology.

2 Shukla, A. 1990. "A Petri Net Simulation Tool." Master's Thesis, Electrical and Computer Engineering Department, New Jersey Institute of Technology.

3 Siddiqi, J.A. 1991. "A Graphical Tool for the Simulation of Timed Petri Nets." Master's Thesis, Electrical and Computer Engineering Department, New Jersey Institute of Technology.

4 Juneja, H. 1993. "Object Oriented Design of Petri Net Simulator." Master's Thesis, Electrical and Computer Engineering Department, New Jersey Institute of Technology.

5 Peterson, J.L. 1981. *Petri Net Theory and the Modeling of Systems*. New Jersey: Prentice-Hall.

6 Molloy, M. 1989. "Petri net Modeling: The Past, the Present, and the Future." *Proceedings of the Third International Workshop on Petri Nets and Performance Models*. pp. 2-9.

7 Ramachandani, C. 1973. "Analysis of Asynchronous Concurrent Systems by Times Petri Nets." Dissertation, Massachusetts Institute of Technology.

8 Sifakis, J. 1978. "Structural Properties of Petri Nets." *Mathematical Foundations of Computer Science*. Springer-Verlag. pp. 474-483.

9 Merlin, P. and D.J. Farber. 1976. "Implication of a Theoretical Study." *IEEE Transactions on Communications*. COM-24:9: 1036-1043.

10 Zubarek, W.M. 1980. "Timed Petri Net and Preliminary Performance Evaluations." *Proceedings of 7th Annual Symposium on Computer Architectures*.

11 Molloy, M. 1985. "Discrete Time Stochastic Petri Nets." *IEEE Transactions on Software Engineering*. 11:4: 417-423.

12 Chen, P.Z., S. Bruell, and G. Balbo. 1989. "Alternative Methods for Incorporating Non-exponential Distributions into Stochastic Time Petri Nets."

13 Marson, A.M., G. Balbo, and G. Conte. 1984. "A Class of Generalized Stochastic Petri Net for the Performance Evaluation of Multiprocessor Systems." *ACM Transactions on Computer Systems*. 2:2: 93-122.

REFERENCES
(Continued)

14 Haas, P.J. and G.S. Shedler. 1986. "Regenerative Stochastic Petri Nets." *Performance Evaluation*. 6:3: 189-204.

15 ————. 1993. *XView^{TM} Developer's Notes*. Mountain View, CA: SunSoft.

16 ————. 1991. *OpenWindows^{TM} User's Guide*. Mountain View, CA: SunSoft.