TOWARDS A GRAHICAL PETRI NET TOOL

By

MUHAMMAD TARIQ HASSAN

Bachelor of Engineering

N.E.D. University of Engineering and Technology

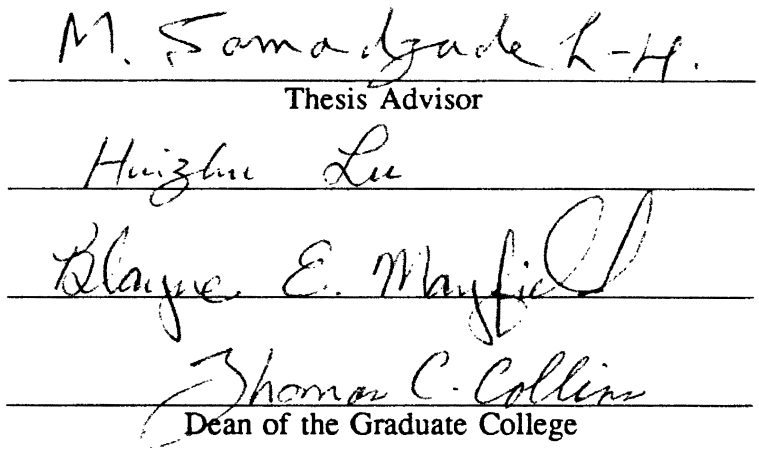
Karachi, Pakistan

1990

Submitted to the faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirement for
the Degree of
MASTER OF SCIENCE
July 1993

TOWARDS A GRAHICAL PETRI NET TOOL

Thesis Approved:



Thesis Advisor

Dean of the Graduate College

ACKNOWLEDGEMENTS

I thank my graduate advisor Dr. Mansur H. Samadzadeh for his advice, assistance, and guidance. His constructive criticism helped me gain confidence. During my whole graduate studies, I got inspiration and motivation due to his constant guidance. My sincere thanks to Drs. Blayne Mayfield and Huizhu Lu for serving on my graduate committee.

I also want to thank Mr. Larry Watkins, my supervisor at the University Computer Center, OSU, for allowing flexible working hours.

Finally, I would like to express my gratitude to my parents, brothers, and sisters. Without their support and encouragement, this task would not have been possible.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Modeling of a system is an important stage of its development life cycle. First a system is modeled and then the model is analyzed for conformity with the requirements and the expected behavior. A system can be initially designed on paper but its dynamic behavior is generally difficult to demonstrate. Modeling of systems has become possible on computers due to the availability of tools for different types of problems. Using these tools, one can design and analyze various systems graphically. A complicated system can be broken down into well-defined subsystems, each of which can be modeled on one screen (for conceptual manageability) or further subdivided if necessary. The components of each subsystem can be represented by simple graphical shapes. Subsystems can communicate with one another as needed and some initial conditions may be needed by each subsystem. These conditions may be inputs originating in other subsystems.

Using such an approach, one can design and model the behavior of any subsystem assuming that its initial condition is satisfied. After modeling of a subsystem is done, one can analyze the model for its required behavior and can refine the design if some discrepancies are detected. By an iterative process of analysis and modification, one can come up with a desired model of each subsystem. Finally, the subsystems can be put together to construct a model of the whole system, which can then be implemented.

For representing a model on a monitor's screen, one can use a simple directed graph with nodes and arcs; with the nodes representing the different parts of a system or the different parts of a subsystem, and the arcs showing their interactions. But this approach cannot capture the parallelism and the resulting synchronization which may exist among the different parts.

Petri Net is a versatile modeling tool that can be an answer to the modeling problems mentioned in the last paragraph. A Petri Net is an abstract and formal model of the information flow of a system [Peterson77]. It is a graphical and mathematical tool that can be applied to different systems [Murata89]. A Petri Net describes a system which may exhibit concurrent and/or asynchronous properties. Since it is also a mathematical model, it can be analyzed for different properties such as reachability, safeness, boundedness, and liveness [Murata89].

The main objective of this thesis was to develop a graphical tool which can help in developing a Petri Net model of a given system and to animate the execution of the Petri Net, i.e., to provide the before-firing and after-firing snapshots of a given Petri Net and to track the movements of tokens upon firing of a transition (for definitions refer to Chapter II).

Chapter II of this thesis provides a literature review on Petri Nets. The design and the implementation issues of the software tool developed as part of this thesis are presented in Chapter III. In Chapter IV, the evaluation of the tool is discussed. Finally, Chapter V summarizes the thesis, gives some conclusions, and suggests some areas of future work.

CHAPTER II

LITERATURE REVIEW

2.1 Definitions

This section contains some of the basic definitions about Petri Nets used in this thesis. These definitions are mostly based on two of the major references on Petri Nets, namely [Agerwala79] and [Murata89].

A *Petri Net* is a triple, PN = (P, T, A), where $P = \{p_1, p_2, ..., p_n\}$ is a finite set of places, $T = \{t_1, t_2, ..., t_n\}$ is a finite set of transition, and $A \subseteq \{P \times T\} \cup \{T \times P\}$ is a set of directed arcs.

A *place* in a Petri Net graph is a node that can be represented by a circle. Each place p has a preset and a postset. The *preset of a place* is defined as $*p = \{t \mid (t,p) \in A\}$, where *p represents the preset of the place p. The *postset of a place* is defined as $p* = \{t \mid (p,t) \in A\}$, where p* represents the postset of the place p. The preset and the postset of a place p define its interconnections with a set of transitions.

A *transition* in a Petri Net is a node which can be represented by a bar. Similar to a place, each transition also has a preset and a postset, which define its interconnections with a set of places.The *preset of a transition* is defined as $*t = \{p \mid (p,t) \in A\}$, where *t represents the preset of the transition t. The *postset of a transition* is defined as $t* = \{p \mid (t,p) \in A\}$, where t* represents the postset of the transition t.

An *enabled transition* is a transition that can fire. *Firing* of a transition means that it removes one token from each place present in its preset and adds one token to each place present in its postset. An enabled transition must satisfy the following condition: $\forall\ p \in\ {}^*t,\ M(p) \geq 1$, where $^*t$ is the preset of the transition t, i.e., the set of places incident on transition t, and M(p) represents the mapping of the place p to a non-negative integer, i.e., the number of tokens in place p.

A *Marked Petri Net*, PN_m, is defined as $PN_m = (PN, M_0)$ or $PN_m = (P, T, A , M_0)$, where M_0 is the initial marking of the Petri Net and M_0: P --> I, where I = {0, 1, ...}. A mapping of a place p to an integer i, $i \geq 0$, is interpreted as i *token(s)* being present in place p. Hence for a marked Petri Net PN_m, if place p_1 is mapped to 3, we say that place p_1 contains 3 tokens. A *marking* is represented by a |P|-vector M, where |P| is the total number of places. The jth entry of the vector M gives the number of tokens currently in place p_j.

A *Petri Net graph* can be formally defined as a bipartite directed graph. It has two types of nodes: circles and bars. Circles represent places and bars represent transitions. Places and transitions are connected via directed arcs. No arcs are allowed from places to places or transitions to transitions, hence bipartite. Tokens are represented by small filled circles or black dots inside the places. A sample Petri Net graph is given in Figure 1.

The marked Petri Net, shown in Figure 1, can be defined as $PN_m = (P, T, A, M_0)$, where $P = \{p_1, p_2, p_3, p_4, p_5\}$, $T = \{t_1, t_2, t_3, t_4\}$, $A = \{(p_1, t_2), (p_2, t_3), (p_3, t_1), (p_4, t_3), (p_5, t_4), (t_1, p_4), (t_2, p_2), (t_3, p_1), (t_3, p_5), (t_4, p_3)\}$, and $M_0 = (0, 1, 1, 0, 0)$.

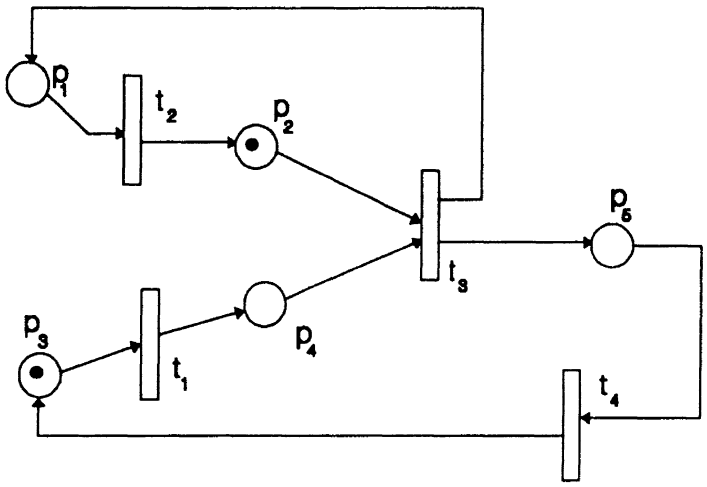


Figure 1. Example of a Petri Net graph

The initial marking, M_0 = (0, 1, 1, 0, 0), indicates that the places p_2 and p_3 contain one token each. At this initial marking, the only enabled transition is t_1. Firing of t_1 leads to the marking M_1 = (0, 1, 0, 1, 0). Now t_3 is enabled, i.e., it can fire. The successive firings of the transitions and the resulting movements of tokens is referred to as the execution of a Petri Net.

2.2 Properties of Petri Nets

2.2.1 Reachability

The notion of reachability captures all the possible configurations that can be derived from a given initial configuration [Peterson81] [Murata89]. For a marked Petri Net PN_m = (PN, M_0), a marking M_n is said to be reachable from a marking M_0 if there exists a firing sequence which leads to marking M_n from marking M_0. The set of all reachable markings, called the *Reachability Set* for PN_m, is represented by R(PN, M_0). The

Reachability Set R(PN, M_0) for a Petri Net PN = (P, T, A) with an initial marking M_0 is the set of markings recursively defined by:

- $M_0 \in R(PN, M_0)$, and
- If $M_1 \in R(PN, M_0)$ and M_2 is reachable from M_1 by a firing of a transition t enabled for M_1, then $M_2 \in R(PN, M_0)$.

2.2.2 Safeness and Boundedness

A place in a Petri Net is safe if the number of tokens in that place never exceeds one [Peterson81]. A Petri Net is safe if all its places are safe. Boundedness is a generalization of safeness. If we declare a Petri Net as being safe when the contents of each of its places never exceeds n tokens, then we say that the given Petri Net is n-safe or n-bounded, where n is a finite non-negative integer [Murata89].

2.2.3 Liveness

A Petri Net is said to be live if, from any marking, it is possible to fire any transition of the net by following some possible firing sequence [Peterson81] [Murata89]. Being live for a Petri Net means that there is no deadlock in the system represented by that Petri Net.

Different levels of liveness are defined as follows [Commoner72] [Peterson81] [Murata89]:

- Level 0: A transition t is live at Level 0 if it can never be fired.
- Level 1: A transition t is live at Level 1 if it can be fired at least once in any reachable marking.

- Level 2: A transition t is live at Level 2 if it can be fired at least n times where n is a finite positive number.
- Level 3: A transition t is live at Level 3 if it can be fired infinitely often in some infinite firing sequence.
- Level 4: A transition t is live at Level 4 if it can be fired from any reachable marking.

2.2.4 Coverability

A marking M_1, for a marked Petri Net PN_m, is said to be coverable if there exists a reachable marking M_2 such that, $\forall\ p \in P$, $M_2(p) \geq M_1(p)$, where $M_2(p)$ stands for the number of token(s) in place p for marking M_2 and $M_1(p)$ stands for the number of token(s) in place p for marking M_1.

2.3 Modeling with Petri Nets

Petri Nets can be used to model different systems, especially systems which contain parallelism. A Petri Net can capture the potential parallelism effectively. The Petri Net shown in the Figure 2 is equivalent to a PARBEGIN/PAREND construct.

A finite state machine can be represented by a subclass of Petri Nets known as state machine Petri Nets [Murata89]. Transitions in this class of Petri Nets have exactly one incoming arc and exactly one outgoing arc, i.e., the indegree and the outdegree of every transition is one. Figure 3 depicts an example of a finite state machine (FSM) and its corresponding state machine Petri Net.

Petri Nets can represent the flow of control in computer programs containing different constructs such as DO-WHILE, IF-THEN-ELSE, CASE, and PARBEGIN-PAREND [Agerwala79] [Peterson81].

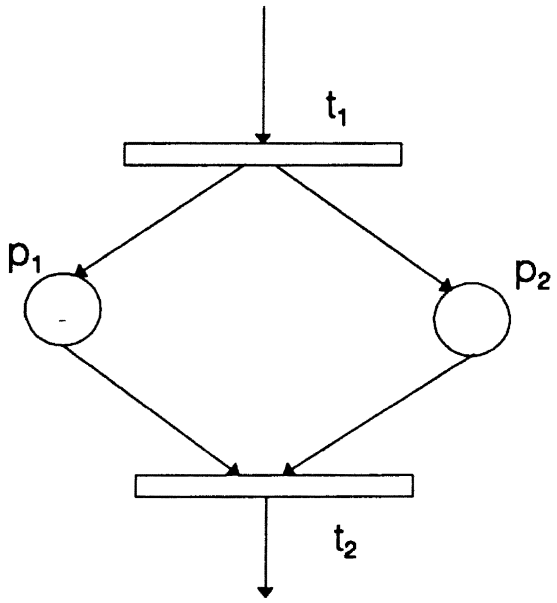


Figure 2. The Petri Net of a PARBEGIN/PAREND construct

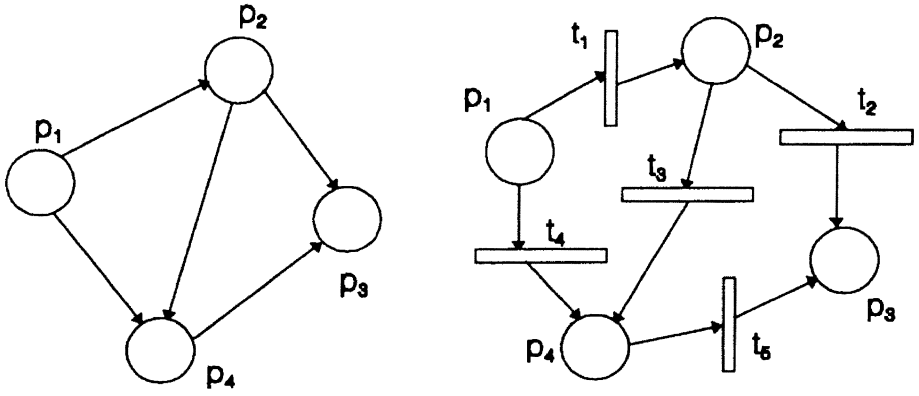


Figure 3. An FSM and its equivalent state machine Petri Net

2.4 Analysis of Petri Nets

A Petri Net can be analyzed for its various properties (see Section 2.2). There are different approaches to analyzing a Petri Net. Murata classifies different methods of analyzing Petri Nets into three categories [Murata89]: coverability/reachability tree, incidence matrix and state equation, and reduction techniques. The following subsections describe these three categories of methods for analyzing Petri Nets.

2.4.1 Coverability/Reachability Tree Method

The Reachability tree represents the reachability set of a Petri Net (see Section 2.2.1). A sample Petri Net and its reachability tree is depicted in Figure 4 (adapted from [Peterson81]).

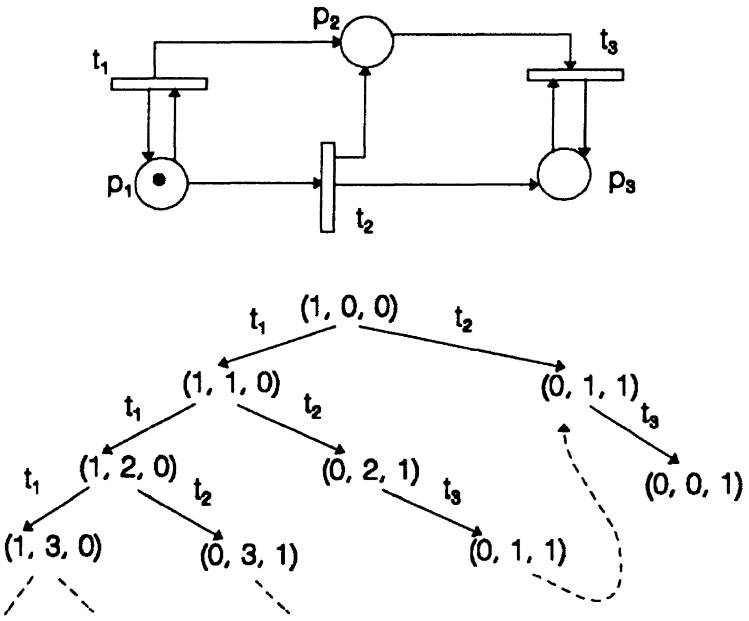


Figure 4. A Petri Net and its corresponding reachability tree

2.4.2 Incidence Matrix and State Equation

Murata and Johnsonbaugh discuss the matrix approach to analyzing the behavior of systems modeled by Petri Nets [Murata89] [Johnsonbaugh89]. For a given marked Petri Net $PN_m = (P, T, A, M_0)$, with $|P| = m$ and $|T| = n$, the incidence matrix $A = [a_{ij}]$ is an n by m matrix of integers, such that $a_{ij} = a_{ij+} - a_{ij-}$, where:

a_{ij+} is the number of arcs from transition t_i to place p_j (output place), and

a_{ij-} is the number of arcs from place p_j (input place) to transition t_i.

The entry a_{ij} of the incidence matrix gives the number of tokens changed in place p_j when transition t_i fires. The marking M_k is represented by an m x 1 column vector of non-negative integers. The jth entry of the incidence matrix represents the number of tokens in place p_j immediately after the kth firing. The state equation for a Petri Net can be written as, $M_k = M_{k-1} + A^T U_k$, $\quad k = 1, 2, \ldots$

where U_k is an n x 1 column vector called the control vector containing only one non-zero entry (i.e., containing a 1 at position i indicating that t_i fires in the kth firing), A^T is the transpose of the incidence matrix A, and M_{k-1} and M_k give the markings before and after the kth firing, respectively.

Murata gives the following theorem for the reachability of a marking M_d from an initial marking M_0 [Murata89].

If M_d is reachable from M_0 in a marked Petri Net $PN_m = (PN, M_0)$, then $(B_f)(\Delta M) = 0$, where ΔM is defined as $M_d - M_0$, and B_f is a fundamental circuit matrix.

The fundamental circuit matrix Bf [Deo74] is given by $B_f = [I_u : - A_{11}^T (A_{12}^T)^{-1}]$.

To get the fundamental circuit matrix Bf, the incidence matrix A of order n x m is partitioned as follows.

$$A = \begin{matrix} \\ \end{matrix} \begin{matrix} m-r & r \end{matrix} \\ \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{matrix} r \\ n-r \end{matrix}$$

where A_{12} is a non-singular square matrix of order r (the rank of the incidence matrix) and the other matrices, A_{11}, A_{21}, and A_{22}, are of the orders shown above. I_u is the identity matrix of order u = m - r, A_{11}^T and A_{12}^T represent the transpose of matrices A_{11} and A_{12}, and the exponent -1 represents the inverse of a matrix.

2.4.2.1 Example This section contains a simple example (adapted from [Murata89]) to illustrate the concepts and notations introduced in the last subsection.

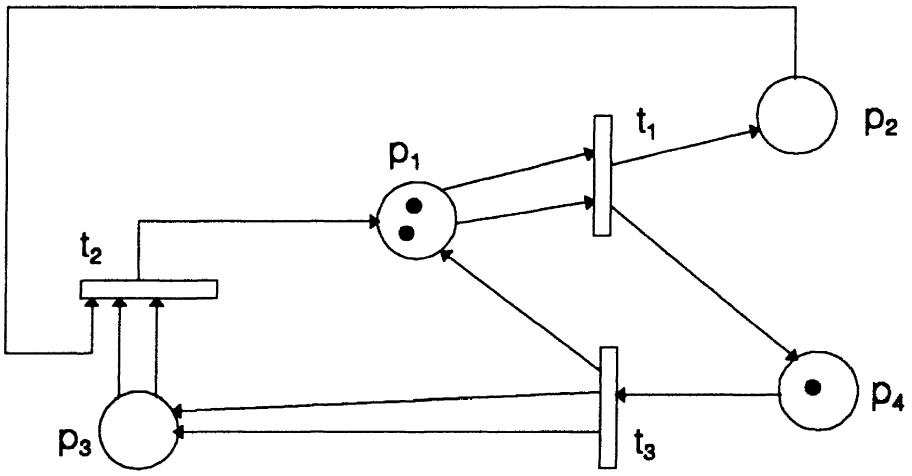


Figure 5. A Petri Net graph

Consider the Petri Net shown in Figure 5 . The corresponding incidence matrix $A = [a_{ij}]$ (see Section 2.4.2), of order 3 x 4, is given bellow.

$$A = \begin{bmatrix} -2 & 1 & 0 & 1 \\ 1 & -1 & -2 & 0 \\ 1 & 0 & 2 & -1 \end{bmatrix}$$

The entry $a_{11} = -2$ indicates that when transition t_1 fires, two tokens disappear from place p_1. Similarly, the entry $a_{33} = 2$ indicates that two tokens appear at place p_3 when transition t_3 fires. The initial marking M_0 is represented by a 4 x 1 column vector.

$$M_0 = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Consider the firing of transition t_3, changing marking M_0 to M_1, which can be given as follows.

$$M_1 = \begin{bmatrix} 3 \\ 0 \\ 2 \\ 0 \end{bmatrix}$$

The state equation $M_k = M_{k-1} + A^T U_k$, k = 1, 2, ... (see Section 2.4.2) can be written as follows.

$$\begin{bmatrix} 3 \\ 0 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 0 & -2 & 2 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The incidence matrix A is of rank 2 and can be partitioned into A_{11}, A_{12}, A_{21}, and A_{22} (see Section 2.4.2), where

$$A_{11} = \begin{bmatrix} -2 & 1 \\ 1 & -1 \end{bmatrix} \qquad A_{12} = \begin{bmatrix} 0 & 1 \\ -2 & 0 \end{bmatrix}$$

Hence the fundamental circuit matrix B_f (see Section 2.4.2) can be given as follows.

$$B_f = \begin{bmatrix} 1 & 0 & 1/2 & 2 \\ 0 & 1 & -1/2 & -1 \end{bmatrix}$$

Now, we can observe that $(B_f)(\Delta M) = 0$ holds for $\Delta M = M_1 - M_0$. Hence marking M_1 is reachable from marking M_0 (see the theorem in Section 2.4.2).

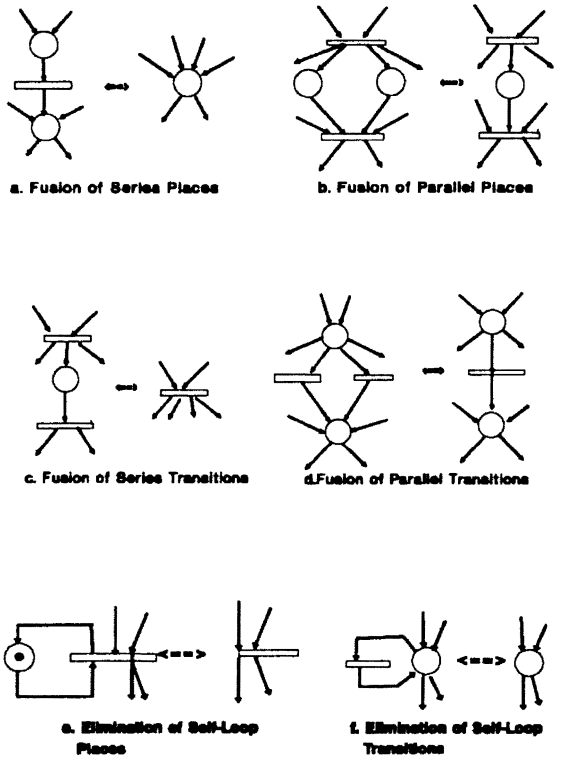


Figure 6. Six transformation techniques preserving safeness, liveness, and boundedness

2.4.3 Reduction Techniques

There are a large number of transformation techniques for Petri Nets [Murata89].

These techniques reduce the complexity of a Petri Net to some extent. Here we will present six simple transformations that preserve properties of safeness, liveness, and boundedness [Murata80]. These transformations are depicted in Figure 6.

CHAPTER III

IMPLEMENTATION ISSUES

3.1 Implementation Platform and Environment

3.1.1 Sequent Symmetry S/81

The Symmetry S/81 is a powerful mainframe-class multiprocessor system developed by Sequent Computer System, Inc. Its shared-memory, multiprocessing architecture consists of the following elements:

- A parallel architecture using multiple industry-standard microprocessors.
- The DYNIX/ptx or DYNIX V3.0 operating system, both UNIX system ports.
- Standard interfaces including Ethernet, MULTIBUS, VMEbus and SCSI [Sequent90].

The operating systems of the Symmetry S/81 have been engineered to incorporate parallel processing features. However, UNIX compatible software can run on the Symmetry S/81 without modification or with slight modification. In multi-user applications, tasks are automatically distributed to multiple processors which increases system throughput and reduces response times [Sequent90].

DYNIX V3.0 supports both the Berkeley UNIX and UNIX System V command sets, whereas DYNIX/ptx is compatible with AT&T System V3.2 only [Sequent90].

3.1.2 The X Window System

The X Window System is a software system used to develop graphical user-interfaces (GUI). Its device-independent nature allows programmers to develop portable GUIs [Young90] [Chandrashekar91] [Chandrashekar93], the only requirement being that the X protocol should be supported by the hardware. The X protocol defines the interaction between a server and a client. The X Window System has a client-server architecture. A server is a process responsible for all input and output devices, and an application acts as a client [Young90].

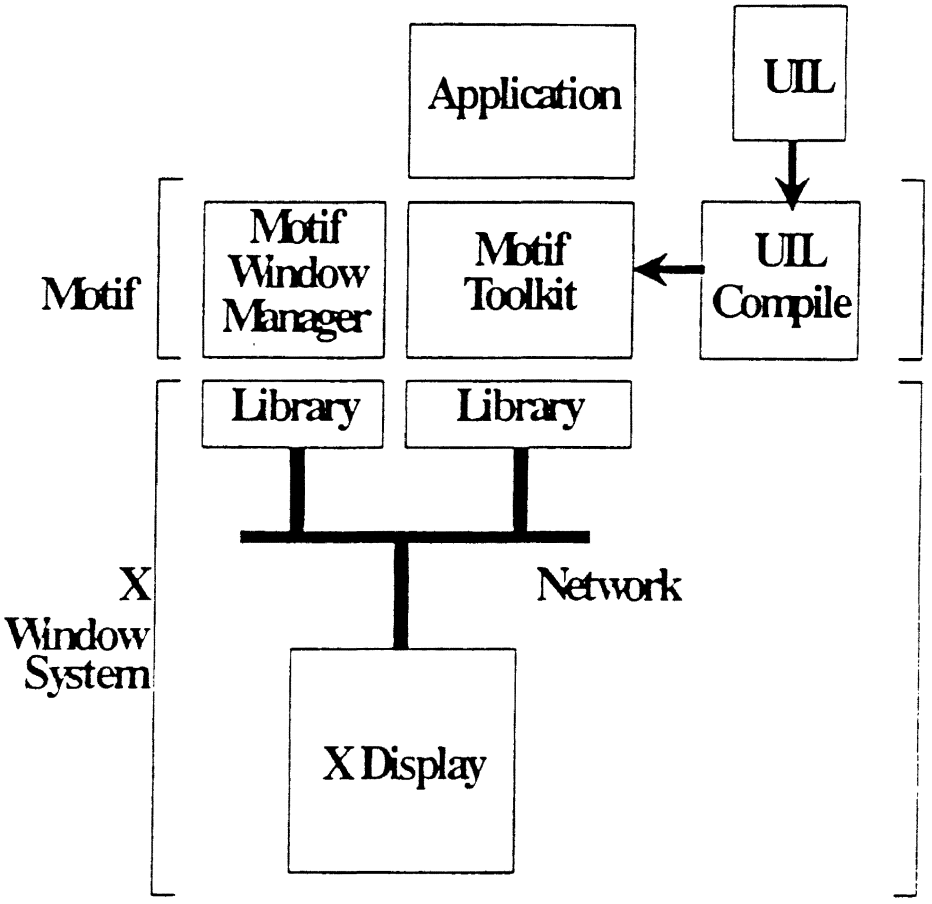
The X library provides the interface between an application and the X Window System. One such library is Xlib, a low level library of C routines that provides access to X graphics and interface functions [Johnson90] [Keller90] [Barkakati91]. Developing a GUI is usually done using toolkits which are easier to use than Xlib. The standard toolkit, known as the X Toolkit, consists of two parts, a layer known as Xt Intrinsics and a set of user-interface components known as widgets [Johnson90]. The Xt Intrinsics supports many different widget sets. One of these widget sets is the Motif Widget Set by the Open Software Foundation (OSF). The Motif widget set provides GUI components such as menus, buttons, and scroll bars.

3.1.3 OSF/Motif Toolkit

The OSF/Motif Toolkit, developed by members of the Open Software Foundation (OSF), is based on the X Toolkit Intrinsics (Xt). It is a set of functions and procedures that provides quick and easy access to the lower levels of the X Window System. The set

of functions and procedures provides user-interface objects called widgets. The Motif user-interface specification is completely implementation independent [Heller91].

The complete architecture of Motif consists of other components as illustrated in Figure 7 [Berlage91].



Legend:

UIL: User Interface Language.

Figure 7. Architecture of OSF/Motif (Source: [Berlage91])

The X Window System acts as the lower layer of the whole structure, which makes the system machine independent. The Motif window manager manages the basic functions of a window such as moving, resizing, and iconizing. The Motif window manager follows a set of standard conventions known as the Inter-Client Communication Conventions (ICCC), which enable it to manage X applications developed using different toolkits. It also manages the stacking order of the overlapping windows and controls the input focus determining which application window receives the input [Berlage91].

A User Interface Language (UIL) is provided to specify the presentation details of the user-interface objects. These specifications are translated by the UIL compiler and are read in at run time. Any Motif application can be designed without the use of UIL [Berlage91].

The most important component of the architecture of OSF/Motif is the Motif toolkit, which provides a set of widgets. It provides widgets for some common user-interface objects such as scroll bars, push buttons, labels, menus, dialog boxes, and text entry or display areas. In addition to these basic widgets, there are widgets called managers, which control the layout of other widgets. A widget operates, to a great extent, independently of the application. Xt dispatches events to a widget, which takes actions accordingly. For example, a label widget knows how to draw itself, how to highlight itself and how to respond to a mouse click (or a user-defined action) by calling an application function [Nye90] [Heller91].

Figure 8 shows the class inheritance hierarchy for the Motif widget set. Classes defined by Xt are also shown. Xt defines certain base classes of widgets, whose behavior

can be inherited by its subclass widgets. These base classes provide a common platform to all Xt-based widget classes.

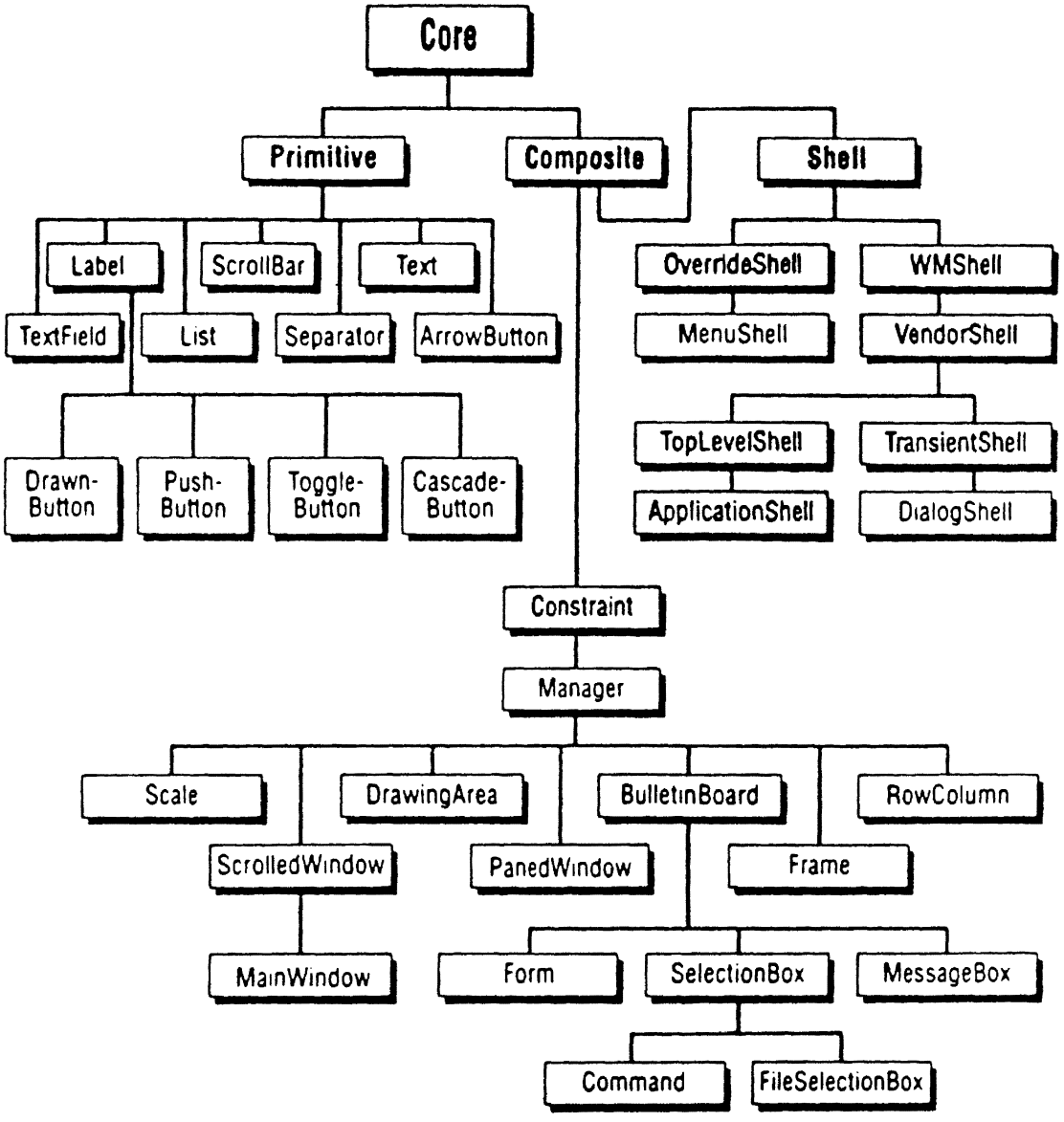


Figure 8. Class inheritance hierarchy for the Motif widget set (Source: [Nye90])

The Xt's Core widget class is the root of the hierarchy and serves as the supperclass for all widgets. It provides some common resources inherited by all widgets, such as position and size.

The Motif Primitive widget class is a subclass of Core. It inherits resources from Core and adds a few features of its own, such as the control of Motif 3-D shadows style. In the hierarchy of Primitive widgets, the Label widget class adds features of its own namely, the ability to display a string or a pixmap, and adds mechanisms for changing the font and placement of the string. All the subclasses of the Label widget class such as PushButton, inherit these added features along with others [Nye90].

The Xt's Composite widget class, which is a subclass of Core, introduces geometry-management capabilities. The Constraint class, also provided by Xt, is a refinement of the Composite class that provides the user or the application a way to manage the size and position of a widget. Motif also provides the Manager class, a subclass of the Constraint class, which acts as the supperclass of all Motif geometry-managing widget, such as RowColumn, ScrolledWindow, and BulletinBoard [Nye90].

The Xt's Shell widget class is a subclass of the Composite class. The Shell widgets provide an interface between other widgets and the window manager. The Shell widgets set the properties required by the window manager, and handle the window manager protocol for the application. As interacting with the window manager is very complex, there are quite a few Shell widget classes. The OverrideShell class is used to put up a temporary window that completely bypasses interaction with the window manager. Motif introduces the MenuShell class to service the special interface required by the Motif architecture. The WMShell widgets are simple, wire bed-frames that have

no special attributes. The VendorShell widget class is a subclass of WMShell so that vendors can define attributes that are specific to their own window managers. Dialog boxes are usually implemented in Xt using the TransientShell widget class. Window managers are not supposed to iconify the TransientShell widgets separately. If an application is iconified, all of its TransientShell widgets are iconified by the window manager. The Motif DialogShell is a subclass of TransientShell. The TopLevelShell and ApplicationShell widget classes are used by applications as their main top-level window [Nye90] [Young90] [Heller91].

3.2 Implementation

3.2.1 Program Structure

The software for the tool DrawPetri is divided into 19 modules (C programs). Every module includes sub-programs (functions) associated with a particular action needed during the invocation of DrawPetri. The first three modules are used to setup the initial environment while others are used to implement different options provided by DrawPetri. All functions necessary to implement an option are included in one module so that each option can be modified independently of the rest of the program. The modules are discussed below.

1. DrawPetri.c: This module is the main program file of DrawPetri. Various external functions are called to setup the initial user-interface of DrawPetri. To initialize the environment, functions are called from the program file init.c. To develop the user-interface, functions present in the program file user_interface.c are called.

2. init.c: This program file contains the functions required to initialize the environment which includes the initialization of different data structures and global parameters. It also creates different cursor shapes used by the application.

3. user_interface.c: This program includes functions which are called to establish the initial user-interface of DrawPetri. It creates pull-down menus, a display bar for the file name, a panel of action buttons, a display bar for the action selected and a scrollable (virtualized) drawing area. Initially, no action is selected and the lower display bar displays "NOTHING" for the prompt ACTION SELECTED. The default file name 'Untitled' is displayed on the display bar located at the top of the main window.

4. alloc_node.c: The functions included in this program file allocate a new node, either a place or a transition. It consists of two functions which provide the desired node, i.e., a place or a transition. Memory is allocated for the data structure PLACE or TRANSITION (whichever is needed), and the members of the data structure are initialized.

5. open_file.c: This program file contains the functions required to implement the Open and the New options, selectable from the pull-down menu File. Motif's FileSelectionDialogBox widget is used for the file selection. Functions needed to read an existing file are also included in open_file.c

6. close_file.c: This file consists of functions needed to implement the Quit option. One of its responsibilities is to provide a dialog box titled SHUT DOWN, when the Quit option is selected.

7. draw_object.c: The functions used to draw graphical objects, representing places, transitions, links, and labels are included in this program file. The Xlib functions

XDrawArc, XDrawRectangle, XDrawSegments, XDrawLine, and XDrawString are used in these functions.

8. link.c: The functions included in this file are related to the operation of drawing a link. When the LINK option is selected, one of its functions is called to pop up a dialog box, where the user is prompted to enter the link definition, i.e., a source and a destination for the link to be drawn. There are a few functions which check the validity of the link.

9. edit_label.c: The functions of this file implement the EditLabel option present in the pull-down menu Edit. A dialog box is presented by a function where a user enters an old label and a new label. There are functions to check the validity of this label change.

10. delete.c: This file consists of functions used to implement the DeleteObject option which can be selected from the pull-down menu Edit. There is a function which selects an object to be deleted, then another function is called to delete the selected object.

11. put_token.c: The functions of this file are responsible for the PUT_TOKEN option. The display of the dialog boxes and the other operations required for this option are managed by these functions.

12. spec.c: The functions included in this file implement the EDIT_SPEC and the SHOW_SPEC options. Dialog boxes are created and a user is provided an editable area to edit the interpretation of a node. One of the functions decides which node is selected by the user.

13. save.c: The functions used to implement the Save and the Save As options are included in this file. The function which actually writes the information to a file is also included here.

14. cursor.c: The functions present in the file cursor.c are used to create and assign different shapes to the cursor. These shapes are used on different occasions.

15. help.c: This file contains the help messages of DrawPetri.

16. utility.c: This file consists of different utilities used by DrawPetri.

17. undo.c: The functions involved in the implementation of the Undo option are included in this file. The Undo option is given in the pull-down menu Edit or it can be called by clicking the third button of the mouse while positioning the cursor inside the drawing area.

18. warn_msg.c: This file provides different warning messages that might be needed during the invocation of DrawPetri.

19. fire.c: The functions of this file are associated with the pull-down menu Firing. One of its function selects a transition to fire. If the selected transition is found to be enabled, the corresponding action is taken and the marking after firing is shown. Several functions are included to implement the options selectable from the pull-down menu Firing.

Each file, except warn_msg.c, has its own header file. The name of the header file is the same as the program file with an extension 'h', e.g., the header file for user_interface.c is user_interface.h. In the header file of a program file, the header files for X and Motif are included. Each header file also declares the external functions that

are called by the functions present in the corresponding program file. Each header file also declares the functions defined in the file.

The include-file called data_str.h is shared among the above-mentioned program files (except for help.c and warn_mesg.c). It contains the definitions of the different data structures and constants. The global variables are declared as external variables in the file global.h, which is included as header file in every file except DrawPetri.c.

The files mentioned above are all managed by a makefile named "Makefile". Each program file is compiled separately and then linked together.

3.2.2 The User Interface

The user-interface of DrawPetri is developed using different Motif widgets. The C program user_interface.c (see Section 3.2.1) contains all the necessary function calls to establish the initial user-interface of DrawPetri. The initial user-interface of DrawPetri is depicted in Figure 9.

The main interface of DrawPetri is divided into five regions. Each region with the constituent components is discussed below.

• Region 1 displays the name of the current file being edited. Whenever DrawPetri is started or a new file is opened, the string "Untitled" is displayed as the file name. During the process of editing a Petri Net structure, if a file is saved with another name, the new file name is displayed. If an existing file is opened, its complete path is not displayed. Only the character string after the last forward-slash (/) is taken as the file name, e.g., if

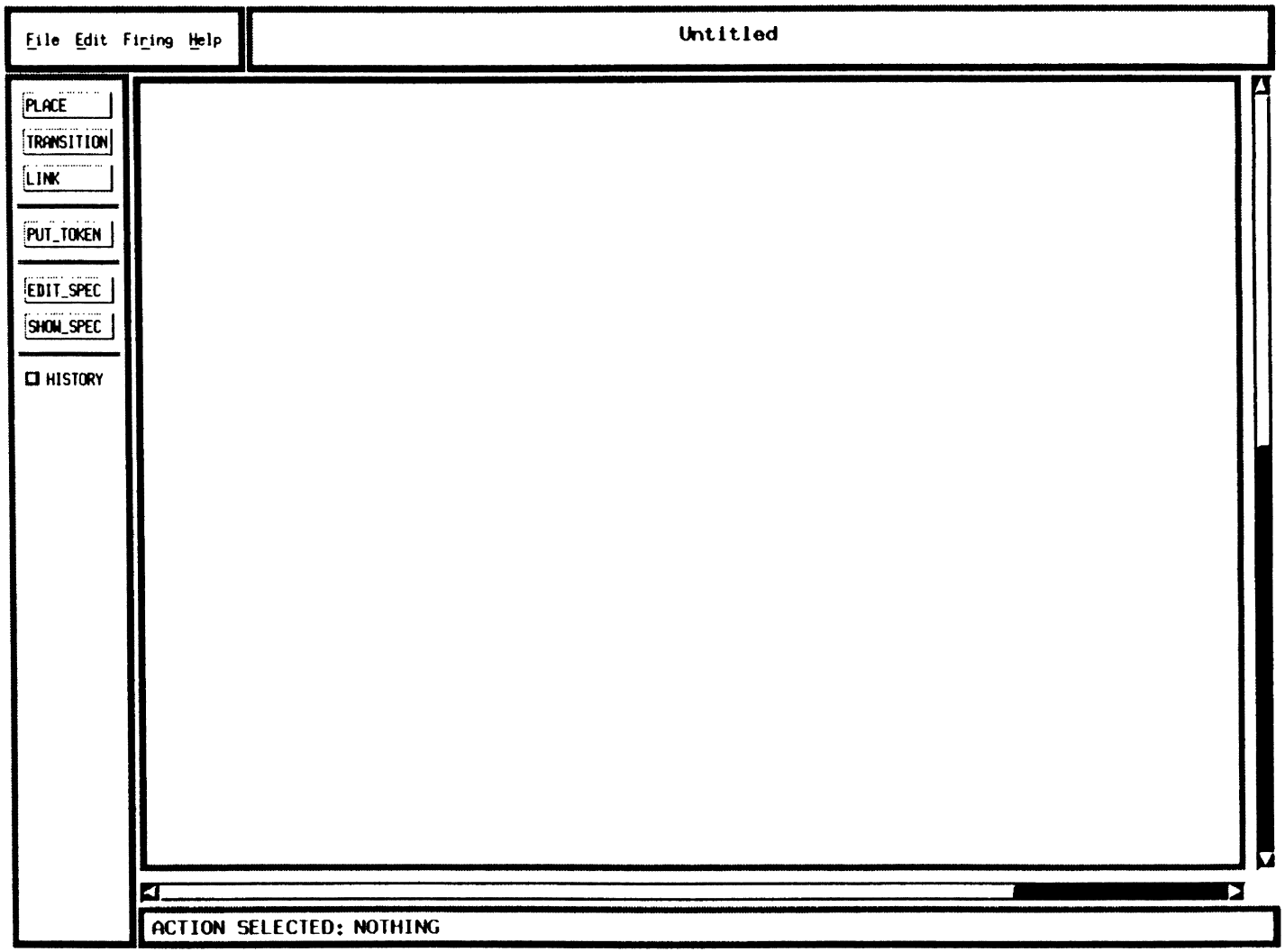


Figure 9. The five regions of DrawPerti

the file name, with a complete path, is "/z/hassanm/temp/petri_file1", then only "petri_file1" will be displayed.

• Region 2 is comprised of a scrollable (virtualized) drawing area. The drawing area is implemented by creating a Motif widget of the class XmDrawingAreaWidgetClass. It is created as a child of a ScrollWindow widget. The size of the drawing area is 1000x1200 (width x height) pixels. Within the drawing area, a user can draw a Petri Net structure of a system. The drawable objects consist of circles (places), arrowhead lines (links), rectangles (transitions), and strings (labels and tokens). The drawing area can be scrolled in both directions, horizontally and vertically, using the horizontal and vertical scroll bars.

• Region 3 is located at the bottom of region 2 (the drawing area). It displays the current action selected from the pull-down menu or from the options available on the left panel. If there is no active action, it displays the string "NOTHING".

• Region 4 consists of four pull-down menus: File, Edit, Firing, and Help. The File menu contains options: New, Open, Save, Save As, and Quit. The New option lets a user create a new file. If no drawing action is done in the current session, a file with a default name "Untitled" is opened and the drawing area is cleared. Otherwise, a dialog box is displayed prompting the user to save the current file. Three buttons: YES, NO, and CANCEL are provided. The YES button saves the current file and opens a new one. The NO button opens a new file without saving the current changes to the file. The CANCEL button cancels this option.

The Open option allows a user to open an existing file. When this option is chosen, a dialog box titled File Selection appears. This dialog box displays the directories and the files of the current directory in separate scrollable sections. Initially, this dialog

box displays the files in the current directory. A directory can be changed by selecting one of the directories from the given list or it can be changed using the Filter option of the dialog box. This option can also be used to display the file names satisfying some generic expression using wild cards (as in UNIX). To change the directory, a valid path in the Filter text box is needed, which should be followed by the choice of the Filter button. A file can be directly selected by typing a valid file name (the full path must be indicated unless the file is in the current directory) in the Selection text box. A file can be opened by selecting it from the list or through the Selection text box, and choosing the OK button. A desired file can also be opened through the Files list by double clicking on it. If the selected file is in the correct format, the file is opened and the corresponding Petri Net structure is displayed in the drawing area. Now the user can edit the file. However, if the selected file is not in the required format, a warning message box will pop up and the user must respond to it before taking any further action. After responding to the warning message box, a new selection can be made. When a file is successfully opened, the File Selection dialog box pops down (otherwise it will remain visible). The CANCEL button is provided to pop down the dialog box without opening a file.

The Save option stores the changes made to the current Petri Net structure without closing the file. If the user chooses the Save option when a file has not yet been named, a Save dialog box pops up. It prompts the user to enter a file name. Two warning message boxes may pop up, one for an invalid file name (no name) and the other for a file name that already exists. The CANCEL button is provided to cancel the Save operation if so desired.

The Save As option allows a user to name and save a copy of the current file with a new name. The current file name, displayed in a text box, can be browsed but cannot be edited. Another text box is provided for the entry of a new name, which should be followed by the choice of the SAVE button. The new name will be displayed and the old file will be closed. If the recent changes to the old file are not saved before saving as a new file, the changes will not appear in the old file but they will appear in the new file. If the old file has not yet been named, a dialog box pops up to prompt the user to save the current file. If a user decides to save the file, the Save dialog box pops up.

The Quit option ends the current session of DrawPetri. When this option is selected, a SHUT DOWN dialog box pops up. The user is prompted to save the current file. Three buttons: YES, NO, and CANCEL are provided. The YES button saves the file and ends the session (if a file name is given). If the file has not yet been named, the Save dialog box pops up (discussed previously). The NO button ends the session without saving the current changes to the file. The CANCEL button cancels this option.

The Edit menu provides three options: Undo, EditLabel, and DeleteObject. Undo reverses the last action (drawing action) that is performed on the drawing area. This action can be executed by clicking the third button of the mouse. The SteppedFire option, which is located in the Firing menu, is not an undoable option. Only drawing actions are undoable. For example, a most recently deleted object can be restored using Undo. After restoring an object, if Undo is selected again, it deletes the object.

Using the option EditLabel, a user can change the label of a node (a place or a transition). A dialog box titled EDIT LABEL pops up. A user is prompted for an old and a new label for a node. If both labels are valid (an existing old label and a non-existing

new label) and the DONE button is pressed, the change takes place. A warning message box may appear if an invalid label is given.

The option DeleteObject is used to delete an object. The object may be a place, a transition, or a link. When this option is selected, the cursor for the drawing area changes to a pirate sign (skull and cross-bones). To delete an object, the cursor must be placed over it and the first button of the mouse should be clicked.

The Firing menu has three options: SteppedFire, ViewHistoryFile, and ExecutionSequence. The options provided may be used to observe the dynamic behavior of a Petri Net model. The SteppedFire option is provided to select and fire a transition. The selected transition should be an enabled transition, otherwise a warning message box will pop up. An enabled transition is indicated by a filled rectangle. If the selected transition is enabled, it is fired. After each firing, the marking of the Petri Net is updated. The ViewHistoryFile option displays the history file, generated for the current session of firing. For each transition fired, the history file contains the label of the transition fired and the marking of the Petri Net (the token assignment). The ExecutionSequence option displays the history of the execution sequence, i.e., a list of the transitions fired. The last fifty transitions (if more than 50) are displayed.

The Help menu provides a quick help during the invocation of DrawPetri. A brief help is available on options PLACE, TRANSITION, LINK, PUT_TOKEN, EDIT_SPEC, SHOW_SPEC, Undo, DeleteObject, EditLabel, SteppedFire, ViewHistoryFile, and ExecutionSequence. More detailed help is available in the User Guide for DrawPetri (APPENDIX B).

• Region 5 is a panel of seven action buttons on the left side of the main window. These are the frequently-used options. The PLACE option can be selected by pointing and clicking on it. Upon selection of the PLACE option, the display area for the action selected displays DRAW PLACE. Now the user can draw a circle on the drawing area representing a place. The radius of the circle is fixed. When a user first clicks some place on the drawing area, a place in the form of a circle with a dotted-line boundary appears. This place can be moved to a desired location. A click of the second button of the mouse will make the place "permanent" (i.e., the boundary becomes a solid line). A default label is provided for each place. The labels can be edited using the EditLabel option (discussed previously).

The TRANSITION option is provided to draw a rectangle representing a transition. The width of a transition is fixed. A user can stretch a transition horizontally or vertically to a desired length. When a user, after selecting the TRANSITION option, clicks on the drawing area, that point on the drawing area is taken as one of the corners of the rectangle representing the transition. After stretching to the desired length, a click of the second button of the mouse will make the transition permanent. A default label is provided for each transition. The labels can be edited using the EditLabel option (discussed previously).

The LINK option is provided to draw a link between a place and a transition. When this option is selected, a dialog box titled LINK DEFINITION appears. The user is prompted to enter the "definition" of the link to be drawn. To "define" a link, the user should mention the source and the destination of the link. A valid label (i.e., an existing label) of a place or a transition should be given for the source and for the destination. If

an invalid label is entered, a warning message box pops up. It also warns if a link is defined for a same type node, which is against the definition of a Petri Net graph (the graph must be bipartite, as discussed in Chapter II). If a link is defined correctly, the dialog box pops down. Now the user should select the source on the drawing area and drag the mouse to the destination. As soon as it hits the destination, the link is established. The source is selected by clicking on its boundary. If a user clicks away from the boundary (inside or outside), a warning message box pops up. It displays the definition of the link being drawn and asks the user to re-select the source. After selecting the correct source, a dotted link becomes visible. The user can stretch it and also change its path by clicking the second button of the mouse. The drawing of a link can be cancelled by clicking the third button of the mouse before hitting the destination. If any other option is selected during the process of drawing a link, the process of drawing a link is canceled.

The PUT_TOKEN option allows a user to assign a number of tokens to a place. When this option is chosen, a dialog box titled PUT TOKEN pops up. A user is prompted for the label of the place where the tokens are to be assigned. Three buttons: INCREMENT, DECREMENT, and DONE are provided. The INCREMENT button increments the number of tokens by one and the DECREMENT button decrements the number of tokens by one. Initially, each place is assigned zero number of tokens. If the number of tokens at a place is a zero and a user clicks the DECREMENT button, a warning message box will pop up (as a negative number of tokens cannot be assigned). Other warning message boxes may also pop up for various reasons: for an invalid label

(no label), for a label which does not exist, or for a label which is a transition. To pop down the PUT TOKEN dialog box, the DONE button is provided.

The EDIT_SPEC and SHOW_SPEC options deal with the specification of a node. The specification of a node consists of the label of the node, the type of the node (place or transition), and the interpretation of the node. The EDIT_SPEC option lets the user type in the interpretation of the node. Using the SHOW_SPEC option, a user can only view the interpretation of a node. When a user selects either of these two options, the cursor definition for the drawing area changes to a hand pointing to an object. When a user clicks on a node whose specification is to be edited or viewed, a window titled EDIT SPECIFICATION or SHOW SPECIFICATION pops up, where the user can edit or view the specification of the selected node. For the EDIT SPECIFICATION window, two buttons, SAVE and CANCEL, are provided. The user can save the edited interpretation by clicking the SAVE button or can cancel it by clicking the CANCEL button. There is only one button, namely DONE, for the SHOW SPECIFICATION window. Clicking it will pop down the window.

The last action button is the HISTORY option. This is a toggle button (like a light switch). Every time it is clicked, it changes its state (ON/OFF). When this HISTORY switch is active (ON), a small square next to it becomes black; otherwise, it becomes white indicating that it is not active (OFF). Initially, it is in the OFF state. When it is turned ON, a history file is generated for a session of the firing of the Petri Net. For each transition fired, the history file contains the label of the transition fired and the marking of the Petri Net (i.e., its token assignment). When it is desired to generate a history file for a session of firing, a user should turn this switch ON. When this switch is turned ON,

a dialog box pops up. It prompts the user to enter a file name for the history file to be generated. After entering a valid file name (a non-existing name), the selection of the DONE button turns ON the HISTORY switch. The CANCEL button pops down the dialog box and the HISTORY switch is turned OFF. A warning message pops up for an invalid file name. If a valid file name is given and the DONE button is clicked, the HISTORY switch is turned ON. Now the user should select the SteppedFire option from the Firing menu to execute (to fire the transitions of) the Petri Net. The history file is updated for each transition fired. If any option other than from the menus Firing and Help is chosen, the history switch is turned OFF (i.e., the firing session ends). The history file generated for a session of firing can be viewed using the ViewHistoryFile option present in the Firing menu.

3.2.3 Other Implementation Details

In this subsection, a brief implementation detail regarding the drawing of a link with an arrowhead is mentioned, plus a few known limitations of DrawPetri.

Initially, drawing a link with an arrowhead proved to be rather difficult. This problem was posted on the internet news group "comp.windows.x" and some help was received from Mr. Paul Brown (e-mail address: brown@zen.wes.army.mil). After necessary modifications the code was used for drawing a link with an arrowhead. The section of the code sent by Mr. Paul Brown is given in Appendix D, located inside the C program draw_object.c. The logic is explained below.

Let the line for which an arrowhead is needed have coordinates (x1, y1) and (x2, y2). Two small line segments *a* and *b* are drawn from the point (x2, y2), to make the

arrowhead. The line segment *a* is drawn between coordinates (x1a, y1a) and (x2a, y2a), and the line segment *b* is drawn between (x1b, y1b) and (x2b, y2b).

The values of these coordinates are given below,

$$
\begin{aligned}
x1a &= x2 \\
y1a &= y2 \\
x2a &= length * \cos(\alpha - \beta) \\
y2a &= length * \sin(\alpha - \beta) \\
x1b &= x2 \\
y1b &= y2 \\
x2b &= length * \cos(\alpha - \beta) \\
y2b &= length * \sin(\alpha - \beta)
\end{aligned}
$$

The constant "length" is the length of each line segment. The value used for length is 15. Other values are, $\alpha = \tan^{-1}((y2 - y1)/(x2 - x1))$ and $\beta = 20.0 * k$, where k is a constant whose value is 0.01745329. The quantities k, length, α, and β are declared as double integer. The value of the expression $\tan^{-1}((y2 - y1)/(x2 - x1))$ is calculated using the C math function atan2(). Finally, the three line segments are drawn using the Xlib function XDrawSegments(), which makes the required line with an arrowhead.

One of the limitations of DrawPetri is that the total number of objects that can be drawn in the drawing area is limited by an array size. The size of this array is currently 1000, which can be changed if desired. Also, the size of the drawing area is limied, so a system that needs a larger area cannot be designed on one screen. A large Petri Net should be broken down into subsystems, so that it can fit in the given drawing area. However, the default size of the drawing area, which is 1000x1200 (width x height) pixels, can be changed.

CHAPTER IV

EVALUATION OF THE TOOL

In this chapter, the evaluation of the tool DrawPetri is mentioned together with some observations and possible improvements based on the evaluation. DrawPetri was prototypically evaluated by giving the graduate students of the Operating Systems II course offered by the Computer Science Department of Oklahoma State University at Stillwater, Oklahoma, assignments to draw the Petri Net model of an existing simulation package named the Unified Simulation Environment (USE). The USE system is used in the prototyping and evaluation of architectures and operating systems [Hassan92] [Jhun92] [Daily93]. Approximately 35 graduate students used DrawPetri and some useful feedback was received. A number of smaller subsystems were also modeled by DrawPetri, as explained in the next section.

4.1 Sample Systems Modeled by the Tool

One of the systems modeled by DrawPetri was the USE system (mentioned above). Figure 10 shows the design of part of the USE system. This system is modeled and its dynamic behavior is observed. The execution of the resulting Petri Net structure of the system was close to the actual expected behavior of the system.

Other subsystems were also modeled using DrawPetri and their dynamic behavior was observed. The Petri Net shown in Figure 11 represents a model of a readers-writers

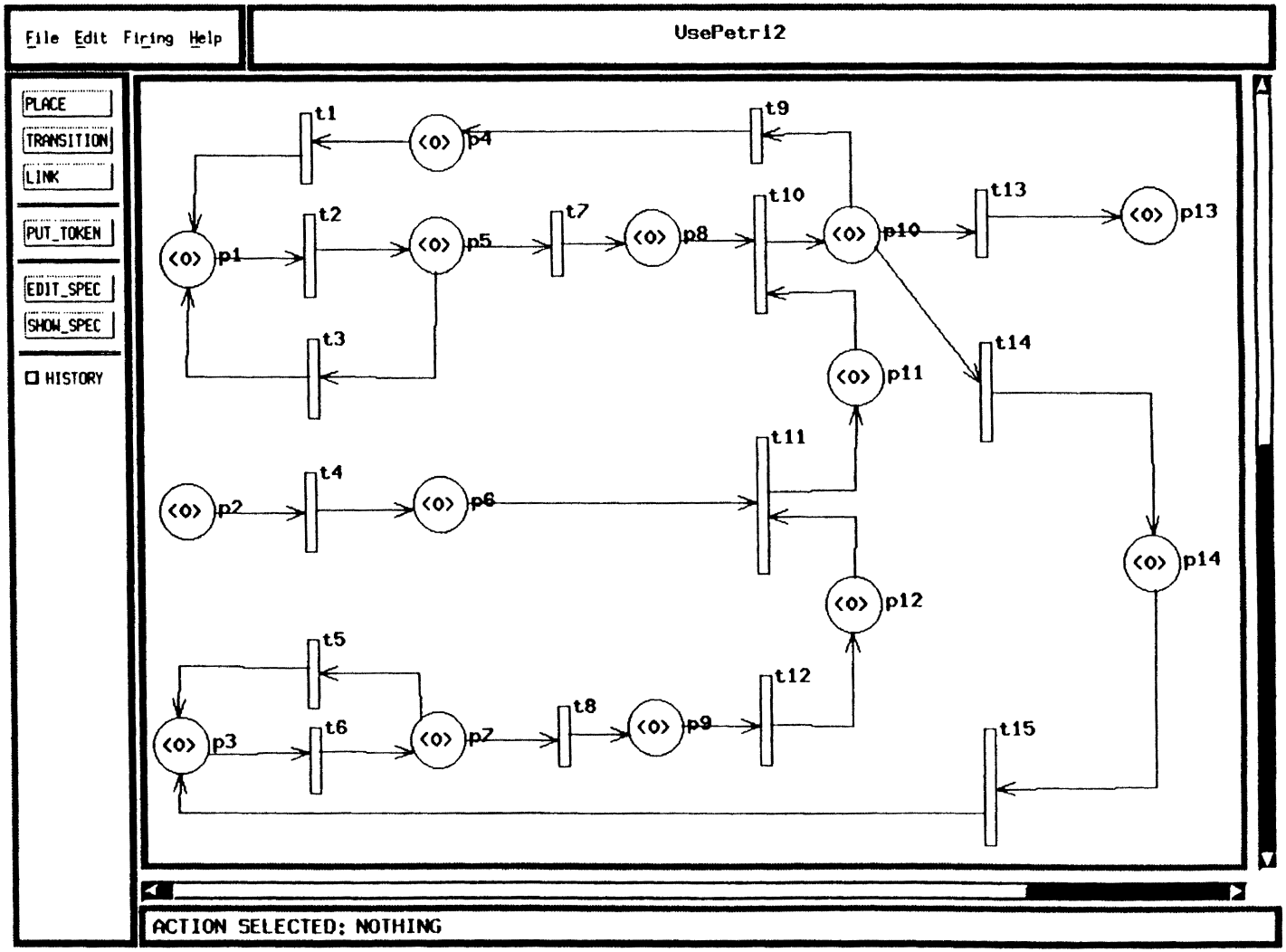


Figure 10. A Petri Net model of the USE system

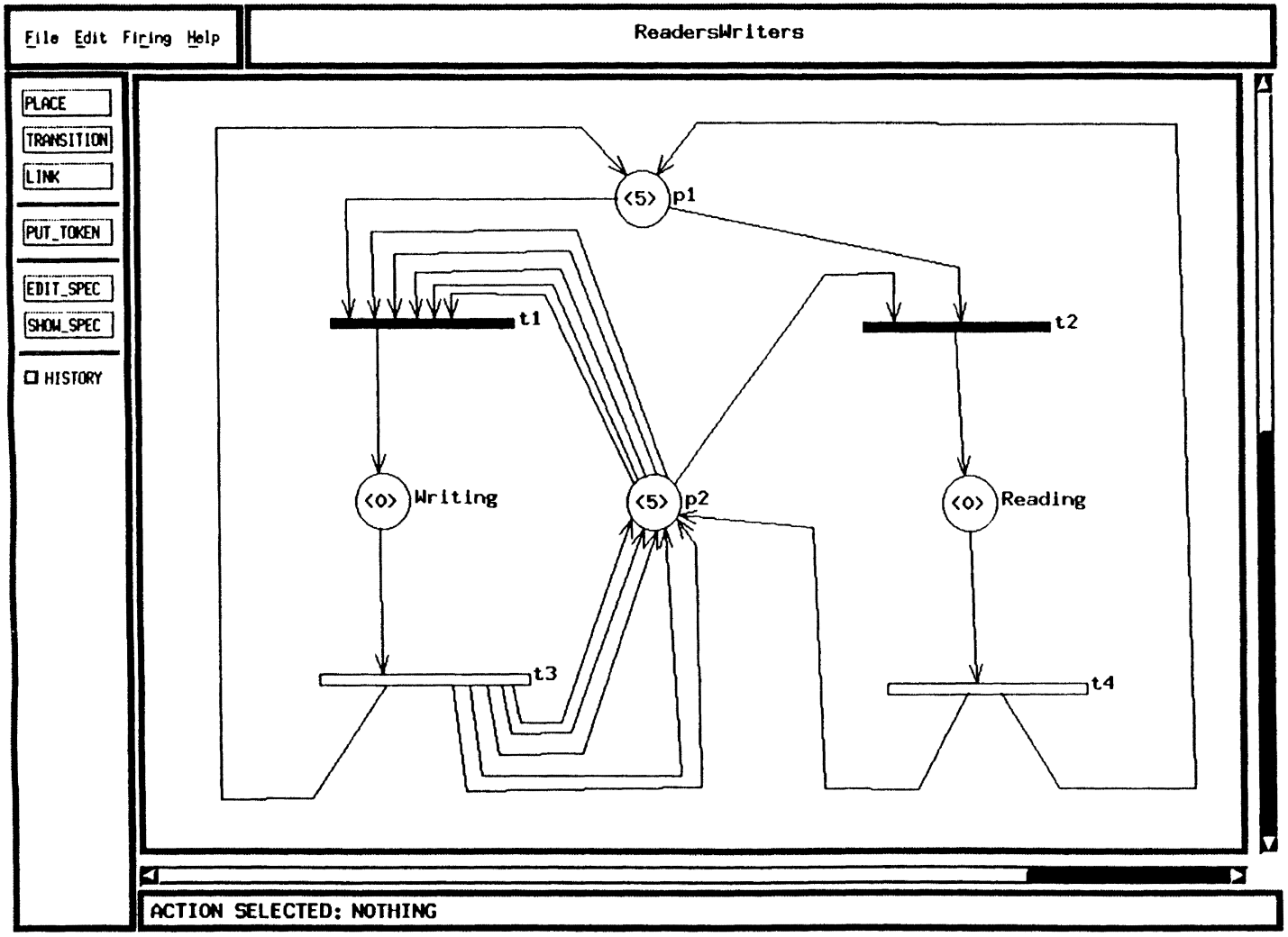


Figure 11. A Petri Net model of a readers-writers system

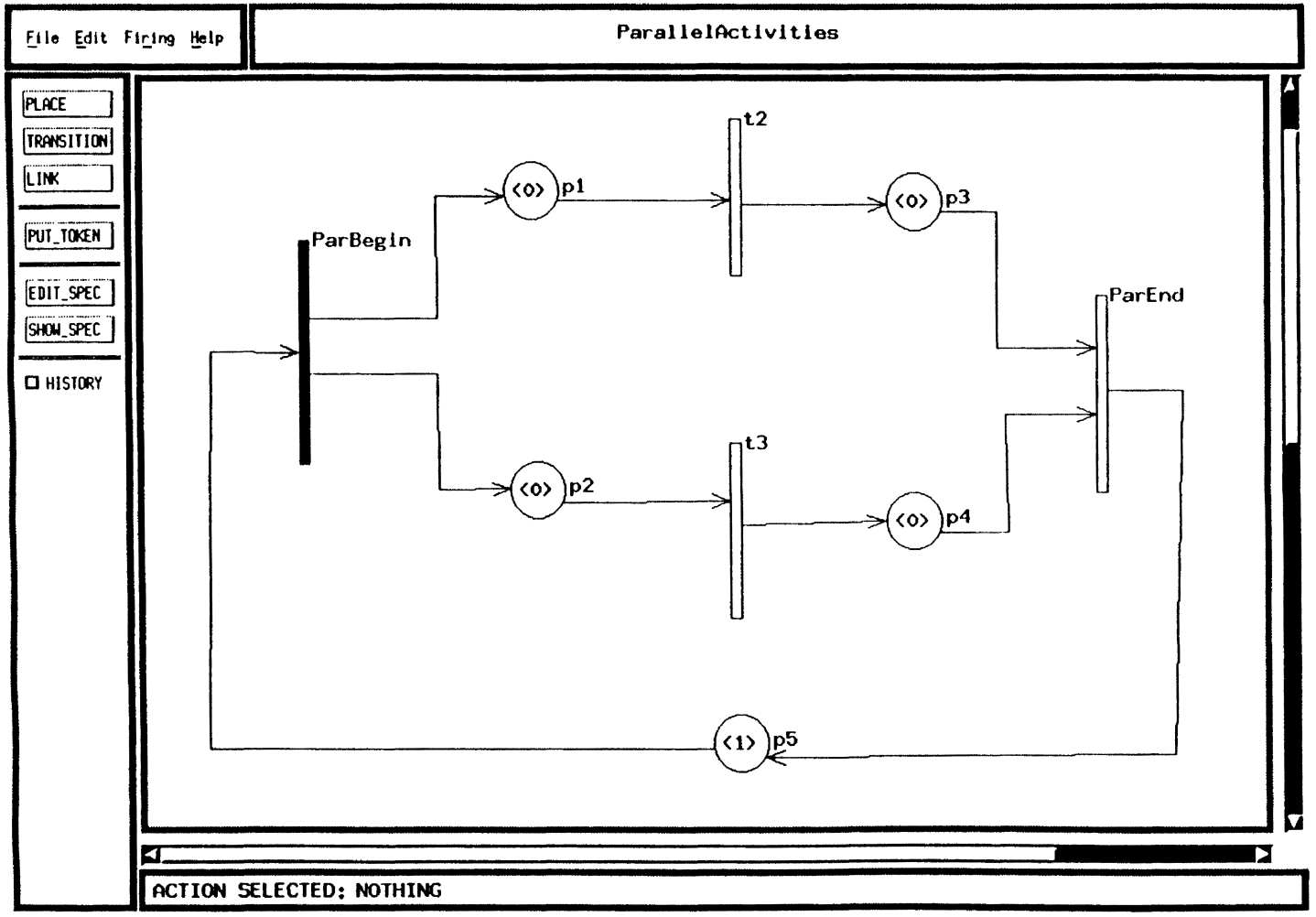


Figure 12. A Petri Net model of parallel activities

synchronization [Murata89], where the 5 tokens at place p1 represent 5 processes that may read and write in a shared memory represented by p2. Up to 5 processes may be reading concurrently, but when one process is writing, no other process can be reading or writing. It is easily verified that up to 5 tokens (processes) may be in the place labeled Reading if no token is in the place labeled Writing, and that only one token (process) can be in the place labeled Writing since all 5 tokens in place p2 will be removed when t1 fires once.

The Petri Net shown in Figure 12 represents parallel activities [Murata89]. The parallel or concurrent activities represented by transitions t2 and t3 begin at the firing of the transition labeled ParBegin and end with the firing of the transition labeled ParEnd.

4.2 Observations

From the feedback obtained from the users of DrawPetri, several useful observations were made. Based on those observations, a number of changes were made to the initial implementation. One of the important changes made was the method of drawing links. Initially, only straight links (one-segment lines) could be made, i.e., a line was drawn between two pairs of coordinates (x1,y1) and (x2, y2). The problem with this approach was that sometimes the user had to pass the link through some objects that came in its path. There was no way to change the direction, thus bypassing an object, and continue making the link. The algorithm for drawing a link was changed so that a link may have as many segments as needed. The course of the path of a link can be changed by clicking the second button of the mouse. As soon as the destination node is hit, a link

is established. If a link consists of several segments, only the segment hitting the destination node will have an arrowhead.

Another observation made was that the Undo option was heavily used during the process of editing a Petri Net structure. To undo the last action, the user had to go to the pull-down menu Edit every time and select the Undo option from there. To ease the process of editing, the Undo action was associated with the third button of the mouse as well as the pull-down menu and the button selection. Now a user can undo the last action by clicking the third button of the mouse, while keeping the cursor inside the drawing area.

CHAPTER V

SUMMARY AND FUTURE WORK

5.1 Summary

In Chapter I, the significance of modeling and the main objective of this thesis was stated. Chapter II presented an introduction to Petri Nets. The topics covered in this chapter consisted of the basic definitions to understand Petri Nets; some of the important properties of Petri Nets such as reachability, safeness and boundedness, liveness, and coverability; modeling with Petri Nets; and analysis of Petri Nets. Chapter III discussed the implementation issues of the DrawPetri tool. Section 1 of Chapter III addressed the implementation platform and the run-time environment, including an introduction to the Sequent Symmetry S/81, the X Window System, and the OSF/Motif Toolkit. Section 2 of Chapter III explained the program structure of the tool, the user-interface, and other implementation details. Chapter IV outlined the pilot evaluation of the tool.

The main objective of this thesis was to develop a graphical tool that can help in developing a Petri Net model of a given system and to animate the execution of the resulting Petri Net (i.e., to provide the before-firing and after-firing snapshots of the corresponding Petri Net). This tool can be used to design a system or part of system that can fit in the drawing area of the tool. If a big and complex system is desired to be designed, it can be broken down into well-defined subsystems, each of which can be

modeled on the given drawing area. The user-interface components are implemented using OSF/Motif widgets (discussed in Chapter III). This tool was used by a number of users to design reasonably complex systems. The graduate students of the Operating System II course, offered by the Computer Science Department, Oklahoma State University, so far have been the main users of this tool (see Chapter IV). In general, the users were able to model a fairly complex system using this tool. Their feedback was used to improve the tool.

5.2 Future Work

The future versions of this tool should incorporate one or more of improvements mentioned below.

A nested Petri Net model can be implemented, so that the limitation of drawing a Petri Net one screen at a time can be removed. A Block and Move option can be implemented for obtaining less cluttered models. Analysis of a Petri Net model can be introduced in the form of some options.

Finally, as it was originally intended, the DrawPetri tool can be incorporated into the existing simulation package called the USE system (used for the prototyping and evaluation of architectures and operating systems) [Jhun92] [Hassan92] [Daily93] not only to model different systems but also to use the resulting Petri Net as an executable specification of a desired system.

REFERENCES

[Agerwala79] T. Agerwala, "Putting Petri Nets to Work", *IEEE Computer*, vol. 9, no.12, pp. 85-94, December 1979.

[Barkakati91] N. Barkakati, *X Window System Programming*, Macmillan Computer Publishing, Carmel, IN, 1991.

[Berlage91] Thomas Berlage, *OSF/Motif: Concepts and Programming*, Addison-Wesley Publishing Company, Reading, MA, 1991.

[Chandrashekar91] S. Chandrashekar, B. E. Mayfield, and Mansur H. Samadzadeh, "A Project Engineering Tool to Assist in the Development and Maintenance of Project Life Cycles", *Proceedings of the ACM/IEEE-CS 1991 Symposium on Applied Computing*, Edited by: V. Kumar and E. A. Unger, Kansas City, MO, pp. 119-122, April 1991.

[Chandrashekar93] S. Chandrashekar, B. E. Mayfield, and Mansur H. Samadzadeh, "Towards Automating Software Project Management", accepted for publication in *The International Journal of Project Management*, vol. 11, no. 1, pp. 29-38, February 1993.

[Commoner72] F. Commoner, *Deadlocks in Petri Nets*, Report CA-7206-2311, Massachussetts Computer Associates, Wakefield, MA, 50 pages, June 1972.

[Daily93] S. R. Daily and Mansur H. Samadzadeh, "Object-Oriented Simulation of Capability Based Architectures", *The Twenty Sixth Annual Simulation Symposium*, Sponsored by SCS, IEEE-CS, and ACM, in conjunction with *The 1993 Simulation Multi-Conference*, pp. 258-266, Washington D.C., March 29-April 1, 1993.

[Deo74] N. Deo, *Graph Theory With Applications to Engineering and Computer Science*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1974.

[Hassan92] Khaled M. Hassan and Mansur H. Samadzadeh, "An Object-Oriented Environment for Simulation and Evaluation of Architectures," *Proceedings of IEEE 25th Annual Simulation Symposium* in conjunction with *The 1992 SCS Simulation Multiconference*, Orlando, FL, pp. 91-97, April 1992.

[Heller91] Dan Heller, *Motif Programming Manual*, O'Reilly & Associates, Inc., Sebastopol, CA, 1991.

[Jhun92] Ik-Jeong Jhun, Khaled M. Hassan, and Mansur H. Samadzadeh, "Simulation of a Computing Environment Using Stochastic Processes and the Object-Oriented Technology," *Proceedings of the Twenty-Third Annual Pittsburgh Conference on Modeling and Simulation*, Volume 23, Part 3, Edited by: William G. Vogt and Marlin H. Mickle, Pittsburgh, PA, pp. 1579-1585, April 30-May 1, 1992.

[Johnson90] E. F. Johnson and K. Reichard, *Advanced X Window Application Programming*, Advanced Computer Books, Management Information Source, Inc., Portland, OR 1990.

[Johnsonbaugh89] R. Johnsonbaugh and T. Murata, "Petri Nets and Marked Graphs: Mathematical Models of Concurrent Computation", *The American Mathematical Monthly*, vol. 89, no. 8, pp. 552-566, October 1989.

[Keller90] B. J. Keller, *A Practical Guide to X Window Programming*, CRC Press Inc., 2000 Corporate Blvd., N.W., Boca Raton, FL, 1990.

[Murata80] T. Murata and J. Y. Koh, "Reduction and Expansion of Live and Safe Marked Graphs", *IEEE Trans. Circuit Syst.*, vol. CAS-27, no. 1, pp. 68-70, January 1980.

[Murata89] T. Murata, "Petri Nets: Properties, Analysis, and Applications", *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541-580, 1989.

[Nye90] Andrian Nye, *X Protocol Reference Manual for Version 11 of the X Window System*, O'Reilly & Associates, Inc., Sebastopol, CA, 1990.

[Peterson77] J. L. Peterson, "Petri Nets", *ACM Computing Surveys*, vol. 9, no. 3, pp. 223-252, September 1977.

[Peterson81] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1981.

[Sequent90] *DYNIX/ptx User's Guide*, Sequent Computer, Inc., 1990.

[Young90] D. A. Young, *The X Window System: Programming and Applications with Xt, OSF/Motif Edition*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1990.

APPENDICES

APPENDIX A

GLOSSARY AND TRADEMARK INFORMATION

accelerator: A key combination used to invoke a menu entry quickly without displaying that menu.

application window: The window where an application resides with its complete user interface.

background: The flat area on which a widget resides.

background color: The color from which all widgets generate their top and bottom shadows and their select color, and against which labels and bitmaps are created with the foreground color.

bipartite graph: A graph G is called bipartite if its vertex set V can be decomposed into two disjoint subsets V_1 and V_2 such that no edge is allowed between pairs of vertices in the same subset.

bitmap: An image created by using only two colors of the screen.

button: Either a physical button on the mouse or a widget that simulates a real button on the screen.

callback: A mechanism invoking a procedure of an application by a widget under specific conditions.

class: Common description for a set of similar objects with the same structures but different attribute values.

class hierarchy: Logical ordering of classes, in which a subclass is the specialization of its superclass. Subclasses may inherit, change, or add features.

click: Pressing and releasing a mouse button without intervening pointer motion.

client: Any widget or a set of widgets that is managed to be displayed in a window.

client-server model: In client-server model, a process known as a server is responsible for providing some facilities to other processes known as clients. In the X Window System, a server (X server) is responsible for all input and output devices and an application, that uses the facilities provided by X server, is referred to as a client.

composite widget: A widget that may have other widgets as children and control their geometries.

control vector: For a Petri Net with n transitions, the control vector U_k is an n by 1 column vector of (n - 1) 0's and one non-zero entry. The 1 is in the jth position indicating that transition t_j fires at the kth firing.

dialog box: A window that is popped up by the application when detailed information needs displaying or needs to be acquired.

enabled transition: A transition t is said to be enabled if every place present in its preset has at least one token.

event: Message sent by the X server to an application.

event handler: A procedure to reposnd to one or more types of events for a specific widget.

execution of a Petri Net: Firing of a transition changes the marking of a Petri Net. The successive firing of transitions and the resulting movements of tokens is referred to as the execution of a Petri Net.

firing: Firing of a transition means that it removes a token from each place present in its preset and puts a token in each place present in its postset.

fundamental circuit matrix: A matrix in which all rows correspond to a set of fundamental circuits.

geometry management: The procedure of automatically negotiating size and location of widgets in a hierarchy.

graphics context(GC): Various information for graphics output such as foreground pixel, background pixel, line width, clipping region, and so on. A graphics context can only be used with drawables that have the same root and the same depth as the graphics context.

graphical user interface (GUI): A visual representation of a computer's functions that can be manipulated by nonprogrammatic means.

icon: A small symbol that represents an object or action that can be selected or performed by selecting that icon.

identity matrix: The matrix having 1's along its main diagonal and 0's for all other entries.

incidence matrix: For a given marked Petri Net, $PN_m = (P, T, A, M_0)$, with m places and n transitions, the incidence matrix $A = [a_{ij}]$ is an n by m matrix of integers, such that $a_{ij} = a_{ij+} - a_{ij-}$, where a_{ij+} is the number of arcs from transition t_i to place p_j and a_{ij-} is the number of arcs from p_j to t_i.

inheritence: A mechanism to make use of the functionality of a superclass in a subclass without duplicating it.

indegree: Indegree of a node is the number of arcs terminating in that node.

inter-client communication conventions (ICCC): A set of standard rules that control interaction among clients and between a client and the window manager.

intrinsics: Basic library underlying Motif that implements fundamental mechanisms for setting up widget classes.

marking: A marking of a Petri Net gives a mapping of places to non-negative integers. Each integer represents the number of token(s) present in the corresponding place.

mnemonic: Character abbreviation used for a menu command.

MULTIBUS: An industry-standard bus that can be used to connect a variety of peripheral devices to an S/81 system.

non-singular matrix: A square matrix A is said to be non-singular if its determinant is non-zero.

outdegree: Outdegree of a node is the number of arcs emanating from that node.

pixel: A single identifiable spot on the screen or in a pixmap. A pixel may have a number of different color values (black and white on a monochrome screen).

pixel values: A pixel is an N-bit value, where N is the number of bit planes used in a particular window or pixmap; that is, N is the depth of the window or pixmap. For a window, a pixel value indexes a colormap to derive an actual color to be visible.

pixmap: A three-dimensional array of bits. A pixmap is usually considered as a two-dimensional array of pixels, where each pixel can be a value from 0 to 2^{N-1} and where N is the depth (z axis) of the pixmap. A pixmap can also be considered as a stack of N bitmaps.

pointer: An alternate term used to describe a mouse cursor.

place: A place in a Petri Net graph is a node represented by a circle.

postset of a place: Postset of a place p is a set of transitions such that there is an arc from place p to each transition of the set.

postset of a transition: Postset of a transition t is a set of places such that there is an arc from transition t to each place of the set.

preset of a place: The set of transitions incident on a place p is called the preset of place p.

preset of a transition: The set of places incident on a transition t is called the preset of transition t.

protocol: A mutual agreement between a client and a server to accomplish certain actions.

reachability set: The set of all reachable markings is called the reachability set.

reachability tree: The reachability tree represents the reachability set of a Petri Net (it can be an infinite tree).

server: The server offers the basic windowing mechanism. It handles inter-process communication (IPC) connections from clients, demultiplexes graphics requests onto the screens, and multiplexes input back to the appropriate clients.

singular matrix: A square matrix A is said to be singular if its determinant is equal to zero.

token: For a marked Petri Net, each place is mapped to a non-negative integer i, which is interpreted as i token(s) being present at that place.

toolkit: Objects and functions available to an application programmer.

transition: A transition in a Petri Net graph is a node represented by a bar. A transition is an abstraction of an action.

transpose matrix: Transpose of a matrix A, of order m by n, is a matrix, of order n by m, obtained by interchanging the rows and columns of matrix A. It is usually denoted by A^T.

widget: Basic interface object of the X toolkit intrinsics associated with an X window with encapsulated functionality.

window: A rectangular area on a display screen that is associated with a particular program.

window manager: The underlying application that makes it possible for windows to be displayed and directly manipulated on the screen.

X: A portable, networked, and transparent window system.

X client: Application process that uses the services of the X server for input and output.

Xm: The prefix Xm is placed before any value that is specified as a widget resource. A convention that identifies the X window system and Motif values as different from other values used in source code.

XmN: The prefix XmN is placed before any resource classification that asks for a specified value.

X protocol: Protocol by which X clients and X server communicate.

X server: Process that exclusively controls the display hardware and accepts requests from clients.

X toolkit intrinsics: A library of utility functions and data structures layered between Xlib and the widget set.

X Window System: Hardware-independent and network-transparent base layer that provides services to graphical user interfaces.

Xt intrinsics: See X toolkit intrinsics.

TRADEMARK INFORMATION

APPENDIX B

USER GUIDE FOR DrawPetri

1. Introduction

The Petri Net tool DrawPetri is available on Sequent Symmetry S/81 system located at the Computer Science Department of Oklahoma State University, Stillwater, Oklahoma.

The tool DrawPetri provides a graphical environment to model a system using Petri Net as a modeling tool. The user interface of DrawPetri is developed using the OSF/Motif widget set. The initial screen of DrawPetri contains various available functions using four pull-down menus and seven action buttons (Figure 13). At the top of the main window, the name of the current file being edited is displayed. The display bar at the bottom of the main window shows the current action selected. To draw a Petri Net model, a scrollable (virtualized) drawing area is provided. Descriptions of all available actions is explained in the next section.

2. Using Menus

The DrawPetri menu system consists of the following parts: menu bar, menu pads, menus, and menu options. The menu system allows a user to communicate with DrawPetri. Each part of the menu system is described below. The initial screen of DrawPetri is illustrated in Figure 13.

Menu Bar: The *menu bar* is located at the top left of the main window. It displays titles for menus.

Menu Pads: The titles on the menu bar are called *menu pads*. They display the names of the menus. A user can use the mouse to display the menu associated with each menu pad.

Menus: When a user chooses a menu pad from the menu bar, DrawPetri displays a menu. A *menu* (also called a pop-up menu) is a list of options. When a user chooses

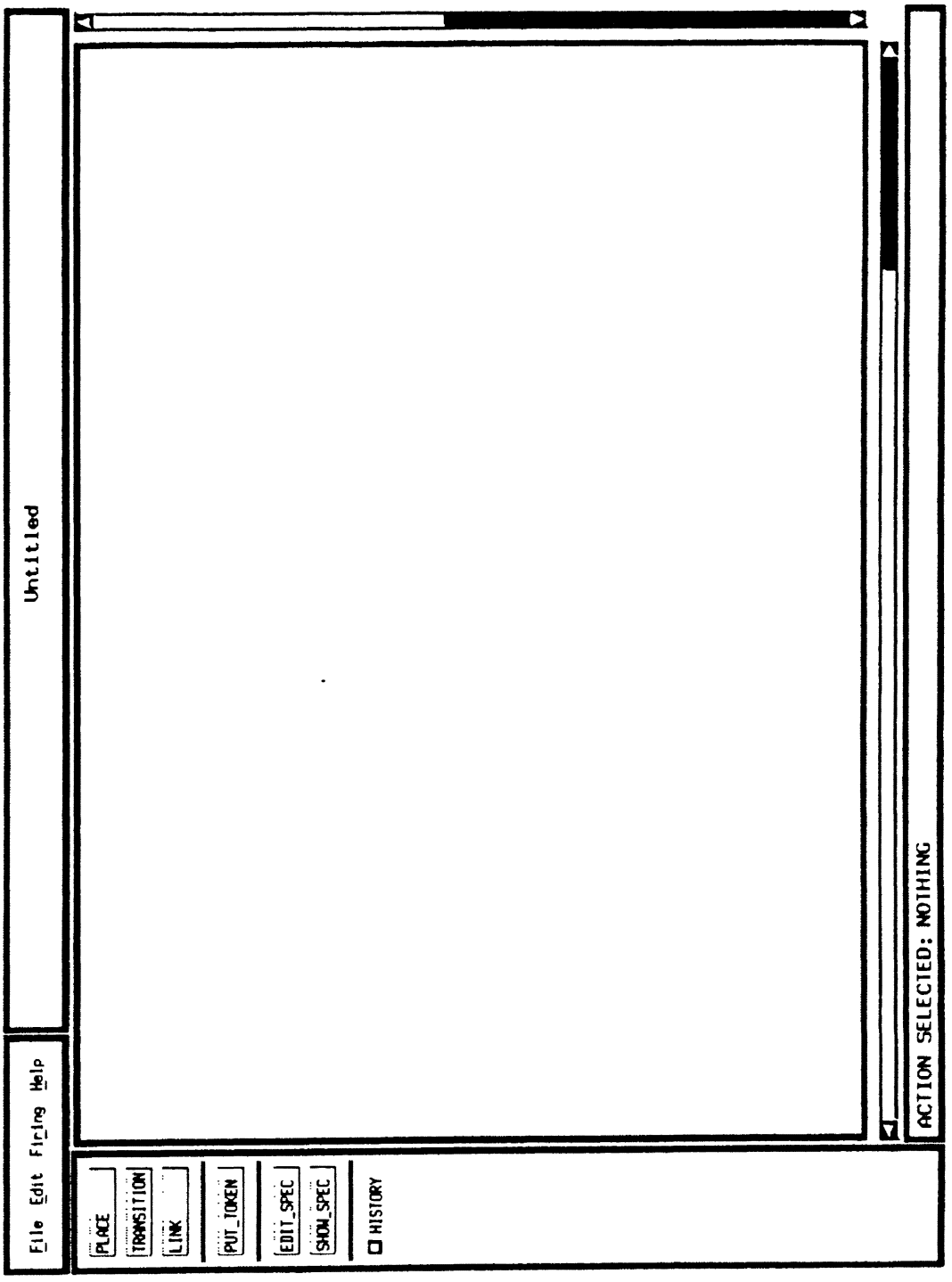


Figure 13. Initial Screen of DrawPetri

an option from a menu, the user is telling DrawPetri what action to take. "Choose" means to activate a selection (highlighted option) by clicking with the mouse.

Menu Options: Each menu contains options. The options on each menu are logically related to the menu pad. On a single menu, options can be further grouped and separated by horizontal lines. When a menu option is chosen, an action occurs: A window may open or close, an action may be selected, a dialog box may appear, or a command may be executed.

Certain menu options are followed by an ellipsis "...". An ellipsis means that more information will be needed to complete the action. With each selection of this type of option, a dialog box appears to request the additional information needed.

Some menu options have a control-key shortcut listed next to them on the menu. A user can use the control-key combination as a shortcut to choose a menu option without displaying the menu.

To choose a menu option with the mouse, point to the menu pad and hold down the mouse button. The menu appears. While still holding down the mouse button, drag until the appropriate option is selected. Then release the mouse button to choose the option. The menu disappears.

3. Using Dialog Boxes

When an option that is followed by an ellipsis is chosen, the box that appears is called a *dialog box*. Dialog boxes also appear at other times during a DrawPetri session when more information is needed to complete an action. A dialog requires a user to provide additional information before an action can be taken.

Once a dialog box is displayed, a user must exit it before being able to use any other feature of DrawPetri. To exit a dialog box, choose the appropriate push button. If a dialog box is displayed and a user tries to perform an action outside of it, the computer beeps. A particular dialog box may contain different objects such as push buttons, text box, and list. These objects are defined below.

Push Button: A *push button* is a small rectangle containing the key words that describe the action it triggers. The action associated with a push button occurs immediately when it is clicked. Sometimes another dialog box appears as an action.

Text Box: This rectangular box indicates an editable text region where text may be entered. To enter text in a text box, click on the text box to position the cursor, then type and edit as usual. DrawPetri also has descriptive text regions, where text is displayed but cannot be edited. Some descriptive text can be browsed by the user, e.g., the text box for current file name in the "Save As" dialog box.

List: This box contains a list of items such as directories, files, and fields that can be selected. If necessary, click on the arrow at either end of the scroll bar to move through a list. Click on a desired item to select it.

Moving within a dialog box using the mouse: A user can move within a dialog box using the mouse by positioning the pointer, selecting, and clicking to choose a push button or a text editing region.

Moving within a dialog box using the keyboard: In a dialog box, the tab key may be used to move from control to control.

Repositioning a dialog box on the screen: A dialog box can be repositioned on the screen. To move a dialog box using the mouse, point to the title bar, hold down the mouse button, and drag until the dialog box is in the desired location and then release the mouse button.

4. File Menu

The File menu contains options that allow a user to create, open, and save a file. A user can also exit DrawPetri through this menu.

The New option lets a user create a new file. If no drawing action is done in the current session of DrawPetri, a file with a default name "Untitled" is opened and the drawing area is cleared. Otherwise, a dialog box is displayed prompting the user to save the current file. Three buttons: YES, NO, and CANCEL are provided. The YES button saves the current file and opens a new one. The NO button opens a new file without saving the current changes to the file. The CANCEL button cancels this option.

The Open option allows a user to open an existing file. When this option is chosen, a dialog box titled File Selection appears. This dialog box displays all the files in a directory. The first time that this dialog box appears during a session, it displays the files in the default directory. The directory can be changed using the Filter option of the dialog box. This option can also be used to display the file names satisfying some generic expression using wild cards (as in UNIX). To change the directory, type a valid path in the Filter text box and choose the Filter button. A file can be directly selected by typing a valid file name (indicate the full path, if not in the current directory) in the Selection text box. To open a file, select it from the list or through the Selection text box, and choose the OK button. A desired file can also be opened through the Files list by double clicking on it. If the selected file is in the correct format, the file is opened and the corresponding Petri Net structure is displayed in the drawing area. Now the user can edit the file. However, if the selected file is not in the required format, a warning message box will pop up and the user must respond to it before taking any further action. After responding to the warning message box, a new selection can be made. When a file is

successfully opened, the File Selection dialog box pops down (otherwise it will remain visible). To leave the dialog box without opening a file, choose the CANCEL button.

The Save option stores the changes made to the current Petri Net structure without closing the file. If the user chooses the Save option when a file has not yet been named, a Save dialog box pops up. It prompts the user to enter a file name. Two warning message boxes may pop up, one for an invalid file name (no name) and the other for a file name that already exists. The CANCEL button is provided to cancel the Save operation if so desired.

The Save As option allows a user to name and save the copy of a current file with a new name. The current file name, displayed in a text box, can be browsed but cannot be edited. Another text box is provided for the entry of a new name, which should be followed by the choice of the SAVE button. The new name will be displayed and the old file will be closed. If the recent changes to the old file are not saved before saving as a new file, the changes will not appear in the old file but they will appear in the new file. If the old file has not yet been named, a dialog box pops up to prompt the user to save the current file. If a user decides to save the file, the Save dialog box pops up.

The Quit option ends the current session of DrawPetri. When this option is selected, a Shut Down dialog box pops up. The user is prompted to save the current file. Three buttons: YES, NO, and CANCEL are provided. The YES button saves the file and ends the session (if a file name is given). If the file has not yet been named, the Save dialog box pops up (discussed previously). The NO button ends the session without saving the current changes to the file. The CANCEL button cancels this option.

5. Edit Menu

The Edit menu provides three options: Undo, EditLabel, and DeleteObject. Undo reverses the last action (drawing action) that is performed on the drawing area. This action can be executed by clicking the third button of the mouse. The SteppedFire option, which is located in the Firing menu, is not an undoable option. Only drawing actions are undoable. For example, a most recently deleted object can be restored using Undo. After restoring an object, if Undo is selected again, it deletes the object.

Using the option EditLabel, a user can change the label of a node (a place or a transition). A dialog box titled EDIT LABEL pops up. A user is prompted for an old and a new label for a node. If both labels are valid (an existing old label and a non-existing new label) and the DONE button is pressed, the change takes place. A warning message box may appear if an invalid label is given.

The option DeleteObject is used to delete an object. The object may be a place, a transition, or a link. When this option is selected, the cursor for the drawing area

changes to a pirate sign (skull and cross-bones). To delete an object, place the cursor over it and press the first button of the mouse.

6. Firing Menu

The Firing menu has three options: SteppedFire, ViewHistoryFile, and ExecutionSequence. To observe the dynamic behavior of a Petri Net model, select the SteppedFire option, and point and click on the transition to be fired. The selected transition should be an enabled transition, otherwise a warning message box will pop up. An enabled transition is indicated by a filled rectangle. If the selected transition is enabled, it is fired. After each firing, the marking of the Petri Net is updated. The ViewHistoryFile option displays the history file, generated for the current session of firing. For each transition fired, the history file contains the label of the transition fired and the marking of the Petri Net (the token assignment). The ExecutionSequence option displays the history of the execution sequence, i.e., a list of the transitions fired. The last fifty transitions (if more than 50) are displayed.

7. Help Menu

The Help menu provides a quick help during the invocation of DrawPetri. A brief help is available on options PLACE, TRANSITION, LINK, PUT_TOKEN, EDIT_SPEC, SHOW_SPEC, Undo, DeleteObject, EditLabel, SteppedFire, ViewHistoryFile, and ExecutionSequence. More detailed help is available in the User Guide for DrawPetri (i.e., this document).

8. Action Buttons

The seven action buttons on the left side of the application window are provided for a quick selection. These are the frequently-used options. The PLACE option can be selected by pointing and clicking on it. Upon selection of the PLACE option, the display area for the "current action selected" displays DRAW PLACE. Now the user can draw a circle on the drawing area representing a place. The radius of the circle is fixed. When a user first clicks some place on the drawing area, a place in the form of a circle with a dotted-line boundary appears. This place can be moved to a desired location. A click of the second button of the mouse will make the place "permanent" (i.e., the boundary becomes a solid line). A default label is provided for each place. The labels can be edited using the EditLabel option (discussed previously).

Select the TRANSITION option to draw a rectangle representing a transition. The width of a transition is fixed. A user can stretch a transition horizontally or vertically to

a desired length. When a user, after selecting the TRANSITION option, clicks on the drawing area, that point on the drawing area is taken as one of the corners of the rectangle representing the transition. After stretching to the desired length, a click of the second button of the mouse will make the transition permanent. A default label is provided for each transition. The labels can be edited using the EditLabel option (discussed previously).

To draw a link between a place and a transition, select the LINK option. When this option is selected, a dialog box titled LINK DEFINITION appears. The user is prompted to enter the "definition" of the link to be drawn. To "define" a link, the user should mention the source and the destination of the link. A valid label (i.e., an existing label) of a place or a transition should be given for the source and for the destination. If an invalid label is entered, a warning message box pops up. It also warns if a link is defined for a same type node, which is against the definition of a Petri Net graph (the graph must be bipartite, as discussed in Chapter II). If a link is defined correctly, the dialog box pops down. Now the user should select the source on the drawing area and drag the mouse to the destination. As soon as it hits the destination, the link is established. The source is selected by clicking on its boundary. If a user clicks away from the boundary (inside or outside), a warning message box pops up. It displays the definition of the link being drawn and asks the user to re-select the source. After selecting the correct source, a dotted link becomes visible. The user can stretch it and also change its path by clicking the second button of the mouse. The drawing of a link can be cancelled by clicking the third button of the mouse before hitting the destination. If any other option is selected during the process of drawing a link, the link will be canceled.

The PUT_TOKEN option allows a user to assign a number of tokens to a place. When this option is chosen, a dialog box titled PUT TOKEN pops up. A user is prompted for the label of the place where the tokens are to be assigned. Three buttons: INCREMENT, DECREMENT, and DONE are provided. The INCREMENT button increments the number of tokens by one and the DECREMENT button decrements the number of tokens by one. Initially, each place is assigned zero number of tokens. If the number of tokens at a place is a zero and a user clicks the DECREMENT button, a warning message box will pop up (negative number of tokens cannot be assigned). Other warning message boxes may also pop up for various reasons: for an invalid label (no label), for a label which does not exist, or for a label which is a transition. To pop down the PUT TOKEN dialog box, the DONE button is provided.

The EDIT_SPEC and SHOW_SPEC options deal with the specification of a node. The specification of a node consists of the label of the node, the type of the node (place or transition), and the interpretation of the node. The EDIT_SPEC option lets the user type in the interpretation of the node. Using the SHOW_SPEC option, a user can only view the interpretation of a node. When a user selects either of these two options, the cursor definition for the drawing area changes to a hand pointing to an object. When a user clicks on a node whose specification is to be edited or viewed, a window titled EDIT SPECIFICATION or SHOW SPECIFICATION pops up, where the user can edit or view the specification of the selected node. For the EDIT SPECIFICATION window, two

buttons, SAVE and CANCEL, are provided. The user can save the edited interpretation by clicking the SAVE button or can cancel it by clicking the CANCEL button. There is only one button, namely DONE, for the SHOW SPECIFICATION window. Clicking it will pop down the window.

The last action button is the HISTORY option. This is a toggle button (like a light switch). Every time it is clicked, it changes its state (ON/OFF). When this HISTORY switch is active (ON), a small square next to it becomes black; otherwise, it becomes white indicating that it is not active (OFF). Initially, it is in the OFF state. When it is turned ON, a history file is generated for a session of the firing of the Petri Net. For each transition fired, the history file contains the label of the transition fired and the marking of the Petri Net (i.e., its token assignment). When it is desired to generate a history file for a session of firing, a user should turn this switch ON. When this switch is turned ON, a dialog box pops up. It prompts the user to enter a file name for the history file to be generated. After entering a valid file name (a non-existing name), select the DONE button. The CANCEL button pops down the dialog box and the HISTORY switch is turned OFF. A warning message pops up for an invalid file name. If a valid file name is given and the DONE button is clicked, the HISTORY switch is turned ON. Now the user should select the SteppedFire option from the menu Firing to execute (to fire the transitions) the Petri Net. The history file is updated for each transition fired. If any option other than from the menus Firing and Help is chosen, the history switch is turned OFF (i.e., the firing session ends). The history file generated for a session of firing can be viewed using the ViewHistoryFile option present in the menu Firing.

APPENDIX C

SYSTEM ADMINISTRATOR GUIDE FOR DrawPetri

1. Description

The DrawPetri program is a graphical tool to model systems using Petri Net as a modeling tool. To run DrawPetri, the X Window System and the Motif widget set are required. The user interface of DrawPetri is developed using the OSF/Motif widget set. DrawPetri's interface consists of menus, windows, dialog boxes, as well as other features that make it easy for a user to communicate with DrawPetri.

DrawPetri is designed for use with a mouse, but its menu system and other selection panels can be traversed using the keyboard also. It is recommended that users use the mouse for ease-of-use. The initial interface screen of DrawPetri contains different actions accessible using four pull down menus and seven action buttons. The display bar at the bottom of the main window shows the current action selected. To draw a Petri Net model of a system, a scrollable (virtualized) drawing area is provided. The filename is displayed at the top of the main window.

2. Maintenance

The information about drawing objects is stored in an array named "buffer". This array is a member of the structure "graphics_data" (see file data_str.h) The size of the buffer is 1000, i.e., a maximum of 1000 objects can to drawn. The constant MAX_OBJECTS defined in the header file data_str.h defines this limit. To change this limit, the definition of the constant MAX_OBJECTS needs to be modified.

A node can have a maximum of 30 nodes in its preset (incoming links) and postset (out going links). These limits are defined by the constants MAX_PRESET and MAX_POSTSET, respectively. They are defined in the header file data_str.h. These limits can be set according to the requirements.

The interpretation of a node has a limit of 200 characters. This limit is defined by the constant INFO_LENGTH. It is defined in the header file data_str.h. It can be modified if desired.

The ExecutionSequence option present in the menu Firing, displays a maximum of the fifty latest transitions fired. The constant HISTORY_COUNT defined in the header file fire.h defines this limit. This limit can be modified. The line "#define HISTORY_COUNT 50" needs to be modified.

The maximum length of the label of a node is 300. The constant LABEL_LENGTH defines this length. It is defined in the header file data_str.h. This value can be modified if desired.

The help and warning messages of DrawPetri are located in the files help.c and warn_msg.c. These are ASCII files, so any ASCII editor can be used to modify these files.

To add or modify a pull down menu, the routine CreatePanel2 located in the file user_interface.c needs to be modified. The BuildPulldownMenu() routine is to be called to add a new pull down menu. To add an item in an existing menu, the declaration of that menu needs to be modified.

The size of the application window is the size of the Motif Form widget (XmFormWidgetClass). The default size is 1000x1200 (width x height) pixels. This size can be modified by setting the XmNwidth and the XmNheight resources of the widget "form". The corresponding code is in the main routine in file DrawPetri.c. Similarly, the default size of the drawing area can be modified by setting the resources of the widget "petri_board".

If a new program file is added in the application, Makefile needs to be modified.

3. Options

As an Xt toolkit-based program, DrawPetri accepts the standard X toolkit command line options [Young90]. The commonly-used options include:

-bg color
: This option defines the color to be used for the background of the window. The default is white.

-display display
: This option defines the X server to which the program is connected.

-fg color
: This option defines the color to be used for the foreground of the window. The default is black.

-geometry geometry
: This option defines the user preferred size and position of the application window.

-iconic

This option indicates that DrawPetri should ask the window manager to start it as an icon instead of as a normal window.

-rv

This option reverses the foreground and background colors.

-title string

This option defines the window title string, which may be displayed by window managers. The default title is the command line specified after the -e option, if any; if not, the application name is used.

APPENDIX D

PROGRAM LISTING

The program structure of the DrawPetri tool is discussed in section 3.2.1. The order of the program listings is as follows:

data_str.h
DrawPetri.h
DrawPetri.c
user_interface.h
user_interface.c
alloc_node.h
alloc_node.c
init.h
init.c
open_file.h
open_file.c
close_file.h
close_file.c
draw_object.h
draw_object.c
link.h
link.c
edit_label.h
edit_label.c
delete.h
delete.c
put_token.h
put_token.c
spec.h
spec.c
save.h
save.c
cursor.h
cursor.c
help.h
help.c
utility.h

utility.c
undo.h
undo.c
warn_msg.c
fire.h
fire.c

```
/*
File name:   data_str.h
Description: This file contains the data structures and the constants,
             used in the software.
*/

#include <X11/Intrinsic.h>

/* constants definitions used for flags */
#define UNDO_LABEL          100
#define UNDO_DEL_LINK        99
#define UNDO_UNDEL_LINK      98
#define UNDO_DEL_PLACE       97
#define UNDO_UNDEL_PLACE     96
#define UNDO_DEL_TRANS       95
#define UNDO_UNDEL_TRANS     94
#define EDIT_SPEC            93
#define SHOW_SPEC            92
#define ENABLED              91
#define PRESET               90
#define POSTSET              89
#define NO                    0
#define YES                   1
#define NONE                 -1

#define MAX_OBJECTS         1000 /* max. number of objects */
#define FILE_NAME_LENGTH     100  /* max. length of a filename */
#define ACTION_LENGTH        100  /* max. length of a string for current action */
#define INFO_LENGTH          200  /* max. length for the interpretation of a node */
#define LABEL_LENGTH         300  /* max. length for a label */
#define MSG_LENGTH           500  /* max. length for a warning message string */
#define MAX_PRESET            30  /* max. number nodes in the preset */
#define MAX_POSTSET           30  /* max. number nodes in the postset */
#define MAX_ARGS              25  /* max. number of arguments */


typedef struct PL {
  char       label[LABEL_LENGTH];                 /* label */
  char       preset[MAX_PRESET][LABEL_LENGTH];    /* preset */
  char       postset[MAX_POSTSET][LABEL_LENGTH];  /* postset */
  int        cur_preset_no;                       /* current number of nodes in
                                                     the preset */
  int        cur_postset_no;                      /* current number of nodes in
                                                     the postset */
  int        no_of_tokens;                        /* number of tokens */
  int        freq_in_preset;                      /* frequency in the preset */
  char       interpretation[INFO_LENGTH];         /* interpretation */
  struct PL  *next;                               /* pointer to a next place */
  struct PL  *prev;                               /* pointer to a previous place */
}PLACE;


typedef struct {                              /* place head */
  char   place_selected[LABEL_LENGTH];        /* last place selected */
  char   place_pre_selected[LABEL_LENGTH];    /* place before the current
                                                 selection */
  int    pre_post;                            /* indicates if a selected place is
                                                 a preset or a postset of a
                                                 transition, used for undo action */
  int    pre_post_index;                      /* gives the index in the preset or
                                                 the postset */
  int    last_token;                          /* last token put in a place, used
                                                 for undo action */
  PLACE  *HEAD;                               /* pointer to the start of the list */
}P_HEAD;


typedef struct TRANS{
```

```
  char        label[LABEL_LENGTH];                /* label */
  char        preset[MAX_PRESET][LABEL_LENGTH];   /* preset */
  char        postset[MAX_POSTSET][LABEL_LENGTH]; /* postset */
  int         cur_preset_no;                      /* current number of nodes in
                                                     the preset */
  int         cur_postset_no;                     /* current number of nodes in
                                                     the postset */
  int         enabled;                            /* flag for enabled transition
                                                     */
  char        interpretation[INFO_LENGTH];        /* interpretation */
  struct TRANS  *next;                            /* pointer to a next
                                                     transition */
  struct TRANS  *prev;                            /* pointer to a previous
                                                     transition */
}TRANSITION;


typedef struct {                                  /* transition head */
  char        trans_selected[LABEL_LENGTH];       /* last transition selected */
  char        trans_pre_selected[LABEL_LENGTH]; /* transition before the current
                                                   selection */
  int         pre_post;                           /* indicates if a selected
                                                     transition is a preset or a
                                                     postset of a place, used for
                                                     undo action */
  int         pre_post_index;                     /* gives the index in the preset
                                                     or the postset */
  char        del_link[LABEL_LENGTH];             /* label of the last deleted
                                                     link, used for undo action */
  TRANSITION  *HEAD;                              /* pointer to the start of the
                                                     transition list */
}T_HEAD;


typedef struct {              /* buffer element */
  char object[ACTION_LENGTH]; /* object to be drawn */
  char label[LABEL_LENGTH];   /* label */
  int  x1, y1, x2, y2;        /* positions */
  void (*func)();             /* function to draw the object */
  GC   gc;                    /* graphic context for the object */
}BUF_ELEMENT;


typedef struct {                    /* graphics data */
  char    object[ACTION_LENGTH];    /* object to be drawn */
  char    label[LABEL_LENGTH];      /* label */
  int     x1, x2, y1, y2;           /* positions */
  int     height, width;            /* width and height of an object */
  int     foreground, background;   /* foreground and background color */
  void    (*current_func)();        /* function to draw the object */
  GC      gc;                       /* graphics context of the object */
  GC      xorgc;                    /* graphics context for the rubber band effect */
  BUF_ELEMENT buffer[MAX_OBJECTS]; /* buffer of objects */
  int     next_pos;                 /* next position in the buffer */
}graphics_data;


typedef struct {
  Widget         label_widget1; /* widget */
  Widget         label_widget2; /* widget */
  graphics_data  *temp_data;    /* pointer to a graphics data */
}EDIT_LABEL_STRUCT;


typedef struct {
  Widget    w;          /* widget */
  graphics_data  *data; /* pointer to a graphics data */
}MenuCallbackData;

typedef struct _menu_item {   /* pull down menu item */
  char         *label;        /* label of the pull down menu */
  WidgetClass  *class;        /* pointer to a Motif widget class */
  char         mnemonic;      /* mnemonic */
  char         *accelerator;  /* accelerator */
  char         *accel_text;   /* text for the accelerator */
  void         (*callback)(); /* callback function */
  caddr_t      callback_data; /* callback data */
}MenuItem;


typedef struct del_item {  /* list for the deleted link, used for undo action */
```

```
  BUF_ELEMENT      buffer; /* buffer */
  struct del_item  *next;  /* pointer to a next deleted item */
}DEL_ITEM;

typedef struct HIS {                /* history item */
  char  trans_label[LABEL_LENGTH]; /* label */
  struct HIS  *next;                /* pointer to a next item */
  struct HIS  *prev;                /* pointer to a previous item */
}HISTORY_ITEM;

typedef struct {       /* history head */
  int          count; /* number of items in the history list */
  HISTORY_ITEM  *HEAD; /* pointer to the start of the history list */
  HISTORY_ITEM  *TAIL; /* pointer to the end of the history list */
}H_HEAD;


/*
File name:   global.h
Description: This file contains the external declaration of the global variables.
             These global variables are actually declared in DrawPetri.c. The file
             global.h is included in all the files.
*/

#define FILE_NAME_LENGTH   100
#define ACTION_LENGTH      100
#define LABEL_LENGTH       300

/* these global variables are declared in DrawPetri.c */

extern H_HEAD      *H_LIST;                    /* see DrawPetri.h for details */
extern DEL_ITEM    *DEL_LIST;
extern P_HEAD      *P_LIST;
extern T_HEAD      *T_LIST;
extern int         PLACE_CNT;
extern int         TRANSITION_CNT;
extern int         LINK_CNT;
extern int         UNDO_ACTION;
extern EDIT_LABEL_STRUCT   edit_label_cb_struct;
extern char  LAST1[LABEL_LENGTH];
extern char  LAST2[LABEL_LENGTH];
extern char  SRC_FOR_UNDO_LINK[LABEL_LENGTH];
extern char  DST_FOR_UNDO_LINK[LABEL_LENGTH];
extern char  GLOBAL_FILE_NAME[FILE_NAME_LENGTH];
extern char  action_selected[ACTION_LENGTH];
extern Widget    drawing, filename_display, action_display, toplevel;
extern Cursor    busy_cursor, pirate_cursor, hand_cursor;


/*
File Name:   DrawPetri.h
Description: This file includes header files required for the module
             DrawPetri.c. It also declares the global variables. All the
             external functions used in DrawPetri.c are also declared.
*/

#include <Xm/Form.h>
#include <Xm/Frame.h>
#include <Xm/RowColumn.h>
#include <Xm/Separator.h>
#include <Xm/SeparatoG.h>
#include <Xm/DrawingA.h>
#include <Xm/ScrolledW.h>
#include "data_str.h"

/*
These externally defined functions are called in DrawPetri.c
*/
extern void initialize();      /* declared in init.c */
extern void InitCursors();     /* declared in cursor.c */
extern Widget CreatePanel2();  /* declared in user_interface.c */
extern void CreatePanel1a();   /* declared in user_interface.c */
extern void CreatePanel1b();   /* declared in user_interface.c */
extern void CreatePanel1c();   /* declared in user_interface.c */
extern void CreatePanel1d();   /* declared in user_interface.c */
extern void init_data();       /* declared in init.c */
extern void RewriteFileName(); /* declared in utility.c */
extern void RewriteAction();   /* declared in utility.c */
extern void GetStartPos();     /* declared in user_interface.c */
extern void TrackMouse();      /* declared in user_interface.c */
extern void Refresh();         /* declared in user_interface.c */
```

```
/* declaration of the global variables */
H_HEAD      *H_LIST;        /* pointer to the head of the history list */
DEL_ITEM    *DEL_LIST;      /* pointer to the first item of the delete list */
P_HEAD      *P_LIST;        /* pointer to the head of the place list */
T_HEAD      *T_LIST;        /* pointer to the head of the transition list */
int         PLACE_CNT;      /* place count */
int         TRANSITION_CNT; /* transition count */
int         LINK_CNT;       /* link count */
int         UNDO_ACTION;    /* undo action */
int         HISTORY_SWITCH; /* history switch (ON/OFF) */
EDIT_LABEL_STRUCT   edit_label_cb_struct; /* structure used by callback function
                                             in edit_label.c */
char  LAST1[LABEL_LENGTH];                /* temporary storage for a label */
char  LAST2[LABEL_LENGTH];                /* temporary storage for a label */
char  SRC_FOR_UNDO_LINK[LABEL_LENGTH];    /* temporary storage for a label */
char  DST_FOR_UNDO_LINK[LABEL_LENGTH];    /* temporary storage for a label */
char  GLOBAL_FILE_NAME[FILE_NAME_LENGTH]; /* filename */
char  trace_filename[FILE_NAME_LENGTH];   /* filename */
char  action_selected[ACTION_LENGTH];     /* current action selected */

/* global widgets */
Widget    drawing, filename_display, action_display, toplevel;

/* global cursor shapes */
Cursor    busy_cursor, pirate_cursor, hand_cursor;


/*
File name:   DrawPetri.c
Description: This file contains the main() module. It sets up the initial
             user-interface of the DrawPetri. Functions included in the file
             user_interface.c are called to established the user-interface.
*/

#include "DrawPetri.h"

main(argc, argv)
int   argc;
char  *argv[];
{
  Widget   form, petri_board, panel1, frame1, frame2, frame3,
           frame4, scroll_win ,panel2, sep1, sep4, sep5;
  int      n;
  Arg      wargs[MAX_ARGS];
  graphics_data  data;

  initialize(); /* initializes the system */
  toplevel = XtInitialize("DRAW_PETRI", "Draw_petri", NULL, 0, &argc, argv);
  InitCursors();
  /*
  Initiating a form widget, child of toplevel widget. All the regions of the
  main user-interface will be children of this form widget. Size of the form is
  950x700 (WidthxHeight) pixels. This size can be changed if desired.
  */
  n = 0;
  XtSetArg(wargs[n], XmNwidth,  950); n++;
  XtSetArg(wargs[n], XmNheight, 700); n++;
  form = XtCreateManagedWidget("form", xmFormWidgetClass,
                               toplevel, wargs, n);
  /*
  The widget frame1 is a frame widget which contains the pull down menus at the
  top left side of the screen. The pull down menus are managed by a row column
  widget panel2 (child of frame1). The function CreatePanel2() creates the pull
  down menu system. Each pull down menu is made child of the widget panel2
  (row column).
  */
  n = 0;
  XtSetArg(wargs[n], XmNtopAttachment, XmATTACH_FORM); n++;
  XtSetArg(wargs[n], XmNleftAttachment, XmATTACH_FORM); n++;
  XtSetArg(wargs[n], XmNorientation, XmHORIZONTAL); n++;
  XtSetArg(wargs[n], XmNshadowThickness, 5); n++;
  XtSetArg(wargs[n], XmNshadowType, XmSHADOW_OUT); n++;
  frame1 = XtCreateManagedWidget("frame1",xmFrameWidgetClass,form,wargs,n);
  panel2 = XtCreateManagedWidget("panel2", xmRowColumnWidgetClass,
                                 frame1, wargs, n);
  CreatePanel2(panel2, &data);
  /*
  The widget sep1 is a separator widget (horizontal), separating the pull down
  menu system and the display region for the filename from rest of the user interface.
  */
  n = 0;
```

```
XtSetArg(wargs[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(wargs[n], XmNtopWidget,     frame1); n++;
XtSetArg(wargs[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(wargs[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(wargs[n], XmNorientation, XmHORIZONTAL); n++;
XtSetArg(wargs[n], XmNseparatorType, XmNO_LINE); n++;
XtSetArg(wargs[n], XmNshadowThickness, 5); n++;
sep1 = XtCreateManagedWidget("sep1",xmSeparatorWidgetClass,
                             form,wargs,n);
/*
The sep4 is an invisible vertical separator between the pull down menu system and
the filename display region.
*/
n = 0;
XtSetArg(wargs[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(wargs[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(wargs[n], XmNleftWidget,     frame1); n++;
XtSetArg(wargs[n], XmNbottomAttachment, XmATTACH_WIDGET); n++;
XtSetArg(wargs[n], XmNbottomWidget,     sep1); n++;
XtSetArg(wargs[n], XmNseparatorType, XmNO_LINE); n++;
XtSetArg(wargs[n], XmNshadowThickness, 0); n++;
XtSetArg(wargs[n], XmNorientation, XmVERTICAL); n++;
sep4 = XtCreateManagedWidget("sep4",xmSeparatorWidgetClass,
                             form,wargs,n);
n = 0;
XtSetArg(wargs[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(wargs[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(wargs[n], XmNleftWidget,     sep4); n++;
XtSetArg(wargs[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(wargs[n], XmNbottomAttachment, XmATTACH_WIDGET); n++;
XtSetArg(wargs[n], XmNbottomWidget,     sep1); n++;
XtSetArg(wargs[n], XmNshadowThickness, 5); n++;
XtSetArg(wargs[n], XmNshadowType, XmSHADOW_OUT); n++;
frame2 = XtCreateManagedWidget("frame2",xmFrameWidgetClass,form,wargs,n);
filename_display = XtCreateManagedWidget("filename",
                                         xmDrawingAreaWidgetClass,
                                         frame2, NULL, 0);
XtAddCallback(filename_display, XmNexposeCallback,
              RewriteFileName, NULL);

/*
The widget frame3 is a frame widget which contains the action buttons
provided at left side of the screen (below pull down menu system and left
of drawing area). The pull down menus are managed by a row column widget,
panel1 (child of frame3). The functions CreatePanel1a() through
CreatePanel1c() create the buttons needed for different actions. Separators
are also made between the group of buttons. Separators are made as above.
*/
n = 0;
XtSetArg(wargs[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(wargs[n], XmNtopWidget,     sep1); n++;
XtSetArg(wargs[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetArg(wargs[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(wargs[n], XmNshadowThickness, 5); n++;
XtSetArg(wargs[n], XmNshadowType, XmSHADOW_OUT); n++;
frame3 = XtCreateManagedWidget("frame3",xmFrameWidgetClass,form,wargs,n);
n = 0;
XtSetArg(wargs[n], XmNorientation, XmVERTICAL); n++;
panel1 = XtCreateManagedWidget("panel1", xmRowColumnWidgetClass,
                               frame3, wargs, n);
CreatePanel1a(panel1, &data);

/* makes separator and calls CreatePanel1b() */
n = 0;
XtSetArg(wargs[n], XmNseparatorType, XmDOUBLE_LINE); n++;
XtSetArg(wargs[n], XmNshadowThickness, 6); n++;
XtCreateManagedWidget("panel1sep1",xmSeparatorWidgetClass,
                      panel1,wargs,n);
CreatePanel1b(panel1, &data);
/* makes separator and calls CreatePanel1c() */
n = 0;
XtSetArg(wargs[n], XmNseparatorType, XmDOUBLE_LINE); n++;
XtSetArg(wargs[n], XmNshadowThickness, 5); n++;
XtCreateManagedWidget("panel1sep2",xmSeparatorWidgetClass,
                      panel1,wargs,n);
CreatePanel1c(panel1, &data);

/* last separator within panel1 */
n = 0;
XtSetArg(wargs[n], XmNseparatorType, XmDOUBLE_LINE); n++;
XtSetArg(wargs[n], XmNshadowThickness, 5); n++;
```

```
  XtCreateManagedWidget("panel1sep3",xmSeparatorWidgetClass,
                        panel1,wargs,n);
  /* creates the option HISTORY */
  CreatePanelId(panel1);
  /*
  An invisible vertical separator. It separates the left panel of buttons from
  the drawing area and the display region for the action selected.
  */
  n = 0;
  XtSetArg(wargs[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
  XtSetArg(wargs[n], XmNtopWidget,     sep1); n++;
  XtSetArg(wargs[n], XmNbottomAttachment, XmATTACH_FORM); n++;
  XtSetArg(wargs[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
  XtSetArg(wargs[n], XmNleftWidget,     frame3); n++;
  XtSetArg(wargs[n], XmNseparatorType, XmNO_LINE); n++;
  XtSetArg(wargs[n], XmNshadowThickness, 0); n++;
  XtSetArg(wargs[n], XmNorientation, XmVERTICAL); n++;
  sep5 = XtCreateManagedWidget("sep5",xmSeparatorWidgetClass,
                               form,wargs,n);
  /*
  The widget action_display is used to display the current selected
  action.
  */
  n = 0;
  XtSetArg(wargs[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
  XtSetArg(wargs[n], XmNleftWidget,     sep5); n++;
  XtSetArg(wargs[n], XmNrightAttachment, XmATTACH_FORM); n++;
  XtSetArg(wargs[n], XmNbottomAttachment, XmATTACH_FORM); n++;
  XtSetArg(wargs[n], XmNshadowThickness, 5); n++;
  XtSetArg(wargs[n], XmNshadowType, XmSHADOW_OUT); n++;
  frame4 = XtCreateManagedWidget("frame4",xmFrameWidgetClass,form,wargs,n);
  action_display = XtCreateManagedWidget("action",
                                         xmDrawingAreaWidgetClass,
                                         frame4, NULL, 0);
  XtAddCallback(action_display, XmNexposeCallback,
                RewriteAction, NULL);
  /*
  The widget scroll_win is the parent of the widget petri_board (drawing area).
  It is used to scroll the drawing area.
  */
  n = 0;
  XtSetArg(wargs[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
  XtSetArg(wargs[n], XmNtopWidget,     sep1); n++;
  XtSetArg(wargs[n], XmNbottomAttachment, XmATTACH_WIDGET); n++;
  XtSetArg(wargs[n], XmNbottomWidget,     frame4); n++;
  XtSetArg(wargs[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
  XtSetArg(wargs[n], XmNleftWidget,     sep5); n++;
  XtSetArg(wargs[n], XmNrightAttachment, XmATTACH_FORM); n++;
  XtSetArg(wargs[n], XmNscrollBarDisplayPolicy, XmAS_NEEDED); n++;
  XtSetArg(wargs[n], XmNscrollingPolicy, XmAUTOMATIC); n++;
  XtSetArg(wargs[n], XmNshadowThickness, 5); n++;
  scroll_win = XtCreateManagedWidget("scroll_win",
                                     xmScrolledWindowWidgetClass,
                                     form,wargs,n);
  /*
  The widget petri_board is used as a drawing area. Its default size is
  1000x1200 (WidthxHeight) pixels. This size can be changed if desired.
  */
  n = 0;
  XtSetArg(wargs[n], XmNwidth, 1000); n++;
  XtSetArg(wargs[n], XmNheight, 1200); n++;
  XtSetArg(wargs[n], XmNshadowThickness, 5); n++;
  petri_board = XtCreateManagedWidget("petri_board", xmDrawingAreaWidgetClass,
                                      scroll_win, wargs, n);
  drawing  = petri_board;
  XtAddCallback(petri_board, XmNexposeCallback, Refresh, &data);

  /* Adds event handlers */
  XtAddEventHandler(petri_board, ButtonPressMask, FALSE, GetStartPos, &data);
  XtAddEventHandler(petri_board, PointerMotionMask, FALSE, TrackMouse, &data);
  init_data(petri_board, &data); /* initializes the data */
  XtRealizeWidget(toplevel);
  XtMainLoop(); /* starts main loop */
}


/*
File name:   init.h
Description: Its contains the header files required for the file init.c.
             It also declares the functions defined in the file init.c.
*/
```

```
#include <X11/StringDefs.h>
#include <stdio.h>
#include <string.h>
#include "data_str.h"
#include "global.h"

/*
The file delete.c consists of following functions.
For details, refer explanation before the declaration of each
function.
*/
void initialize();
void init_data();
GC CreateXorGc();


/*
File name:   init.c
Description: It contains the functions required to initialize the system.
*/

#include "init.h"
extern char trace_filename[FILE_NAME_LENGTH];

/*
initialize() initializes the environment at the startup of the
system. It initializes some global variables which are
defined and explained in the header file DrawPetri.h.
*/

void initialize()
{
  /* initializes the variables */
  PLACE_CNT      = 0;
  TRANSITION_CNT = 0;
  LINK_CNT       = 0;
  UNDO_ACTION    = NO;
  DEL_LIST       = NULL;

  /* initial name of the file is given */
  strcpy(GLOBAL_FILE_NAME,"Untitled");

  /* initial name of the file is given */
  strcpy(trace_filename,"Untitled");

  /* initial display for the action selected */
  strcpy(action_selected,"NOTHING");

  /*
  The P_LIST is dynamically created. Allocates memory for P_HEAD
  (see data structure for its members) and initializes its members
  with default values.
  */
  if((P_LIST = (P_HEAD *)malloc(sizeof(P_HEAD))) == NULL) {
    printf("Out Of Memory\n");
    exit(0);
  }
  P_LIST->HEAD = NULL;
  strcpy(P_LIST->place_pre_selected,"");
  strcpy(P_LIST->place_selected,"");
  P_LIST->pre_post       = NULL;
  P_LIST->pre_post_index = -1;

  /*
  The T_LIST is dynamically created. Allocates memory for T_HEAD
  (see data structure for its members) and initializes its members
  with default values.
  */
  if((T_LIST = (T_HEAD *)malloc(sizeof(T_HEAD))) == NULL) {
    printf("Out Of Memory\n");
    exit(0);
  }
  T_LIST->HEAD = NULL;
  strcpy(T_LIST->trans_pre_selected,"");
  strcpy(T_LIST->trans_selected,"");
  T_LIST->pre_post = NONE;
  T_LIST->pre_post_index = NONE;
  /*
  The H_LIST is dynamically created. Allocates memory for H_HEAD
  (see data structure for its member) and initializes its only
  member HEAD with NULL .
```

```
   */
   if((H_LIST = (H_HEAD *)malloc(sizeof(H_HEAD))) == NULL) {
     printf("Out Of Memory\n");
     exit(0);
   }
   H_LIST->HEAD = NULL;
}


/*
init_data() creates two graphics contexts (GCs). One GC is used
by the graphics objects (standard), while the other GC is used
to get rubber band effect.
The member gc of the structure "graphics_data" (see data_str.h) is standard the GC
and the member xorgc is for rubber band effect.
The graphics context is created using the foreground and background
colors obtained from the widget's resources.
*/

void init_data(w, data)
Widget      w;
graphics_data   *data;
{
  XGCValues      values;
  Arg            wargs[5];
  int            n;

  data->current_func = NULL;
  data->next_pos     = 0;

  /* obtains the foreground and background colors from the widget */
  n = 0;
  XtSetArg(wargs[n], XtNforeground, &values.foreground); n++;
  XtSetArg(wargs[n], XtNbackground, &values.background); n++;
  XtGetValues(w, wargs, n);

  /* defines the standard GC */
  data->gc = XtGetGC(w, GCForeground | GCBackground , &values);

  /*
  The function CreateXorGc() creates a graphics context in
  exclusive-OR mode which is used for the rubber band effect.
  */
  data->xorgc =  CreateXorGc(w);
}


/*
CreateXorGc() creates a graphics context in exclusive-OR mode
which is used to draw the rubber objects.
*/

GC CreateXorGc(w)
Widget   w;
{
  XGCValues values;
  GC        gc;
  Arg       wargs[10];

  /* obtains the foreground and background colors from the widget */
  XtSetArg(wargs[0], XtNforeground, &values.foreground);
  XtSetArg(wargs[1], XtNbackground, &values.background);
  XtGetValues(w, wargs,2);
  /* sets the foreground to the XOR of the foreground and background */
  values.foreground = values.foreground ^ values.background;
  /* sets to draw a dashed line */
  values.line_style = LineOnOffDash;
  values.function   = GXxor;
  gc = XtGetGC(w, GCForeground | GCBackground |
               GCFunction | GCLineStyle, &values);
  return gc;
}


/*
File name:   user_interface.h
Description: It contains header files required for the file user_interface.c.
             It also declares the external functions and the functions
             defined in the file user_interface.c
*/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <Xm/PushB.h>
#include <Xm/PushBG.h>
#include <Xm/SeparatoG.h>
#include <Xm/CascadeB.h>
#include <Xm/RowColumn.h>
#include <Xm/FileSB.h>
#include <Xm/ToggleBG.h>
#include <math.h>
#include "data_str.h"
#include "global.h"

typedef struct {    /* used to defined an option and its function */
  char   *object;   /* object selected */
  void  (*func) (); /* function to draw the object */
}user_data;

/*
These externally defined function are called in user_interface.c
*/
extern void SetHandCursor();      /* defined in cursor.c */
extern void SetBusyCursor();
extern void SetPirateCursor();
extern void ResetCursor();
extern void InitCursors();
extern void PutTokenHelp();       /* defined in help.c */
extern void EditSpecHelp();
extern void ShowSpecHelp();
extern void DeleteHelp();
extern void PlaceHelp();
extern void TransHelp();
extern void LinkHelp();
extern void FireHelp();
extern void UndoHelp();
extern void TransSelected();      /* defined in fire.c */
extern void FiringHistory();
extern void SteppedFire();
extern void DrawPlace();          /* defined in draw_objects.c */
extern void DrawToken();
extern void DrawLink();
extern void DrawTransition();
extern void DrawArrowHeadLink();
extern TRANSITION *GetNewTrans(); /* defined in alloc_node.c */
extern PLACE      *GetNewPlace();
extern void PopUpFileSelector();  /* defined in open_file.c */
extern void OpenNewFile();
extern int  OpenFile();
extern void save_object();        /* defined in save.c */
extern void SelectSaveFileOption();
extern void SaveFileAs();
extern void SaveLinkInfo();
extern void GetLinkInfo();        /* defined in link.c */
extern int  BellIfLinkSourceOk();
extern int  BellIfLinkDestOk();
extern void RewriteFileName();    /* defined in utility.c */
extern void RewriteAction();
extern void RedrawDrawingBoard();
extern void PutErrorDialog();
extern void initialize();         /* defined in init.c. */
extern void init_data();
extern void DeleteObject();       /* defined in delete.c. */
extern void UndoLastDraw();       /* defined in undo.c. */
extern void SelectNodeForSpec();  /* defined in spec.c */
extern void GetTokenInfo();       /* defined in put_token.c */
extern void GetLabelInfo();       /* defined in edit_label.c */
extern void CloseFile();          /* defined in close_file.c */
extern char *GetFileMsgStr();     /* defined in warn_msg.c. */
extern void ViewHistoryFile();
extern void ExeSequence();

/*
The file user_interface.c consists of the following functions.
For details, refer explanation before the declaration of each function.
*/
void RemoveIncompleteLink();
void FileSelectionCancel();
void FileSelectionDone();
void CreatePanel1a();
void CreatePanel1b();
```

```
void CreatePanel1c();
void CreatePanel1d();
void SetDeleteObj();
void GetStartPos();
void TrackMouse();
void GetEndPos();
void Activate();
void Refresh();
Widget CreateFileSelector();
Widget CreatePanel2();
Widget BuildPulldownMenu();


/*
File name:   user_interface.c
Description: This file contains the functions required to make the
             user-interface of the DrawPetri.
 */

#include "user_interface.h"

user_data selection[] = {          /* three options */
    "PLACE",        DrawPlace,     /* title of the option and its function */
    "TRANSITION",   DrawTransition,
    "LINK",         DrawLink,
};

/*
CreatePanel1a() creates a part of the option panel provided on the left
side of the screen. The options provided are PLACE, TRANSITION, and LINK.
*/

void CreatePanel1a(parent, data)
Widget    parent;
graphics_data  *data;
{
  Widget   w, titles;
  Arg      wargs[MAX_ARGS];
  int      i, n;

  n = 0;
  XtSetArg(wargs[n], XmNentryClass, xmPushButtonWidgetClass);
  n++;
  XtSetArg(wargs[n], XmNx, 10); n++;
  XtSetArg(wargs[n], XmNy, 60); n++;
  titles = XmCreateRadioBox(parent, "titles", wargs, n);
  XtManageChild(titles);
  for(i = 0; i < XtNumber(selection); i++) {
    n = 0;
    XtSetArg(wargs[n], XmNuserData, &selection[i]); n++;
    XtSetArg(wargs[n], XmNshadowThickness, 1); n++;
    w = XtCreateManagedWidget(selection[i].object, xmPushButtonWidgetClass,
                              titles, wargs, n);
    XtAddCallback(w, XmNactivateCallback, Activate, data);
  }
}


user_data panel1b[] = {
    "PUT_TOKEN",          DrawToken,
};


/*
CreatePanel1b() creates the fourth option of the panel provided
on the left side of the screen. The option created is PUT_TOKEN.
*/

void CreatePanel1b(parent, data)
Widget    parent;
graphics_data  *data;
{
  Widget   w, titles;
  Arg      wargs[MAX_ARGS];
  int      i, n;

  n = 0;
  XtSetArg(wargs[n], XmNentryClass, xmPushButtonWidgetClass);
  n++;
  XtSetArg(wargs[n], XmNx, 10); n++;
```

```
    XtSetArg(wargs[n], XmNy, 60); n++;
    titles = XmCreateRadioBox(parent, "titles", wargs, n);
    XtManageChild(titles);
    for(i = 0; i < XtNumber(panel1b); i++) {
      n = 0;
      XtSetArg(wargs[n], XmNuserData, &panel1b[i]); n++;
      XtSetArg(wargs[n], XmNshadowThickness, 1); n++;
      w = XtCreateManagedWidget(panel1b[i].object, xmPushButtonWidgetClass,
                                titles, wargs, n);
      XtAddCallback(w, XmNactivateCallback, Activate, data);
    }
}

user_data panel1c[] = {
    "EDIT_SPEC",        SelectNodeForSpec,
    "SHOW_SPEC",        SelectNodeForSpec,
};

/*
CreatePanel1c() creates the fifth and the sixth options of the panel provided
on the left side of the screen. The options created are EDIT_SPEC and
SHOW_SPEC.
*/

void CreatePanel1c(parent, data)
Widget    parent;
graphics_data  *data;
{
  Widget    w, titles;
  Arg       wargs[MAX_ARGS];
  int       i, n;

  n = 0;
  XtSetArg(wargs[n], XmNentryClass, xmPushButtonWidgetClass);
  n++;
  XtSetArg(wargs[n], XmNx, 10); n++;
  XtSetArg(wargs[n], XmNy, 60); n++;
  titles = XmCreateRadioBox(parent, "titles", wargs, n);
  XtManageChild(titles);
  for(i = 0; i < XtNumber(panel1c); i++) {
    n = 0;
    XtSetArg(wargs[n], XmNuserData, &panel1c[i]); n++;
    XtSetArg(wargs[n], XmNshadowThickness, 1); n++;
    w = XtCreateManagedWidget(panel1c[i].object, xmPushButtonWidgetClass,
                              titles, wargs, n);
    XtAddCallback(w, XmNactivateCallback, Activate, data);
  }
}

/*
CreatePanel1d() creates the option HISTORY
*/

void CreatePanel1d(parent)
Widget parent;
{
  Widget    history_on;

  history_on = XtCreateManagedWidget("HISTORY", xmToggleButtonGadgetClass,
                                     parent, NULL, 0);
  XtAddCallback(history_on, XmNvalueChangedCallback, FiringHistory, NULL);
}


/*
CreatePanel2() creates the panel which contains all the pull-down menus.
The available pull-down menus are File, Edit, Firing, and Help.
*/

Widget CreatePanel2(parent, data)
Widget    parent;
graphics_data  *data;
{
  Widget    file_selector;
  Widget    w;
  Arg       wargs[10];
  int       n;

static MenuItem  fire_items[] = { /* items of Firing */
  {"SteppedFiring", &xmPushButtonWidgetClass, 'S', NULL, NULL,
    SteppedFire, NULL},
```

```
  {"ExecutionSequence", &xmPushButtonWidgetClass, 'V', NULL, NULL,
    ExeSequence, NULL},
  {"ViewHistoryFile", &xmPushButtonWidgetClass, 'V', NULL, NULL,
    ViewHistoryFile, NULL},
  NULL,
};

static MenuItem  help_items[] = { /* items of Help */
  {"PLACE", &xmPushButtonWidgetClass, 'P', NULL, NULL,
    PlaceHelp, NULL},
  {"TRANSITION", &xmPushButtonWidgetClass, 'T', NULL, NULL,
    TransHelp, NULL},
  {"LINK", &xmPushButtonWidgetClass, 'L', NULL, NULL,
    LinkHelp, NULL},
  {"PUT_TOKEN", &xmPushButtonWidgetClass, 'K', NULL, NULL,
    PutTokenHelp, NULL},
  {"EDIT_SPEC", &xmPushButtonWidgetClass, 'E', NULL, NULL,
    EditSpecHelp, NULL},
  {"SHOW_SPEC", &xmPushButtonWidgetClass, 'S', NULL, NULL,
    ShowSpecHelp, NULL},
  {"FIRING", &xmPushButtonWidgetClass, 'F', NULL, NULL,
    FireHelp, NULL},
  {"DELETE_OBJECT", &xmPushButtonWidgetClass, 'D', NULL, NULL,
    DeleteHelp, NULL},
  {"UNDO", &xmPushButtonWidgetClass, 'U', NULL, NULL,
    UndoHelp, NULL},
  NULL,
};

static MenuItem  file_items[] = { /* items of File */
  {"New", &xmPushButtonWidgetClass, 'N', NULL, NULL,
    OpenNewFile, NULL},
  {"Open ...", &xmPushButtonWidgetClass, 'O', NULL, NULL,
    PopUpFileSelector, NULL},
  {"", &xmSeparatorGadgetClass, NULL, NULL, NULL, NULL, NULL},
  {"Save", &xmPushButtonWidgetClass, 'S', "Ctrl<Key>S", "Ctrl-S",
    SelectSaveFileOption, NULL},
  {"Save As ...", &xmPushButtonWidgetClass, 'A', NULL, NULL,
    SaveFileAs, NULL},
  {"", &xmSeparatorGadgetClass, NULL, NULL, NULL, NULL, NULL},
  {"Quit", &xmPushButtonWidgetClass, 'Q', "Ctrl<Key>Q",
    "Ctrl-Q", CloseFile, NULL},
  NULL,
};


static MenuItem edit_items[] = { /* items of Edit */
  {"Undo", &xmPushButtonGadgetClass, 'U', NULL, NULL,
    UndoLastDraw, NULL},
  {"DeleteObject", &xmPushButtonGadgetClass, 'D', NULL, NULL,
    SetDeleteObj, NULL},
  {"EditLabel ...", &xmPushButtonWidgetClass, 'E', NULL, NULL,
    GetLabelInfo, NULL},
  NULL,
};

  /* registers the callback data */
  fire_items[0].callback_data = (caddr_t)data;
  edit_items[0].callback_data = (caddr_t)data;
  edit_items[1].callback_data = (caddr_t)data;
  edit_items[2].callback_data = (caddr_t)data;

  /* creates a Motif file selection widget and registers callback data */
  file_selector = CreateFileSelector(parent,data);
  file_items[0].callback_data = (caddr_t)data;
  file_items[1].callback_data = (caddr_t)file_selector;
  file_items[3].callback_data = (caddr_t)data;
  file_items[4].callback_data = (caddr_t)data;
  file_items[6].callback_data = (caddr_t)data;
  /* creates a menu bar for the pull-down menus */
  n = 0;
  XtSetArg(wargs[n], XmNshadowThickness, 0); n++;
  w = XmCreateMenuBar(parent, "MenuBar", wargs, n);
  /* creates the pull-down menus */
  (void) BuildPulldownMenu(w, "File", 'F', file_items);
  (void) BuildPulldownMenu(w, "Edit", 'E', edit_items);
  (void) BuildPulldownMenu(w, "Firing", 'r', fire_items);
  (void) BuildPulldownMenu(w, "Help", 'H', help_items);
  XtManageChild(w);
  return w;
}
```

```
/*
CreateFileSelector() creates a Motif file selection dialog box.
*/

Widget CreateFileSelector(w, data)
Widget   w;
graphics_data *data;
{
  Widget parent = w;
  static Widget  file_selector;
  Arg     wargs[MAX_ARGS];
  XmString str;
  int      n;

  while(!XtIsWMShell(parent))
    parent = XtParent(parent);
  str = XmStringCreateSimple("File Selection");
  n = 0;
  XtSetArg(wargs[n],XmNdialogTitle, str); n++;
  XtSetArg(wargs[n],XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;
  XtSetArg(wargs[n],XmNtextColumns, 40); n++;
  file_selector = XmCreateFileSelectionDialog(parent,"FILES",wargs,n);
  XtUnmanageChild(XmFileSelectionBoxGetChild(file_selector,
                                             XmDIALOG_HELP_BUTTON));
  XtAddCallback(file_selector, XmNokCallback, FileSelectionDone, data);
  XtAddCallback(file_selector, XmNcancelCallback, FileSelectionCancel,data);
  return file_selector;
}


/*
FileSelectionCancel() is a callback function associated with the
cancel button of the file selection dialog box.
*/

void FileSelectionCancel(w, data, call_data)
Widget     w;
graphics_data  *data;
XmFileSelectionBoxCallbackStruct  *call_data;
{
  ResetCurrentAction(data); /* resets the current action */
  strcpy(action_selected,"NOTHING");
  RewriteAction(); /* displays the action selected */
  XtUnmanageChild(w); /* pops down the dialog box */
}


/*
FileSelectionDone() is a callback function associated with the
done button of the file selection dialog box.
*/

void FileSelectionDone(w, data, call_data)
Widget   w;
graphics_data  *data;
XmFileSelectionBoxCallbackStruct  *call_data;
{
  FILE  *fp;
  char  *mask, filename[100], inverse[100], mesg[MSG_LENGTH];
  int   i, j;

  ResetCurrentAction(data);
  /* if nothing is selected as filename */
  if(!XmStringGetLtoR(call_data->value,XmSTRING_DEFAULT_CHARSET,&mask)) {
    strcpy(mesg,GetFileMsgStr("NULL_FILE"));
    PutErrorDialog(w,mesg);
    XtUnmanageChild(w);
    XtFree(mask);
    return;
  }
  /* if file cannot be opened */
  if(OpenFile(mask,data) == -1) {
    strcpy(mesg,GetFileMsgStr("FILE_CANNOT_OPEN"));
    PutErrorDialog(w,mesg);
    XtFree(mask);
    return;
  }
  /* if file is not in the correct format */
  else if(OpenFile(mask,data) == 0) {
    strcpy(mesg,GetFileMsgStr("FILE_FORMAT_ERR"));
```

```
    PutErrorDialog(w,mesg);
    XtFree(mask);
    return;
  }
  /* extracting the filename */
  j = 0;
  for(i = strlen(mask); i > 0 ; i--) {
    if(mask[i-1] == '/')
      i = 0;
    else
      inverse[j++] = mask[i-1];
  }
  inverse[j] = '\0';
  j = 0;
  for(i = strlen(inverse); i > 0; i--)
    filename[j++] = inverse[i-1];
  filename[j] = '\0';
  strcpy(GLOBAL_FILE_NAME,filename); /* updates the filename */
  RewriteFileName(); /* displays the filename */
  RedrawDrawingBoard(data);
  XtFree(mask);
  strcpy(action_selected,"NOTHING");
  RewriteAction();
  XtUnmanageChild(w); /* pops down the dialog box */
}


/*
SetDeleteObj() as a callback function assigns DeleteObject() function
to the current function, whenever the option Delete Object is selected
from the pull-down menu "Edit".
*/

void SetDeleteObj(w, data, call_data)
Widget   w;
graphics_data  *data;
XtPointer      call_data;
{
  ResetCurrentAction(data); /* resets the current action */
  strcpy(data->object,"DEL_OBJECT");
  data->current_func = DeleteObject; /* assigns the function */
  strcpy(action_selected,"DELETE OBJECT");
  RewriteAction();
  SetPirateCursor(drawing); /* sets cursor shape to pirate sign */
}


/*
GetStartPos() is an event handler function. It is called whenever a
mouse button is pressed within the drawing area. Depending upon
which button is pressed and the current action, an action takes place.
*/

void GetStartPos(w, data, event)
Widget             w;
graphics_data *data;
XEvent        *event;
{
  TRANSITION     *t_temp;
  PLACE    *p_temp;
  char     token_label[20];
  char     link_label[100];

  if((data->current_func)&&(event->xbutton.button == Button1)) {
    /*
    If the current object is PLACE and button 1 is pressed, a
    place is made, i.e., a circle with a default label.
    */
    if (strcmp(data->object,"PLACE") == 0) {
      strcpy(data->object,"START_PLACE");
      data->x1 = event->xbutton.x - 20;
      data->y1 = event->xbutton.y - 20;
      data->x2 = data->x1 + 40;
      data->y2 = data->y1 + 40;
      data->height = 40;
      data->width  = 40;
      sprintf(data->label, "p%d",PLACE_CNT+1);
      DrawPlace(w, data->xorgc, data->x1, data->y1, data->height,
                data->width, data); /* draws a place */
    }
    /*
```

```
If the current object is DEL_OBJECT and button 1 is pressed, a
selected object is deleted.
*/
else if(strcmp(data->object,"DEL_OBJECT") == 0) {
  data->x1 = data->x2 = event->xbutton.x;
  data->y1 = data->y2 = event->xbutton.y;
  (*(data->current_func))(w, data);
}
/*
If the current object is STEPPED_FIRE and button 1 is pressed, a
selected transition is fired.
*/
else if (strcmp(data->object,"STEPPED_FIRE") == 0) {
  data->x1 = data->x2 = event->xbutton.x;
  data->y1 = data->y2 = event->xbutton.y;
  (*(data->current_func))(data);
}
/*
If the current object is EDIT_SPEC and button 1 is pressed, a
dialog box pops up, displaying the specification of the selected
node. User can edit the interpretation of the node.
*/
else if (strcmp(data->object,"EDIT_SPEC") == 0) {
  data->x1 = data->x2 = event->xbutton.x;
  data->y1 = data->y2 = event->xbutton.y;
  (*(data->current_func))(data,EDIT_SPEC);
}
/*
If the current object is SHOW_SPEC and button 1 is pressed, a
dialog box pops up, displaying the specification of the selected
node. User cannot edit the interpretation of the node.
*/
else if (strcmp(data->object,"SHOW_SPEC") == 0) {
  data->x1 = data->x2 = event->xbutton.x;
  data->y1 = data->y2 = event->xbutton.y;
  (*(data->current_func))(data,SHOW_SPEC);
}
/*
If the current object is LINK and button 1 is pressed, the selected
node is checked for the source of the link. If the selected node
is a valid source, a bell is sounded.
*/
else if(strcmp(data->object,"LINK") == 0) {
  data->x1 = data->x2 = event->xbutton.x;
  data->y1 = data->y2 = event->xbutton.y;
  if(BellIfLinkSourceOk(data))
    LINK_CNT++;
}
else if(strcmp(data->object,"START_LINK") == 0) {
  data->current_func = DrawLink;
  (*(data->current_func))(w, data->gc,
                          data->x1, data->y1,
                          data->x2, data->y2);
  /* if the place selected is the source i.e., postset is to be updated */
  if(P_LIST->pre_post == POSTSET)
    sprintf(link_label,"PLACE=>~~%s~~TRANSITION=>~~%s~~LINK%d~~",
                          P_LIST->place_pre_selected,
                          T_LIST->trans_pre_selected,
                          LINK_CNT);
  else if(P_LIST->pre_post == PRESET)
    sprintf(link_label,"TRANSITION=>~~%s~~PLACE=>~~%s~~LINK%d~~",
                                T_LIST->trans_pre_selected,
                                P_LIST->place_pre_selected,
                                LINK_CNT);
  strcpy(data->label,link_label);
  save_object(data);
  data->x1  =  data->buffer[data->next_pos-1].x2;
  data->x2  =  event->xbutton.x;
  data->y1  =  data->buffer[data->next_pos-1].y2;
  data->y2  =  event->xbutton.y;
  (*(data->current_func))(w, data->xorgc,
                          data->x1, data->y1,
                          data->x2, data->y2);
  data->current_func = DrawLink;
  strcpy(data->object,"START_LINK");
}
else if(strcmp(data->object,"TRANSITION") == 0) {
  data->x1 = data->x2 = event->xbutton.x;
  data->y1 = data->y2 = event->xbutton.y;
  sprintf(data->label, "t%d",TRANSITION_CNT+1);
  strcpy(data->object,"START_TRANS");
```

```
  }
}
/* button 2 of the mouse is pressed */
else if(event->xbutton.button == Button2) {
  /*
  If the current object is START_PLACE, a circle is drawn with
  a default label
  */
  if(strcmp(data->object,"START_PLACE") == 0) {
    /* erases the previously drawn circle for the rubber band effect */
    DrawPlace(w, data->xorgc, data->x1, data->y1, data->height,
              data->width, data);
    strcpy(data->object,"PLACE");
    UNDO_ACTION = UNDO_DEL_PLACE; /* updates the undo action */
    p_temp = GetNewPlace(); /* gets a new place node */
    strcpy(data->label,p_temp->label);
    data->x1 = event->xbutton.x - 20;
    data->y1 = event->xbutton.y - 20;
    data->x2 = data->x1 + 40;
    data->y2 = data->y1 + 40;
    data->height = 40;
    data->width  = 40;
    save_object(data);
    DrawPlace(w, data->gc, data->x1, data->y1, data->height,
              data->width, data);
    data->x1 = data->x1 + 5;
    data->y1 = data->y1 + 25;
    data->x2 = 0;
    data->y2 = 0;
    /* displays the number of tokens */
    data->current_func = DrawToken;
    strcpy(data->object,"PUT_TOKEN");
    strcpy(data->label,p_temp->label);
    save_object(data);
    DrawToken(data->x1, data->y1, data);
    strcpy(data->object,"PLACE");
    P_LIST->last_token = 0;
  }
  /*
  If the current object is START_TRANS, a transition is drawn with
  a default label
  */
  else if(strcmp(data->object,"START_TRANS") == 0) {
    strcpy(data->object,"TRANSITION");
    /* erase the previously drawn transition for the rubber band effect */
    DrawTransition(w, data->xorgc, data->x1, data->y1,
                   data->x2, data->y2, data, "NO_LABEL");
    UNDO_ACTION = UNDO_DEL_TRANS; /* updates the undo action */
    t_temp = GetNewTrans(); /* gets a new transition */
    strcpy(data->label, t_temp->label);
    data->x2  =  event->xbutton.x;
    data->y2  =  event->xbutton.y;
    /* checks the default size of the transition */
    if(abs(data->x2-data->x1) >= abs(data->y2-data->y1)) {
      if(abs(data->x2-data->x1) < 8) {
        if(data->x2 > data->x1)
          data->x2 = data->x1 + 8;
        else
          data->x2 = data->x1 - 8;
      }
      if(data->y2 > data->y1)
        data->y2 = data->y1 + 8;
      else
        data->y2 = data->y1 - 8;
    }
    else if(abs(data->y2-data->y1) >= abs(data->x2-data->x1)) {
      if(abs(data->y2-data->y1) < 8) {
        if(data->y2 > data->y1)
          data->y2 = data->y1 + 8;
        else
          data->y2 = data->y1 - 8;
      }
      if(data->x2 > data->x1)
        data->x2 = data->x1 + 8;
      else
        data->x2 = data->x1 - 8;
    }
    save_object(data); /* saves the currently drawn object */
    /* draws a transition at current position */
    DrawTransition(w, data->gc,data->x1, data->y1,
                   data->x2, data->y2, data, "PUT_LABEL");
```

```
    }
    /*
    If the current object is START_LINK, the current drawing action of a
    link is cancelled
    */
    else if(strcmp(data->object,"START_LINK") == 0){
      RemoveIncompleteLink(data); /* removes the incomplete links */
      RedrawDrawingBoard(data);
      data->current_func  = NULL; /* updates the current action */
      strcpy(data->object,"");
      strcpy(action_selected,"NOTHING");
      RewriteAction();
      return;
    }
  }
  /*
  If button 3 of the mouse is pressed inside the drawing area and no
  object is being drawn, an undo action is taken
  */
  else if(event->xbutton.button == Button3) {
    if((strcmp(data->object,"START_TRANS") != 0) &&
       (strcmp(data->object,"START_LINK") != 0)   &&
       (strcmp(data->object,"START_PLACE") != 0)){
      UndoLastDraw(w, data);
    }
  }
}


/*
RemoveIncompleteLink() removes any incomplete link from the buffer
of the objects.
*/

void RemoveIncompleteLink(data)
graphics_data   *data;
{
  int    i;
  char   link_num[30];

  data->current_func  = NULL;
  strcpy(data->object,"");
  sprintf(link_num,"LINK%d",LINK_CNT);
  for(i = data->next_pos-1 ; i >= 0; i--) {
    if(!(strstr(data->buffer[i].label,link_num))) {
      return;
    }
    else {
      data->next_pos--;
    }
  }
  LINK_CNT--;
}

/*
TrackMouse() is an event handler function. It is called whenever a
mouse is moved within the drawing area. Depending upon the value of
the current object, an action takes place.
*/

void TrackMouse(w, data, event)
Widget          w;
graphics_data  *data;
XEvent         *event;
{
  DEL_ITEM  *temp_item,
            *free_item;
  char     link_label[100];
  if(data->current_func){
    /* if the object is START_PLACE */
    if(strcmp(data->object,"START_PLACE") == 0) {
      /* erases the previously drawn place for the rubber band effect */
      DrawPlace(w, data->xorgc, data->x1, data->y1, data->height,
                data->width, data);
      data->x1 = event->xbutton.x - 20;
      data->y1 = event->xbutton.y - 20;
      data->x2 = data->x1 + 40;
      data->y2 = data->y1 + 40;
      data->height = 40;
      data->width  = 40;
      /* draws a place at current position */
```

```
  DrawPlace(w, data->xorgc, data->x1, data->y1, data->height,
            data->width, data);
}
/* if the object is START_LINK */
else if(strcmp(data->object,"START_LINK") == 0) {
  /* erases the previously drawn link for the rubber band effect */
  (*(data->current_func))(w, data->xorgc, data->x1, data->y1,
                          data->x2, data->y2);
  /* updates the last point */
  data->x2  =  event->xbutton.x;
  data->y2  =  event->xbutton.y;
  /* draws a link at current position */
  (*(data->current_func))(w, data->xorgc, data->x1, data->y1,
                          data->x2, data->y2);
  /* checks if the link hits the destination */
  if(BellIfLinkDestOk(data)) {
    data->current_func = DrawLink;
    /* erases the previously drawn link for the rubber band effect */
    (*(data->current_func))(w, data->xorgc, data->x1, data->y1,
                            data->x2, data->y2);
    UNDO_ACTION = UNDO_DEL_LINK;
    sprintf(link_label,"LINK%d",LINK_CNT);
    strcpy(T_LIST->del_link,link_label);
    data->current_func  = DrawArrowHeadLink;
    /* draws a final link at current position with an arrow head */
    DrawArrowHeadLink(w, data->gc, data->x1, data->y1, data->x2, data->y2);
    strcpy(data->object,"LINK");
    SaveLinkInfo(); /* saves the link information */
    strcpy(link_label,"");
    if(P_LIST->pre_post == POSTSET)
      sprintf(link_label,"PLACE=>~~%s~~TRANSITION=>~~%s~~LINK%d~~",
                          P_LIST->place_selected,
                          T_LIST->trans_selected,
                          LINK_CNT);
    else if(P_LIST->pre_post == PRESET)
      sprintf(link_label,"TRANSITION=>~~%s~~PLACE=>~~%s~~LINK%d~~",
                                  T_LIST->trans_selected,
                                  P_LIST->place_selected,
                                  LINK_CNT);
    strcpy(data->label,link_label);
    save_object(data); /* saves the object in the buffer */
    RedrawDrawingBoard(data);
    data->current_func  = NULL;
    strcpy(data->object,"END_LINK");
    temp_item = DEL_LIST; /* frees any previously saved deleted link */
    while(temp_item != NULL) {
      free_item = temp_item;
      temp_item = temp_item->next;
      free(free_item);
    }
    DEL_LIST = NULL;
  }
}
else if(strcmp(data->object,"START_TRANS") == 0) {
  /* erases the previously drawn transition for the rubber band effect */
  DrawTransition(w, data->xorgc, data->x1, data->y1,
                 data->x2, data->y2, data, FALSE);
  data->x2  =  event->xbutton.x;
  data->y2  =  event->xbutton.y;
  /* checks the default size of the transition */
  if(abs(data->x2-data->x1) >= abs(data->y2-data->y1)) {
    if(abs(data->x2-data->x1) < 8) {
      if(data->x2 > data->x1)
        data->x2 = data->x1 + 8;
      else
        data->x2 = data->x1 - 8;
    }
    if(data->y2 > data->y1)
      data->y2 = data->y1 + 8;
    else
      data->y2 = data->y1 - 8;
  }
  else if(abs(data->y2-data->y1) >= abs(data->x2-data->x1)) {
    if(abs(data->y2-data->y1) < 8) {
      if(data->y2 > data->y1)
        data->y2 = data->y1 + 8;
      else
        data->y2 = data->y1 - 8;
    }
    if(data->x2 > data->x1)
      data->x2 = data->x1 + 8;
```

```
        else
          data->x2 = data->x1 - 8;
      }
      /* draws a transition at the current position */
      DrawTransition(w, data->xorgc, data->x1, data->y1,
                     data->x2, data->y2, data, FALSE);
    }
  }
}


/*
Activate() is a callback function. It is called whenever a button provided
on the left panel is pressed. Depending upon which button is pressed, an
action takes place.
*/

void Activate(w, data, call_data)
Widget          w;
graphics_data  *data;
XmToggleButtonCallbackStruct  *call_data;
{
  user_data *select;
  Arg   wargs[5];
  int   n;

  if(!call_data->set) return;
  ResetCurrentAction(data); /* resets the current action */
  n = 0;
  XtSetArg(wargs[n], XmNuserData, &select); n++;
  XtGetValues(w, wargs, n);
  strcpy(data->object,select->object);
  ResetCursor(toplevel); /* resets cursor shapes */
  ResetCursor(drawing);
  if(strcmp(data->object,"LINK") == 0) { /* if the object is LINK */
    SetBusyCursor(toplevel);
    strcpy(action_selected,"DRAW LINK"); /* updates the action selected */
    RewriteAction();
    GetLinkInfo(w, data); /* pops up a dialog box for the link definition */
  }
  else if(strcmp(data->object,"PUT_TOKEN") == 0) { /* if the object is PUT_TOKEN */
    SetBusyCursor(toplevel); /* sets cursor to a watch sign */
    strcpy(action_selected,"PUT TOKEN"); /* updates the action selected */
    RewriteAction();
    GetTokenInfo(w, data); /* pops up a dialog box */
  }
  else if(strcmp(data->object,"PLACE") == 0) { /* if the object is PLACE */
    strcpy(action_selected,"DRAW PLACE"); /* updates the action selected */
    RewriteAction();
    data->current_func = select->func; /* assigns a function */
  }
  else if(strcmp(data->object,"TRANSITION") == 0){ /* if the object is TRANSITION */
    strcpy(action_selected,"DRAW TRANSITION"); /* updates the action selected */
    RewriteAction();
    data->current_func = select->func; /* assigns a function */
  }
  else if(strcmp(data->object,"EDIT_SPEC") == 0) { /* if the object is EDIT_SPEC */
    strcpy(action_selected,"EDIT NODE SPECIFICATION");
    RewriteAction();
    SetHandCursor(drawing);
    data->current_func = select->func; /* assigns a function */
  }
  else if(strcmp(data->object,"SHOW_SPEC") == 0) { /* if the object is SHOW_SPEC */
    strcpy(action_selected,"SHOW NODE SPECIFICATION");
    RewriteAction();
    SetHandCursor(drawing);
    data->current_func = select->func; /* assigns a function */
  }
}


/*
BuildPulldownMenu() builds a pull-down menu system.
*/

Widget  BuildPulldownMenu(parent, title, mnemonic, items)
Widget  parent;
char    *title, mnemonic;
MenuItem  *items;
{
  Widget   pull_down, cascade, w;
```

```
  int      i;
  XmString  str;

  pull_down = XmCreatePulldownMenu(parent, "_pulldown", NULL, 0);
  str = XmStringCreateSimple(title);
  cascade = XtVaCreateManagedWidget(title, xmCascadeButtonWidgetClass, parent,
                                    XmNsubMenuId,  pull_down,
                                    XmNlabelString, str,
                                    XmNshadowThickness, 4,
                                    XmNmnemonic,    mnemonic,
                                    NULL);
  XmStringFree(str);

  for(i = 0; items[i].label != NULL; i++) {
    w = XtVaCreateManagedWidget(items[i].label, *items[i].class, pull_down,
                                XmNshadowThickness, 4,
                                NULL);
    if(items[i].mnemonic)
      XtVaSetValues(w, XmNmnemonic, items[i].mnemonic, NULL);
    if(items[i].accelerator) {
      str = XmStringCreateSimple(items[i].accel_text);
      XtVaSetValues(w, XmNaccelerator, items[i].accelerator,
                    XmNacceleratorText, str, NULL);
      XmStringFree(str);
    }
    if(items[i].callback) {
      XtAddCallback(w, XmNactivateCallback, items[i].callback,
                    items[i].callback_data);
    }
  }
  return cascade;
}


/*
Refresh() is a callback function registered for the widget drawing. It is
called when the drawing widget get exposed.
*/

void Refresh(w, data, call_data)
Widget        w;
graphics_data *data;
caddr_t       call_data;
{
  if(strcmp(data->object,"START_TRANS") == 0) {
    strcpy(data->object,"TRANSITION");
  }
  else if(strcmp(data->object,"START_PLACE") == 0) {
    strcpy(data->object,"PLACE");
  }
  RedrawDrawingBoard(data);
}


/*
File name:   alloc_node.h
Description: It includes the header files and declares the functions defined
             in the file alloc_node.c. */

#include "data_str.h"
#include "global.h"

/*
The file alloc_node.c consists of following functions.
For details, refer explanation before the declaration of each
function.
*/
PLACE      *GetNewPlace();
TRANSITION *GetNewTrans();


/*
File name:   alloc_node.c
Description: It contains the functions which dynamically allocate the memory
             for the nodes PLACE and TRANSITION.
*/

#include "alloc_node.h"

/*
GetNewPlace() dynamically allocates the memory needed for the node
```

```
PLACE and initializes its members. It also increments the PLACE_CNT,
assigns a default label to currently created place and puts this
place in the link list of the places, i.e., P_LIST.
Value returned: (PLACE *).
*/

PLACE *GetNewPlace()
{
  PLACE    *p_temp;
  char     plabel[LABEL_LENGTH]; /* label for the place */

  if((p_temp = (PLACE *)malloc(sizeof(PLACE))) == NULL) {
    printf("Out Of Memory\n");
    exit(0);
  }
  p_temp->next = NULL;  /* initializes the members of the structure PLACE */
  p_temp->prev = NULL;
  p_temp->cur_preset_no  = 0;
  p_temp->cur_postset_no = 0;
  p_temp->no_of_tokens   = 0;
  p_temp->freq_in_preset = 0;
  strcpy(p_temp->interpretation,"");
  PLACE_CNT++; /* increments PLACE_CNT */
  sprintf(plabel, "p%d",PLACE_CNT); /* creates a default label */
  strcpy(p_temp->label, plabel);
  strcpy(P_LIST->place_selected, plabel);
  if(P_LIST->HEAD == NULL)  /* updates the link list of the places */
    P_LIST->HEAD = p_temp;
  else {
    p_temp->next = P_LIST->HEAD;
    P_LIST->HEAD->prev = p_temp;
    P_LIST->HEAD = p_temp;
  }
  return(p_temp);  /* returns a pointer to a new place */
}


/*
GetNewTrans() dynamically allocates the memory needed for the node
TRANSITION and initializes its members. It also increments the
TRANSITION_CNT, assigns a default label to currently created transition
and puts this transition in the link list of the transition, i.e.,
T_LIST.
Value returned: (TRANSITION *).
Same inline comments as in GetNewPLACE().
*/

TRANSITION *GetNewTrans()
{
  TRANSITION  *t_temp;
  char        tlabel[LABEL_LENGTH];

  if((t_temp = (TRANSITION *)malloc(sizeof(TRANSITION))) == NULL) {
    printf("Out Of Memory\n");
    exit(0);
  }
  t_temp->next = NULL;
  t_temp->prev = NULL;
  t_temp->cur_preset_no  = 0;
  t_temp->cur_postset_no = 0;
  t_temp->enabled        = NO;
  strcpy(t_temp->interpretation,"");
  TRANSITION_CNT++;
  sprintf(tlabel, "t%d",TRANSITION_CNT);
  strcpy(t_temp->label, tlabel);
  strcpy(T_LIST->trans_selected, tlabel);
  if(T_LIST->HEAD == NULL)
    T_LIST->HEAD = t_temp;
  else {
    t_temp->next = T_LIST->HEAD;
    T_LIST->HEAD->prev = t_temp;
    T_LIST->HEAD = t_temp;
  }
  return(t_temp);
}


/*
File name:   open_file.h
Description: It contains the header files required for the file open_file.c.
             It also declares the external functions and the functions
```

```
            defined in the file open_file.c
*/

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/MessageB.h>
#include <Xm/Text.h>
#include <Xm/PushB.h>
#include <Xm/BulletinB.h>
#include <stdio.h>
#include "data_str.h"
#include "global.h"


/*
These externally defined functions are called in the functions
defined in the file open_file.c.
*/
extern void DrawPlace();            /* defined in draw_object.c */
extern void DrawTransition();       /* defined in draw_object.c */
extern void DrawToken();            /* defined in draw_object.c */
extern void DrawLink();             /* defined in draw_object.c */
extern void DrawArrowHeadLink();    /* defined in draw_object.c */
extern void AllTrim();              /* defined in utility.c */
extern void ResetCurrentAction();   /* defined in utility.c */
extern void RewriteAction();        /* defined in utility.c */
extern void RedrawDrawingBoard();   /* defined in utility.c */
extern void RemoveIncompleteLink(); /* defined in utility.c */
extern void ResetCursor();          /* defined in cursor.c */
extern void PreSaveFile();          /* defined in save.c */
extern void SaveFile();             /* defined in save.c */

/*
The file open_file.c consists of following functions.
For details, refer explanation before the declaration of each
function.
*/
void PopUpFileSelector();
void CancelOpenNew();
void NoSaveOpenNew();
void SaveOldOpenNew();
void InitEnvironment();
void FreePlaceList();
void FreeTransList();
int  OpenFile();
void AssignFunction();
void FileOpeningWarn();
void DoneFileOpeningWarn();


/*
File name:   open_file.c
Description: It contains the functions related to the operation of opening
             a file.
*/

#include "open_file.h"

/*
PopUpFileSelector() as a callback function pops up the file selection
dialog box. If any object is drawn, it puts warning message about
saving the current file.
*/

void PopUpFileSelector(w,file_selector,call_data)
Widget   w;
Widget   file_selector;
XtPointer call_data;
{
  if(UNDO_ACTION != NO) { /* warning message */
  FileOpeningWarn(w,"Save the changes in current file.\nOpening a file wouldn't save the
changes.\nWhenever you change the path, click Filter to update the
change.",file_selector);
  }
  else {
  strcpy(action_selected,"OPEN FILE"); /* updates the action selected */
  RewriteAction();
  ResetCursor(drawing); /* resets the cursor */
  XtManageChild(file_selector); /* pops up the file selection dialog box */
  }
}
```

```
/*
OpenNewFile(), as a callback function pops up a dialog box titled
Open New File.
*/

void OpenNewFile(w, data, call_data)
Widget   w;
graphics_data  *data;
XtPointer call_data;
{
  static Widget   open_file_dialog = NULL;
  Arg      wargs[MAX_ARGS];
  int      n;
  XmString str1, str2, str3, str4, str5;

  ResetCurrentAction(data);
  if(data->next_pos == 0) { /* if nothing is drawn */
    InitEnvironment(data); /* initializes the system for a new file */
    strcpy(action_selected,"NOTHING"); /* updates the action selected */
    RewriteAction();
    ResetCursor(drawing); /* resets the cursor */
    return;
  }
  strcpy(action_selected,"OPEN NEW FILE"); /* updates the action selected */
  RewriteAction();
  ResetCursor(drawing);
  if(!open_file_dialog) {
    /* creates a dialog box titled Open New File */
    n = 0;
    str1 = XmStringCreate(" Open New File ",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNdialogTitle,str1); n++;
    str2 = XmStringCreate("Save Current file",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNmessageString,str2); n++;
    str3 = XmStringCreate("YES",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNokLabelString,str3); n++;
    str4 = XmStringCreate("NO",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNcancelLabelString,str4); n++;
    str5 = XmStringCreate("CANCEL",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNhelpLabelString,str5); n++;
    XtSetArg(wargs[n],XmNmessageAlignment,XmALIGNMENT_BEGINNING);
    XtSetArg(wargs[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL); n++;
    XtSetArg(wargs[n],XmNnoResize,True); n++;
    XtSetArg(wargs[n],XmNautoUnmanage,False); n++;
    XtSetArg(wargs[n],XmNdefaultButtonType,XmDIALOG_HELP_BUTTON); n++;
    open_file_dialog = XmCreateErrorDialog(w, "open_file_dialog",wargs,n);
    XmStringFree(str1);
    XmStringFree(str2);
    XmStringFree(str3);
    XmStringFree(str4);
    XmStringFree(str5);
    XtAddCallback(open_file_dialog, XmNokCallback, SaveOldOpenNew, data);
    XtAddCallback(open_file_dialog, XmNcancelCallback, NoSaveOpenNew, data);
    XtAddCallback(open_file_dialog, XmNhelpCallback, CancelOpenNew, NULL);
    XtManageChild(open_file_dialog);
  }
  else {
    XtManageChild(open_file_dialog);
  }
}


/*
CancelOpenNew() is a callback function registered with the CANCEL
button of the dialog box, titled Open New File. It cancels the operation
and pops down the dialog box.
*/

void CancelOpenNew(w, client_data, call_data)
Widget  w;
XtPointer   client_data;
XtPointer   call_data;
{
  strcpy(action_selected,"NOTHING"); /* updates the action selected */
  RewriteAction();
  ResetCursor(drawing); /* resets the cursor */
  XtUnmanageChild(w); /* pops down the dialog box */
}


/*
```

```
NoSaveOpenNew() is a callback function registered with the NO
button of the dialog box, titled Open New File. User selects NO when
the current file is not to be saved.
*/

void NoSaveOpenNew(w, data, call_data)
Widget  w;
graphics_data  *data;
XtPointer   call_data;
{
  InitEnvironment(data); /* initializes the system */
  strcpy(action_selected,"NOTHING"); /* updates the action selected */
  RewriteAction();
  ResetCursor(drawing); /* resets the cursor */
  XtUnmanageChild(w); /* pops down the dialog */
}


/*
SaveOldOpenNew() is a callback function registered with the YES
button of the dialog box titled Open New File. User selects YES when
the current file is to be saved. If a filename is already given, it
saves and opens a new file. Otherwise it calls the function PreSaveFile()
to display the dialog box for the save option.
*/

void SaveOldOpenNew(parent, data, call_data)
Widget         parent;
graphics_data  *data;
XtPointer      call_data;
{
  /* if a filename is previously given, it is saved and a new file is opened */
  if(strcmp(GLOBAL_FILE_NAME,"Untitled") != 0) {
    SaveFile(data); /* save the current file */
    InitEnvironment(data); /* initializes the system */
    strcpy(action_selected,"NOTHING"); /* updates the action selected */
    RewriteAction();
    ResetCursor(drawing); /* resets the cursor */
    XtUnmanageChild(parent); /* pops down the dialog */
    return;
  }
  PreSaveFile(parent, data);  /* call to save the file */
}


/*
InitEnvironment() initializes the environment for a new file. It
frees up the memory and initializes the variables.
*/

void InitEnvironment(data)
graphics_data  *data;
{
  FreePlaceList(); /* frees up the memory associated with the P_LIST */
  FreeTransList(); /* frees up the memory associated with the T_LIST */
  UNDO_ACTION = NO;
  LINK_CNT = 0;
  strcpy(GLOBAL_FILE_NAME,"Untitled");
  data->next_pos = 0;
  data->current_func = NULL;
  strcpy(data->object,"");
  strcpy(data->label,"");
  RedrawDrawingBoard(data); /* clears the drawing area */
  RewriteFileName();
  ResetCursor(drawing);
  strcpy(action_selected,"NOTHING");
  RewriteAction();
}


/*
FreePlaceList() frees up the memory associated with the linked
list for the places.
*/

void FreePlaceList()
{
  PLACE    *p_temp1,
           *p_temp2;

  p_temp1 = P_LIST->HEAD;
```

```
  if(p_temp1 == NULL)  return;

  while(p_temp1->next != NULL)
    p_temp1 = p_temp1->next;

  while(p_temp1 != NULL) {
    p_temp2 = p_temp1;
    p_temp1 = p_temp1->prev;
    free(p_temp2);
  }
  PLACE_CNT = 0;
  strcpy(P_LIST->place_pre_selected,"");
  strcpy(P_LIST->place_selected,"");
  P_LIST->pre_post = NULL;
  P_LIST->pre_post_index = -1;
  P_LIST->HEAD = NULL;
}


/*
FreeTransList() frees up the memory associated with the linked
list for the transitions.
*/

void FreeTransList()
{
  TRANSITION   *t_temp1,
               *t_temp2;

  t_temp1 = T_LIST->HEAD;

  if(t_temp1 == NULL)  return;
  while(t_temp1->next != NULL)
    t_temp1 = t_temp1->next;
  while(t_temp1 != NULL) {
    t_temp2 = t_temp1;
    t_temp1 = t_temp1->prev;
    free(t_temp2);
  }
  TRANSITION_CNT = 0;
  strcpy(T_LIST->trans_pre_selected,"");
  strcpy(T_LIST->trans_selected,"");
  T_LIST->pre_post = NULL;
  T_LIST->pre_post_index = -1;
  T_LIST->HEAD = NULL;
}


/*
OpenFile() reads in a input file.
*/

int OpenFile(filename, data)
char  *filename;
graphics_data  *data;
{
  FILE   *fp;
  FILE   *fp1;
  PLACE        *p_temp;
  TRANSITION   *t_temp;
  char         buffer[INFO_LENGTH],
               *p;
  int i;

  if(!(fp1 = fopen(filename,"r"))) {
    printf("Cannot open file \n");
    return(-1);
  }

  /* opens an intermediate file to save the encoded information */
  if(!(fp = fopen("PETRINETTEMPFILE","w+"))) {
    printf("Cannot open file \n");
    return(-1);
  }
  EncodeFile(fp,fp1); /* encodes the input file */
  fclose(fp);
  fclose(fp1);
  if(!(fp = fopen("PETRINETTEMPFILE","r"))) {
    printf("Cannot open file \n");
    system("rm PETRINETTEMPFILE"); /* removes the intermediate file */
    return(-1);
```

```
}
/* reads the enocoded information from the intermediate file */
fgets(buffer,80,fp);
p =strtok(buffer," \n");
if(strcmp(p,"BUFFER_INFO") != 0) { /* file not in required format */
  fclose(fp);
  system("rm PETRINETTEMPFILE");
  return(0);
}
i = 0;
InitEnvironment(data); /* initializes the environment */
fgets(buffer,80,fp);
p =strtok(buffer," \n");
/* reads the informations from the file */
while((strcmp(p,"PLACE_LIST_INFO") != 0) && (!feof(fp))) {
  AssignFunction(data, i, p);
  data->buffer[i].gc = data->gc;
  strcpy(data->buffer[i].object,p); /* reads an object */

  fgets(buffer,80,fp);
  p =strtok(buffer," \n");
  strcpy(data->buffer[i].label,p); /* reads a label for a place */

  /* reads position of the object */
  fgets(buffer,80,fp);
  p =strtok(p," \n");
  data->buffer[i].x1 = atoi(p);
  fgets(buffer,80,fp);
  p =strtok(buffer," \n");
  data->buffer[i].x2 = atoi(p);
  fgets(buffer,80,fp);
  p =strtok(buffer," \n");
  data->buffer[i].y1 = atoi(p);
  fgets(buffer,80,fp);
  p =strtok(buffer," \n");
  data->buffer[i].y2 = atoi(p);
  i++;
  fgets(buffer,80,fp);
  p =strtok(buffer," \n");
}
data->next_pos = i;
if(strcmp(p,"PLACE_LIST_INFO") != 0) { /* error in reading */
  fclose(fp);
  system("rm PETRINETTEMPFILE");
  InitEnvironment(data);
  return(0);
}
fgets(buffer,80,fp);
p =strtok(buffer," \n");
P_LIST->HEAD = NULL;                    /* adds a place in the P_LIST */
while((strcmp(p,"TRANSITION_LIST_INFO") != 0) && (!feof(fp))) {
  if((p_temp = (PLACE *)malloc(sizeof(PLACE))) == NULL) {
    printf("Out Of Memory\n");
    exit(0);
  }
  /* initializes the members */
  p_temp->next = NULL;
  p_temp->prev = NULL;
  p_temp->cur_preset_no = 0;
  p_temp->cur_postset_no = 0;
  p_temp->no_of_tokens = 0;

  strcpy(p_temp->label, p); /* stores the label */
  strcpy(p_temp->interpretation, "");
  if(P_LIST->HEAD == NULL)  /* updates the P_LIST */
    P_LIST->HEAD = p_temp;
  else {
    p_temp->next = P_LIST->HEAD;
    P_LIST->HEAD->prev = p_temp;
    P_LIST->HEAD = p_temp;
  }
  fgets(buffer,80,fp);
  p =strtok(buffer," \n");
  if(strcmp(p,"PRESET_LIST") != 0) { /* error in reading */
    fclose(fp);
    system("rm PETRINETTEMPFILE");
    InitEnvironment(data);
    return(0);
  }
  fgets(buffer,80,fp);
  p =strtok(buffer," \n");
```

```
    p_temp->cur_preset_no = atoi(p);  /* current number of nodes in the preset */
    for(i = 0; i < p_temp->cur_preset_no; i++) {
      fgets(buffer,80,fp);
      p =strtok(buffer," \n");
      strcpy(p_temp->preset[i],p); /* reads preset */
    }
    fgets(buffer,80,fp);
    p =strtok(buffer," \n");
    if(strcmp(p,"POSTSET_LIST") != 0) { /* error in reading */
      fclose(fp);
      system("rm PETRINETTEMPFILE");
      InitEnvironment(data);
      return(0);
    }
    fgets(buffer,80,fp);
    p =strtok(buffer," \n");
    p_temp->cur_postset_no = atoi(p); /* current number of nodes in the postset */
    for(i = 0; i < p_temp->cur_postset_no; i++) {
      fgets(buffer,80,fp);
      p =strtok(buffer," \n");
      strcpy(p_temp->postset[i],p); /* reads postset */
    }
    fgets(buffer,80,fp);
    p =strtok(buffer," \n");
    p_temp->no_of_tokens = atoi(p); /* number of tokens */
    fgets(buffer,INFO_LENGTH,fp);
    p =strtok(buffer,"\n");

    /* reads the interpretation */
    while((strcmp(p,"END_INTERPRETATION") != 0) && (!feof(fp))) {
      strcat(p_temp->interpretation, p);
      strcat(p_temp->interpretation, "\n");
      fgets(buffer,INFO_LENGTH,fp);
      p =strtok(buffer,"\n");
    }
    AllTrim(p_temp->interpretation); /* trims the interpretation */
    fgets(buffer,80,fp);
    p =strtok(buffer," \n");
  }
  if(strcmp(p,"TRANSITION_LIST_INFO") != 0) { /* error in reading */
    fclose(fp);
    system("rm PETRINETTEMPFILE");
    InitEnvironment(data);
    return(0);
  }
  fgets(buffer,80,fp);
  p =strtok(buffer," \n");
  T_LIST->HEAD = NULL;          /* adds a transition in the T_LIST */
  while((strcmp(p,"GLOBAL_PTL_COUNTS") != 0) && (!feof(fp))) {
    if((t_temp = (TRANSITION *)malloc(sizeof(TRANSITION))) == NULL) {
      printf("Out Of Memory\n");
      exit(0);
    }
    /* initializes the members */
    t_temp->next = NULL;
    t_temp->prev = NULL;
    t_temp->cur_preset_no  = 0;
    t_temp->cur_postset_no = 0;
    t_temp->enabled        = NO;
    strcpy(t_temp->label, p); /* stores the label */
    strcpy(t_temp->interpretation,"");
    if(T_LIST->HEAD == NULL)  /* updates the T_LIST */
      T_LIST->HEAD = t_temp;
    else {
      t_temp->next = T_LIST->HEAD;
      T_LIST->HEAD->prev = t_temp;
      T_LIST->HEAD = t_temp;
    }
    fgets(buffer,80,fp);
    p =strtok(buffer," \n");
    if(strcmp(p,"PRESET_LIST") != 0) { /* error in reading */
      fclose(fp);
      system("rm PETRINETTEMPFILE");
      InitEnvironment(data);
      return(0);
    }
    fgets(buffer,80,fp);
    p =strtok(buffer," \n");
    t_temp->cur_preset_no = atoi(p); /* current number of nodes in the preset */
    for(i = 0; i < t_temp->cur_preset_no; i++) {
      fgets(buffer,80,fp);
```

```
      p =strtok(buffer," \n");
      strcpy(t_temp->preset[i],p); /* reads preset */
    }
    fgets(buffer,80,fp);
    p =strtok(buffer," \n");
    if(strcmp(p,"POSTSET_LIST") != 0) { /* error in reading */
      fclose(fp);
      system("rm PETRINETTEMPFILE");
      InitEnvironment(data);
      return(0);
    }
    fgets(buffer,80,fp);
    p =strtok(buffer," \n");
    t_temp->cur_postset_no = atoi(p); /* current number of nodes in the postset */
    for(i = 0; i < t_temp->cur_postset_no; i++) {
      fgets(buffer,80,fp);
      p =strtok(buffer," \n");
      strcpy(t_temp->postset[i],p); /* reads postset */
    }
    fgets(buffer,80,fp);
    p =strtok(buffer," \n");
    t_temp->enabled = atoi(p); /* flag for enabled transition */
    fgets(buffer,INFO_LENGTH,fp);
    p =strtok(buffer,"\n");

    /* reads the interpretation */
    while((strcmp(p,"END_INTERPRETATION") != 0) && (!feof(fp))) {
      strcat(t_temp->interpretation, p);
      strcat(t_temp->interpretation, "\n");
      fgets(buffer,INFO_LENGTH,fp);
      p =strtok(buffer,"\n");
    }
    AllTrim(t_temp->interpretation); /* trims the interpretation */
    fgets(buffer,80,fp);
    p =strtok(buffer," \n");
  }
  if(strcmp(p,"GLOBAL_PTL_COUNTS") != 0) { /* error in reading */
    fclose(fp);
    system("rm PETRINETTEMPFILE");
    InitEnvironment(data);
    return(0);
  }
  fgets(buffer,80,fp);
  p =strtok(p," \n");
  PLACE_CNT = atoi(p);       /* total number of places */
  fgets(buffer,80,fp);
  p =strtok(p," \n");
  TRANSITION_CNT = atoi(p); /* total number of transitions */
  fgets(buffer,80,fp);
  p =strtok(p," \n");
  LINK_CNT = atoi(p);        /* current link count */
  fgets(buffer,80,fp);
  p =strtok(p," \n");
  if(strcmp(p,"END_OF_FILE") != 0) { /* error in reading */
    fclose(fp);
    system("rm PETRINETTEMPFILE");
    InitEnvironment(data);
    return(0);
  }
  fclose(fp);  /* closes the file after reading */
  system("rm PETRINETTEMPFILE");     /* removes the temporary file */
  strcpy(GLOBAL_FILE_NAME,filename); /* updates the filename */
}


/*
AssignFunction() assigns a drawing function depending upon the
object to be drawn. Function is assigned while reading an input
file.
*/

void AssignFunction(data, i, str)
graphics_data  *data;
char   *str;
{
  if(strcmp(str,"PLACE") == 0)
    data->buffer[i].func = DrawPlace;
  else if(strcmp(str,"TRANSITION") == 0)
    data->buffer[i].func = DrawTransition;
  else if(strcmp(str,"LINK") == 0)
    data->buffer[i].func = DrawArrowHeadLink;
```

```
  else if(strcmp(str,"PUT_TOKEN") == 0)
    data->buffer[i].func = DrawToken;
  else if(strcmp(str,"START_LINK") == 0)
    data->buffer[i].func = DrawLink;
}


/*
FileOpeningWarn() pops up a warning message about saving the current
file before trying to open a new file.
*/

void FileOpeningWarn(w,reason,file_selector)
Widget    w;
char      *reason;
Widget    file_selector;
{
  static Widget   warning = NULL;
  Arg       wargs[MAX_ARGS];
  int       n;
  XmString str1, str2, str3;

  /* creates a dialog box */
  if(!warning) {
  n = 0;
  str1 = XmStringCreateLtoR(" WARNING ",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNdialogTitle,str1); n++;
  str2 = XmStringCreateLtoR(reason,XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNmessageString,str2); n++;
  str3 = XmStringCreateLtoR("CLICK HERE",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNokLabelString,str3); n++;
  XtSetArg(wargs[n],XmNmessageAlignment,XmALIGNMENT_BEGINNING);
  XtSetArg(wargs[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL); n++;
  XtSetArg(wargs[n],XmNnoResize,True); n++;
  XtSetArg(wargs[n],XmNdefaultPosition,False); n++;
  XtSetArg(wargs[n],XmNx,350); n++;
  XtSetArg(wargs[n],XmNy,300); n++;
  warning = XmCreateErrorDialog(drawing, "warning",wargs,n);
  XmStringFree(str1);
  XmStringFree(str2);
  XtAddCallback(warning, XmNokCallback, DoneFileOpeningWarn, file_selector);
  XtUnmanageChild(XmMessageBoxGetChild(warning, XmDIALOG_CANCEL_BUTTON));
  XtUnmanageChild(XmMessageBoxGetChild(warning, XmDIALOG_HELP_BUTTON));
  XtManageChild(warning);
  }
  else { /* dialog box already exists */
    n = 0;
    str2 = XmStringCreateLtoR(reason,XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNmessageString,str2); n++;
    XtSetValues(warning,wargs,n);
    XtManageChild(warning);
  }
}


/*
DoneFileOpeningWarn(), as a callback function pops up the file
selection dialog box.
*/

void DoneFileOpeningWarn(w, file_selector, call_data)
Widget  w;
Widget  file_selector;
XtPointer   call_data;
{
  strcpy(action_selected,"OPEN FILE"); /* updates the action selected */
  RewriteAction();
  ResetCursor(drawing); /* resets the cursor shape */
  XtManageChild(file_selector);
  XtUnmanageChild(w);
}



/*
File name:   close_file.h
Description: It contains the header files required for the file close_file.c.
             It also declares the external functions and the functions
             defined in the file close_file.c
*/
```

```
#include<stdio.h>
#include<string.h>
#include<X11/StringDefs.h>
#include<Xm/MessageB.h>
#include"data_str.h"
#include"global.h"

/*
These externally defined functions are called in the functions
defined in the file close_file.c.
*/
extern void RewriteAction();      /* defined in utility.c */
extern void ResetCurrentAction(); /* defined in utility.c */
extern void PreSaveFile();        /* defined in save.c */

/*
The file close_file.c consists of following functions.
For details, refer explanation before the declaration of each
function.
*/

void CloseFile();
void CancelCloseFile();
void NoSaveCloseFile();
void SaveCloseFile();


/*
File name:   close_file.c
Description: It contains the functions used to implement the option "Quit".
*/

#include "close_file.h"


/*
CloseFile() is a callback function. It is called when the option "Quit"
is selected from the "File" pull-down menu. It pops up a dialog
box titled SHUT_DOWN, where user is prompted to save the current
file. Here user can select YES, NO or CANCEL.
*/

void CloseFile(w, data, call_data)
Widget   w;
graphics_data  *data;
XtPointer call_data;
{
  static Widget shut_down = NULL;  /* widget for the dialog box SHUT DOWN */
  Arg    wargs[10];
  int    n;
  XmString xmstr1, xmstr2, xmstr3, xmstr4, xmstr5;  /* XmStrings used in the
                                                       dialog box SHUT DOWN */
  if(data->next_pos == 0)  /* if no object is drawn, exit without saving */
    exit(0);
  ResetCurrentAction(data);  /* resets any current drawing action */
  strcpy(action_selected,"QUIT");  /* sets current action to QUIT */
  RewriteAction();  /* displays the selected action */
  if(!shut_down) {  /* creates if the shut_down widget does not exist */
    n = 0;
    /* sets resorces for shut_down widget */
    xmstr1 = XmStringCreate(" SHUT DOWN ",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNdialogTitle,xmstr1); n++;
    xmstr2 = XmStringCreate("Save Current file",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNmessageString,xmstr2); n++;
    xmstr3 = XmStringCreate("YES",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNokLabelString,xmstr3); n++;
    xmstr4 = XmStringCreate("NO",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNcancelLabelString,xmstr4); n++;
    xmstr5 = XmStringCreate("CANCEL",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNhelpLabelString,xmstr5); n++;
    XtSetArg(wargs[n],XmNmessageAlignment,XmALIGNMENT_BEGINNING);
    XtSetArg(wargs[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL); n++;
    XtSetArg(wargs[n],XmNnoResize,True); n++;
    XtSetArg(wargs[n],XmNautoUnmanage,False); n++;
    XtSetArg(wargs[n],XmNdefaultButtonType,XmDIALOG_HELP_BUTTON); n++;
    shut_down = XmCreateErrorDialog(w, "shut_down",wargs,n);
    XmStringFree(xmstr1);  /* frees memory */
    XmStringFree(xmstr2);
    XmStringFree(xmstr3);
    XmStringFree(xmstr4);
    XmStringFree(xmstr5);
```

```
    /* registers callback functions */
    XtAddCallback(shut_down, XmNokCallback, SaveCloseFile, data);
    XtAddCallback(shut_down, XmNcancelCallback, NoSaveCloseFile, data);
    XtAddCallback(shut_down, XmNhelpCallback, CancelCloseFile, NULL);
    XtManageChild(shut_down);  /* manages the shut_down widget */
  }
  else { /* if the shut_down widget exists, simply manage it */
    XtManageChild(shut_down);
  }
}


/*
CancelCloseFile() is a callback function of shut_down widget's
CANCEL button. It cancels the operation of Quit and gets back
to the application.
*/

void CancelCloseFile(w, client_data, call_data)
Widget  w;
XtPointer   client_data;
XtPointer   call_data;
{
  strcpy(action_selected,"NOTHING"); /* current action is NOTHING */
  RewriteAction();  /* displays the selected action */
  XtUnmanageChild(w);  /* unmanages the widget shut_down */
}



/*
NoSaveCloseFile() is a callback function of shut_down widget's
NO button. It shuts down the application without saving the file.
*/

void NoSaveCloseFile(w, data, call_data)
Widget  w;
graphics_data  *data;
XtPointer   call_data;
{
  exit(0);
}



/*
SaveCloseFile() is a callback function of shut_down widget's
SAVE button. If filename is already given, it saves and shuts
down the application. Otherwise it calls the function PreSaveFile() to
display the dialog box for save option.
*/

void SaveCloseFile(parent, data, call_data)
Widget         parent;
graphics_data  *data;
XtPointer      call_data;
{
  if(strcmp(GLOBAL_FILE_NAME,"Untitled") != 0) {
    SaveFile(data);  /* saves the file */
    exit(0);  /* shuts down the application */
  }
  PreSaveFile(parent, data);  /* call for save dialog */
}



/*
File name:   draw_object.h
Description: It contains the header files required for the file draw_object.c.
             It also declares the functions defined in the file draw_object.c
*/

#include <math.h>
#include "data_str.h"
#include "global.h"

/*
The file draw_object.c consists of following functions.
For details, refer explanation before the declaration of each
function.
```

```
*/
void CheckPoints();
void DrawPlace();
void DrawToken();
void DrawLink();
void DrawArrowHeadLink();
void DrawTransition();


/*
File name:   draw_object.c
Description: It contains the functions used to draw differnt objects, e.g.,
             circle, rectangle, line, etc.
*/

#include "draw_object.h"

/*
Some X servers do not draw polygonal figures correctly if the numeric
value of the second point is less than the first in either direction.
The function CheckPoints() checks for this case and reverses the
coordinates if necessary.
*/

void CheckPoints(x1, y1, x2, y2)
int  *x1, *y1, *x2, *y2;
{
  int temp;

  if(*x2 < *x1) {
    temp = *x1;
    *x1  = *x2;
    *x2  = temp;
  }
  if(*y2 < *y1) {
    temp = *y1;
    *y1  = *y2;
    *y2  = temp;
  }
}


/*
DrawPlace() uses Xlib graphics functions XDrawArc() and
XDrawString() to draw a circle and a string representing
a place and its label.
*/

void DrawPlace(w, gc, x1, y1, x2, y2, data)
Widget  w;
GC      gc;
int     x1, y1, x2, y2;
graphics_data  *data;
{
  Display  *dpy = XtDisplay(w);
  Window   win = XtWindow(w);
  char     p_label[LABEL_LENGTH];

  strcpy(p_label,data->label);  /* gets the label of a place */

  /* draws a circle */
  XDrawArc(dpy, win, gc, x1, y1, 40, 40, 0, 64 * 360);

  /* draws a string */
  XDrawString(dpy, win, gc, x1+43, y1+22, p_label, strlen(p_label));
}


/*
DrawToken() uses Xlib graphics function XDrawString() to display the
number of tokens in a place.
*/

void DrawToken(x1, y1,data)
int            x1, y1;
graphics_data *data;
{
  PLACE    *p_temp;
  char     label[LABEL_LENGTH];
  Display  *dpy = XtDisplay(drawing);
  Window    win = XtWindow(drawing);
```

```
  /* searches for the desired place in the P_LIST, i.e., list of the places */
  p_temp = P_LIST->HEAD;
  while((strcmp(p_temp->label,data->label) != 0) && (p_temp != NULL))
    p_temp = p_temp->next;

  if(p_temp->no_of_tokens > 99)  /* if tokens > 99, displays w */
    sprintf(label,"<w>");
  else
    sprintf(label,"<%d>",p_temp->no_of_tokens);

  /* displays token as string */
  XDrawString(dpy, win, data->gc, x1, y1, label, strlen(label));
}


/*
DrawArrowHeadLink() draws the final part of a link with an
arrow head, i.e., the part that hits the destination.
*/

void DrawArrowHeadLink(w, gc, x1, y1, x2, y2)
Widget  w;
GC      gc;
int     x1, y1, x2, y2;
{
  XSegment arrow_segment[3];
  double   r = .01745329;
  double   l = 15;
  double   temp;
  double   theta, alpha = 20.0*r;
  Display  *dpy = XtDisplay(w);
  Window   win = XtWindow(w);

  temp = sqrt(pow((double)abs(y2-y1),2.0)+pow((double)abs(x2-x1),2.0));
  arrow_segment[0].x1 = (short)x1;
  arrow_segment[0].y1 = (short)y1;
  arrow_segment[0].x2 = (short)x2;
  arrow_segment[0].y2 = (short)y2;
  theta = atan2((double)( y2 - y1), (double)(x2 - x1));
  arrow_segment[1].x1 = (short)x2;
  arrow_segment[1].y1 = (short)y2;
  arrow_segment[1].x2 = (short)x2 - (short)(l*cos(theta - alpha));
  arrow_segment[1].y2 = (short)y2 - (short)(l*sin(theta - alpha));
  arrow_segment[2].x1 = (short)x2;
  arrow_segment[2].y1 = (short)y2;
  arrow_segment[2].x2 = (short)x2 - (short)(l*cos(theta + alpha));
  arrow_segment[2].y2 = (short)y2 - (short)(l*sin(theta + alpha));

  XDrawSegments(dpy, win, gc, arrow_segment, 3);
}


/*
DrawLink() draws the intermediate part of a link i.e.,
a part without arrow head (not hitting the destination). The
Xlib function XDrawLine() is used to draw a line.
*/

void DrawLink(w, gc, x1, y1, x2, y2)
Widget  w;
GC      gc;
int     x1, y1, x2, y2;
{
  Display  *dpy = XtDisplay(w);
  Window    win = XtWindow(w);

  XDrawLine(dpy, win, gc, x1, y1, x2, y2);
}


/*
DrawTransition() draws a rectangle representing a transition.
If a transition is enabled, it is represented by a filled
rectangle. It also draws the label of the transition. The Xlib functions,
XDrawRectangle(), XFillRectangle(), and XDrawString() are used.
*/

void DrawTransition(w, gc, x1, y1, x2, y2, data, flag)
Widget  w;
GC      gc;
int     x1, y1, x2, y2;
```

```
graphics_data  *data;
char    *flag;
{
  TRANSITION  *t_temp;
  Display     *dpy = XtDisplay(w);
  Window      win = XtWindow(w);
  char        t_label[LABEL_LENGTH];

  CheckPoints(&x1, &y1, &x2, &y2);
  /* searches the required transition in the T_LIST */
  t_temp = T_LIST->HEAD;
  while((strcmp(t_temp->label,data->label) != 0) && (t_temp != NULL))
    t_temp = t_temp->next;

  if(t_temp == NULL) /* if a new tarnsition is being drawn */
    XDrawRectangle(dpy, win, gc, x1, y1, x2 - x1, y2 - y1);
  else {
    if(t_temp->enabled == YES) /* if the transition is enabled */
      XFillRectangle(dpy, win, gc, x1, y1, x2 - x1, y2 - y1);
    else
      XDrawRectangle(dpy, win, gc, x1, y1, x2 - x1, y2 - y1);
  }
  /* if is required to put a label */
  if(strcmp(flag,"PUT_LABEL") == 0) {
    strcpy(t_label,data->label);
    XDrawString(dpy, win, gc, x2+2, y1+5, t_label, strlen(t_label));
  }
}


/*
File name:   link.h
Description: It contains the header files required for the file link.c.
             It also declares the external functions and the functions
             defined in the file link.c
*/


#include <string.h>
#include <math.h>
#include <Xm/Text.h>
#include <Xm/BulletinB.h>
#include <Xm/PushB.h>
#include <Xm/MessageB.h>
#include "data_str.h"
#include "global.h"

/*
These externally defined functions are called in the functions
defined in the file link.c.
*/
extern void DrawLink();           /* defined in draw_object.c */
extern void RewriteAction();      /* defined in utility.c */
extern void ResetCurrentAction(); /* defined in utility.c */
extern void TrimString();         /* defined in utility.c */
extern void PutErrorDialog();     /* defined in utility.c */
extern char *GetLinkMsgStr();     /* defined in warn_msg.c */
extern void ResetCursor();        /* defined in cursor.c */
extern void SetBusyCursor();      /* defined in cursor.c */

/*
The file link.c consists of following functions.
For details, refer explanation before the declaration of each
function.
*/
void GetLinkInfo();
void CancelLinkInfo();
void DoneLinkInfo();
int  IsWithinPlace();
int  IsWithinTrans();
int  BellIfLinkSourceOk();
int  BellIfLinkDestOk();
void SaveLinkInfo();


/*
File name:   link.c
Description: It contains the functions required to implement the option
             LINK.
*/
```

```
#include "link.h"

/*
GetLinkInfo() pops up a dialog box which prompts the user for a source
and a destination of a link to be drawn. It also provides two
buttons, DONE and CANCEL. If the definition of the link is valid
and the DONE button is clicked, the dialog box pops down and then user
draws the link as defined.
*/
void GetLinkInfo(parent, data)
Widget   parent;
graphics_data  *data;
{
  Widget  board, label1,label2, heading1, heading2, done_button, help_button,
          cancel_button;
  Arg     wargs[MAX_ARGS];
  int     i, n;
  XmString  str;
  EDIT_LABEL_STRUCT *info_to_send;

  SetBusyCursor(toplevel); /* sets cursor to watch symbol */
  /* defines the LINK DEFINITION dialog box */
  str = XmStringCreate("LINK DEFINITION",XmSTRING_DEFAULT_CHARSET);
  n = 0;
  XtSetArg(wargs[n],XmNdialogTitle,str); n++;
  XtSetArg(wargs[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL); n++;
  XtSetArg(wargs[n],XmNautoUnmanage,False); n++;
  XtSetArg(wargs[n],XmNnoResize,True); n++;
  board = XmCreateBulletinBoardDialog(parent, "board", wargs, n);
  XmStringFree(str);
  /* creates a text widget */
  n = 0;
  XtSetArg(wargs[n],XmNx,130); n++;
  XtSetArg(wargs[n],XmNy,10); n++;
  label1 = XtCreateManagedWidget("label1", xmTextWidgetClass, board, wargs, n);
  /* creates a text widget */
  n = 0;
  XtSetArg(wargs[n],XmNx,130); n++;
  XtSetArg(wargs[n],XmNy,50); n++;
  label2 = XtCreateManagedWidget("label2", xmTextWidgetClass, board, wargs, n);
  /* creates a label widget */
  n = 0;
  str = XmStringCreate("SOURCE LABEL",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNx,10); n++;
  XtSetArg(wargs[n],XmNy,15); n++;
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNtraversalOn,False); n++;
  heading1 = XtCreateManagedWidget("heading1", xmLabelWidgetClass,
                          board, wargs, n);
  XmStringFree(str);
  /* creates a label widget */
  n = 0;
  str = XmStringCreate("DESTINATION LABEL",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNx,10); n++;
  XtSetArg(wargs[n],XmNy,55); n++;
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNtraversalOn,False); n++;
  heading2 = XtCreateManagedWidget("heading2", xmLabelWidgetClass,
                          board, wargs, n);
  XmStringFree(str);
  /* creates a callback data */
  info_to_send = (EDIT_LABEL_STRUCT *)malloc(sizeof(EDIT_LABEL_STRUCT));
  if(info_to_send == NULL) {
    printf("Out of memory\n");
    exit(0);
  }
  info_to_send->label_widget1 = label1; /* stores widget in callback data */
  info_to_send->label_widget2 = label2;
  /* creates a button widget */
  n = 0;
  str = XmStringCreate("DONE",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNx,10); n++;
  XtSetArg(wargs[n],XmNy,90); n++;
  XtSetArg(wargs[n],XmNuserData,info_to_send); n++;
  XtSetArg(wargs[n],XmNshadowThickness,4); n++;
  done_button = XtCreateManagedWidget("done", xmPushButtonWidgetClass,
                                  board, wargs, n);

  XmStringFree(str);
  XtAddCallback(done_button, XmNactivateCallback, DoneLinkInfo,
                data);
```

```
  /* creates a button widget */
  n = 0;
  str = XmStringCreate("CANCEL",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNx,50); n++;
  XtSetArg(wargs[n],XmNy,90); n++;
  XtSetArg(wargs[n],XmNshadowThickness,4); n++;
  cancel_button = XtCreateManagedWidget("cancel", xmPushButtonWidgetClass,
                                        board, wargs, n);
  XmStringFree(str);
  /* adds a callback function */
  XtAddCallback(cancel_button, XmNactivateCallback, CancelLinkInfo,data);
  XtManageChild(board);
  XmProcessTraversal(label1,XmTRAVERSE_CURRENT); /* focus to first entry */
  ResetCursor(toplevel);
}


/*
CancelLinkInfo() is a callback function registered with the CANCEL
button of the dialog box titled LINK DEFINITION. It cancels the
action and pops down the dialog box.
*/
void CancelLinkInfo(w, data, call_data)
Widget                w;
graphics_data         *data;
XmAnyCallbackStruct   *call_data;
{
  XtUnmanageChild(XtParent(w)); /* pops down the dialog box */
  ResetCurrentAction(); /* resets the action selected */
  strcpy(action_selected,"NOTHING"); /* updates the action selected */
  RewriteAction();
}


/*
DoneLinkInfo() is a callback function registered with the DONE
button of the dialog box titled LINK DEFINITION. It checks the
validity of the defined link. If the definition is valid, it
pops down the dialog box and then user draws the link.
*/

void DoneLinkInfo(w, data, call_data)
Widget                w;
graphics_data         *data;
XmAnyCallbackStruct   *call_data;
{
  EDIT_LABEL_STRUCT  *info;
  PLACE       *src_p_temp, *dest_p_temp; /* used to traverse the lists */
  TRANSITION  *src_t_temp, *dest_t_temp;
  Arg wargs[2];
  int      i;
  Display  *dpy = XtDisplay(drawing);
  Window    win = XtWindow(drawing);

  char  *src_label, /* label of the source node */
        *dst_label; /* label of the destination node */
  char  source_type[20], /* type of the source node, i.e., place or transition */
        dest_type[20]; /* type of the destination node */
  char  mesg[MSG_LENGTH];

  XtSetArg(wargs[0], XmNuserData, &info);
  XtGetValues(w, wargs, 1);
  src_label = XmTextGetString(info->label_widget1); /* gets the labels */
  dst_label = XmTextGetString(info->label_widget2);

  TrimString(src_label);
  TrimString(dst_label);
  if(strlen(src_label) == 0) { /* empty string for the source */
    strcpy(mesg,GetLinkMsgStr("NULL_SRC"));
    PutErrorDialog(drawing,mesg);
    return;
  }
  else if(strlen(dst_label) == 0) { /* empty string for the destination */
    strcpy(mesg,GetLinkMsgStr("NULL_DST"));
    PutErrorDialog(drawing,mesg);
    return;
  }
  src_p_temp = P_LIST->HEAD;
  src_t_temp = T_LIST->HEAD;
  while((strcmp(src_p_temp->label,src_label) != 0) && (src_p_temp != NULL))
```

```
    src_p_temp = src_p_temp->next;

  if(src_p_temp == NULL) {
    while((strcmp(src_t_temp->label,src_label) != 0) && (src_t_temp != NULL))
      src_t_temp = src_t_temp->next;
  }
  else
    strcpy(source_type,"PLACE"); /* the source is a place */

  /* displays the message when the source is not found  */
  if((src_t_temp == NULL) && (src_p_temp == NULL)){
    strcpy(mesg,GetLinkMsgStr("NO_SRC"));
    PutErrorDialog(drawing,mesg);
    return;
  }
  else if(src_p_temp == NULL)
    strcpy(source_type,"TRANS"); /* the source is a transition */

  dest_p_temp = P_LIST->HEAD;
  dest_t_temp = T_LIST->HEAD;
  while((strcmp(dest_p_temp->label,dst_label) != 0) && (dest_p_temp != NULL))
    dest_p_temp = dest_p_temp->next;

  if(dest_p_temp == NULL) {
    while((strcmp(dest_t_temp->label,dst_label) != 0) && (dest_t_temp != NULL))
      dest_t_temp = dest_t_temp->next;
  }
  else
    strcpy(dest_type,"PLACE"); /* the destination is a place */

  /* displays the message when the destination is not found  */
  if((dest_t_temp == NULL) && (dest_p_temp == NULL)) {
    strcpy(mesg,GetLinkMsgStr("NO_DST"));
    PutErrorDialog(drawing,mesg);
    return;
  }
  else if(dest_p_temp == NULL)
    strcpy(dest_type,"TRANS"); /* destination is a transition */

  /* displays the message when the source and the destination are same type node */
  if((strcmp(source_type,dest_type)) == 0) {
    strcpy(mesg,GetLinkMsgStr("SAME_TYPE_NODE"));
    PutErrorDialog(drawing,mesg);
    return;
  }

  /* makes the postset of the source and the preset of the destination */
  if((strcmp(source_type,"PLACE") == 0) && (strcmp(dest_type,"TRANS") == 0)) {
    strcpy(P_LIST->place_pre_selected,src_p_temp->label);
    P_LIST->pre_post = POSTSET;
    P_LIST->pre_post_index = src_p_temp->cur_postset_no;
    strcpy(T_LIST->trans_pre_selected,dest_t_temp->label);
    T_LIST->pre_post = PRESET;
    T_LIST->pre_post_index = dest_t_temp->cur_preset_no;
  }
  else {
    strcpy(P_LIST->place_pre_selected,dest_p_temp->label);
    P_LIST->pre_post = PRESET;
    P_LIST->pre_post_index = dest_p_temp->cur_preset_no;
    strcpy(T_LIST->trans_pre_selected,src_t_temp->label);
    T_LIST->pre_post = POSTSET;
    T_LIST->pre_post_index = src_t_temp->cur_postset_no;
  }
  data->current_func = DrawLink; /* sets the current function */
  XtUnmanageChild(XtParent(w)); /* pops down the dialog box */
}


/*
IsWithinPlace() checks if the coordinate (h, k) lies within the allowed
boundary of a place with center (x, y). It calculates the radius
with these two points and checks against the allowed radius, i.e.,
18.0 to 21.0.
*/

int IsWithinPlace(x, y, h, k)
int  x, y, h, k;
{
  float  radius;

  radius = sqrt(pow((double)abs(x-h),2.0)+pow((double)abs(y-k),2.0));
```

```
  if((radius <= 21.0) && (radius >= 18.0))
    return(1);
  else
    return(0);
}


/*
IsWithinTrans() checks if the coordinate (h, k) lies within the allowed
boundary of a transition whose top left and bottom right points are
(x1, y1) and (x2, y2).
*/

int IsWithinTrans(h, k, x1, y1, x2, y2)
int  x1, y1, x2, y2, h, k;
{
  if(((h >= x1) && (h <= x2) && (k >= y1) && (k <= y2))||
     ((h <= x1) && (h >= x2) && (k >= y1) && (k <= y2))||
     ((h >= x1) && (h <= x2) && (k <= y1) && (k >= y2))||
     ((h <= x1) && (h >= x2) && (k <= y1) && (k >= y2))) {
    return(1);
  }
  else
    return(0);
}


/*
BellIfLinkSourceOk() is called when user clicks on a source to
start drawing a link. It checks if the selected source is
valid (according to the link definition). If the selected source is
valid, a bell sounds, otherwise a warning message is displayed
with the valid link definition.
*/

int BellIfLinkSourceOk(data)
graphics_data   *data;
{
  int  i,
       place_ndx = -1,
       trans_ndx = -1;
  char mesg[MSG_LENGTH];
  Display  *dpy = XtDisplay(drawing);

  for(i = 0; i < data->next_pos; i++) {
    if(strcmp(P_LIST->place_pre_selected,data->buffer[i].label) == 0) {
      place_ndx = i;
      break;
    }
  }
  for(i = 0; i < data->next_pos; i++) {
    if(strcmp(T_LIST->trans_pre_selected,data->buffer[i].label) == 0) {
      trans_ndx = i;
      break;
    }
  }
  if(P_LIST->pre_post == POSTSET) {
    /* if the source is valid */
    if(IsWithinPlace(data->x1, data->y1, data->buffer[place_ndx].x1+20,
                     data->buffer[place_ndx].y1+20)) {
      XBell(dpy,2);
      strcpy(data->object,"START_LINK");
      return(1);
    }
    /* else a warning message is displayed */
    else {
      strcpy(mesg,GetLinkMsgStr("PSRC_ERR_MSG"));
      PutErrorDialog(drawing,mesg);
    }
  }
  else if(T_LIST->pre_post == POSTSET) {
    /* if the source is valid */
    if(IsWithinTrans(data->x1, data->y1,
                     data->buffer[trans_ndx].x1,
                     data->buffer[trans_ndx].y1,
                     data->buffer[trans_ndx].x2,
                     data->buffer[trans_ndx].y2)) {
      XBell(dpy,2);
      strcpy(data->object,"START_LINK");
      return(1);
    }
```

```
    /* else a warning message is displayed */
    else {
      strcpy(mesg,GetLinkMsgStr("TSRC_ERR_MSG"));
      PutErrorDialog(drawing,mesg);
    }
  }
  return(0); /* if the source is not valid */
}


/*
BellIfLinkDestOk() checks if the user hits the valid destination
during the process of making a link. When the valid destination is
hit, a bell sounds and the link is established between the source
and the destination.
*/

int BellIfLinkDestOk(data)
graphics_data   *data;
{
  int  i,
       place_ndx = -1,
       trans_ndx = -1;
  Display  *dpy = XtDisplay(drawing);

  for(i = 0; i < data->next_pos; i++) {
    if(strcmp(P_LIST->place_pre_selected,data->buffer[i].label) == 0) {
      place_ndx = i;
      break;
    }
  }
  for(i = 0; i < data->next_pos; i++) {
    if(strcmp(T_LIST->trans_pre_selected,data->buffer[i].label) == 0) {
      trans_ndx = i;
      break;
    }
  }
  if(P_LIST->pre_post == PRESET) {
    /* if the destination is valid, bell sounds and the link is completed */
    if(IsWithinPlace(data->x2, data->y2, data->buffer[place_ndx].x1+20,
                     data->buffer[place_ndx].y1+20)) {
      XBell(dpy,2); /* the link is completed */
      strcpy(action_selected,"NOTHING"); /* updates the action selected */
      RewriteAction();
      return(1);
    }
  }
  else if(T_LIST->pre_post == PRESET) {
    /* if the destination is valid, bell sounds and the link is completed */
    if(IsWithinTrans(data->x2, data->y2,
                     data->buffer[trans_ndx].x1,
                     data->buffer[trans_ndx].y1,
                     data->buffer[trans_ndx].x2,
                     data->buffer[trans_ndx].y2)) {
      XBell(dpy,2); /* the link is completed */
      strcpy(action_selected,"NOTHING"); /* updates the action selected */
      RewriteAction();
      return(1);
    }
  }
  return(0);
}


/*
SaveLinkInfo() saves the information about a completed link.
*/

void SaveLinkInfo()
{
  PLACE       *p_temp;
  TRANSITION  *t_temp;

  p_temp = P_LIST->HEAD;
  while((strcmp(p_temp->label,P_LIST->place_pre_selected) != 0) &&
        (p_temp != NULL))
    p_temp = p_temp->next;
  if(P_LIST->pre_post == PRESET)
    strcpy(p_temp->preset[p_temp->cur_preset_no++],T_LIST->trans_pre_selected);  else
if(P_LIST->pre_post == POSTSET)
    strcpy(p_temp->postset[p_temp->cur_postset_no++],
```

```
                                T_LIST->trans_pre_selected);
  t_temp = T_LIST->HEAD;
  while((strcmp(t_temp->label,T_LIST->trans_pre_selected) != 0) &&
        (t_temp != NULL))
    t_temp = t_temp->next;
  if(T_LIST->pre_post == PRESET)
    strcpy(t_temp->preset[t_temp->cur_preset_no++],P_LIST->place_pre_selected);  else
if(T_LIST->pre_post == POSTSET)
    strcpy(t_temp->postset[t_temp->cur_postset_no++],
           P_LIST->place_pre_selected);
  strcpy(P_LIST->place_selected,P_LIST->place_pre_selected);
  strcpy(T_LIST->trans_selected,T_LIST->trans_pre_selected);
}


/*
File name:   edit_label.h
Description: It contains the header files required for the file edit_label.c.
             It also declares the external functions and the functions
             defined in the file edit_label.c
*/

#include <string.h>
#include <Xm/Text.h>
#include <Xm/BulletinB.h>
#include <Xm/PushB.h>
#include <Xm/MessageB.h>
#include "data_str.h"
#include "global.h"

/*
These externally defined functions are called in the functions
defined in the file edit_label.c.
*/
extern void RewriteAction();        /* defined in utility.c */
extern void RedrawDrawingBoard();   /* defined in utility.c */
extern void ResetCurrentAction();   /* defined in utility.c */
extern void RemoveIncompleteLink(); /* defined in utility.c */
extern void TrimString();           /* defined in utility.c */
extern void PutErrorDialog();       /* defined in utility.c */
extern void ResetCursor();          /* defined in cursor.c */
extern void GetEditLabelMsgStr();   /* defined in warn_msg.c */


/*
The file edit_label.c consists of following functions. These functions are
used to implement the Edit Label option present in the pull down menu Edit.
For details, refer explanation before the declaration of each
function.
*/
void GetLabelInfo();
void CancelEditLabel();
void DoneEditLabel();
int  CheckAndEditPlaceList();
int  CheckAndEditTransList();
void GetLabelFromTransPreset();
void GetLabelFromTransPostset();
void GetLabelFromPlacePreset();
void GetLabelFromPlacePostset();
int  UpdatePlaceInPreset();
int  UpdateTransInPreset();
int  UpdatePlaceInPostset();
int  UpdateTransInPostset();
void UpdateLabelInLinks();


/*
File name:   edit_label.c
Description: It contains the functions required to implement the option
             Edit Label.
*/

#include "edit_label.h"


/*
GetLabelInfo(), as a callback procedure, pops up a dialog box which
asks a user to give an old label and a new label. It provides two buttons,
DONE and CANCEL for a user to respond. If the old label and the new label are
valid, and DONE button is clicked, then the old label is changed by the new
label.
```

```
*/
void GetLabelInfo(parent, data ,call_data)
Widget        parent;
graphics_data *data;
XtPointer     call_data;
{
  static Widget  board = NULL, label1,label2, heading1, heading2,
                 done_button, help_button,
          cancel_button;
  Arg     wargs[MAX_ARGS];
  int     i, n;
  XmString  str;
  static EDIT_LABEL_STRUCT  callback_data;

  ResetCurrentAction(data); /* resets current drawing action */
  strcpy(action_selected,"EDIT LABEL"); /* updates the action selected */
  RewriteAction(); /* displays the action selected */

  /* creates the user interface of the dialog box */
  if(!board) {
  str = XmStringCreate("EDIT_LABEL",XmSTRING_DEFAULT_CHARSET);
  n = 0;
  XtSetArg(wargs[n],XmNdialogTitle,str); n++;
  XtSetArg(wargs[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL); n++;
  XtSetArg(wargs[n],XmNautoUnmanage,False); n++;
  XtSetArg(wargs[n],XmNnoResize,True); n++;
  board = XmCreateBulletinBoardDialog(parent, "board", wargs, n);
  XmStringFree(str);
  /* creates a text widget */
  n = 0;
  XtSetArg(wargs[n],XmNx,100); n++;
  XtSetArg(wargs[n],XmNy,10); n++;
  XtSetArg(wargs[n],XmNmaxLength,300); n++;
  label1 = XtCreateManagedWidget("label1", xmTextWidgetClass, board, wargs, n);
  /* creates a text widget */
  n = 0;
  XtSetArg(wargs[n],XmNx,100); n++;
  XtSetArg(wargs[n],XmNy,50); n++;
  XtSetArg(wargs[n],XmNmaxLength,300); n++;
  label2 = XtCreateManagedWidget("label2", xmTextWidgetClass, board, wargs, n);
  /* creates a label widget */
  n = 0;
  XtSetArg(wargs[n],XmNx,10); n++;
  XtSetArg(wargs[n],XmNy,10); n++;
  str = XmStringCreate("OLD_LABEL",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNtraversalOn,False); n++;
  heading1 = XtCreateManagedWidget("heading1", xmLabelWidgetClass,
                          board, wargs, n);
  XmStringFree(str);
  /* creates a label widget */
  n = 0;
  XtSetArg(wargs[n],XmNx,10); n++;
  XtSetArg(wargs[n],XmNy,50); n++;
  str = XmStringCreate("NEW_LABEL",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNtraversalOn,False); n++;
  heading2 = XtCreateManagedWidget("heading2", xmLabelWidgetClass,
                          board, wargs, n);
  XmStringFree(str);
  /* creates a button widget */
  n = 0;
  str = XmStringCreate("DONE",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNx,10); n++;
  XtSetArg(wargs[n],XmNy,90); n++;
  XtSetArg(wargs[n],XmNshadowThickness,4); n++;
  done_button = XtCreateManagedWidget("done", xmPushButtonWidgetClass,
                                  board, wargs, n);
  XmStringFree(str);
  /* callback data */
  callback_data.label_widget1 = label1;
  callback_data.label_widget2 = label2;
  callback_data.temp_data = data;
  /* adds a callback function */
  XtAddCallback(done_button, XmNactivateCallback, DoneEditLabel,
            &callback_data);
  /* creates a button widget */
  n = 0;
  str = XmStringCreate("CANCEL",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNlabelString,str); n++;
```

```
  XtSetArg(wargs[n],XmNx,50); n++;
  XtSetArg(wargs[n],XmNy,90); n++;
  XtSetArg(wargs[n],XmNshadowThickness,4); n++;
  cancel_button = XtCreateManagedWidget("cancel", xmPushButtonWidgetClass,
                                        board, wargs, n);
  XmStringFree(str);
  /* adds a callback function */
  XtAddCallback(cancel_button, XmNactivateCallback, CancelEditLabel,NULL);
  XtManageChild(board);
  XmProcessTraversal(label1,XmTRAVERSE_CURRENT); /* focus to first entry */
  ResetCursor(toplevel);
  }
  else {
    n = 0;
    XtSetArg(wargs[n],XmNvalue,""); n++;
    XtSetValues(label1,wargs,n);
    n = 0;
    XtSetArg(wargs[n],XmNvalue,""); n++;
    XtSetValues(label2,wargs,n);
    XtManageChild(board);
    XmProcessTraversal(label1,XmTRAVERSE_CURRENT); /* focus to first entry */
    ResetCursor(toplevel);
  }
}


/*
CancelEditLabel() is a callback function for the dialog box titled
EDIT_LABEL, created above. This is for CANCEL button. It cancels the
action of editing a label and pops down the EDIT_LABEL dialog box.
*/

void CancelEditLabel(w, client_data, call_data)
Widget                  w;
XtPointer               client_data;
XmAnyCallbackStruct  *call_data;
{
  strcpy(action_selected,"NOTHING"); /* updates the action selected */
  RewriteAction();
  XtUnmanageChild(XtParent(w)); /* pops down the EDIT_LABEL dialog box */
}


/*
DoneEditLabel() is a callback function for the dialog box titled
EDIT_LABEL, created above. This is for DONE button. If the old label
and the new label are valid, then the old label is changed by the new label.
Every occurence of the label is updated. It calls different
functions to accomplish this task.
*/

void DoneEditLabel(w, info, call_data)
Widget                  w;
EDIT_LABEL_STRUCT    *info;
XmAnyCallbackStruct  *call_data;
{
  PLACE       *p_temp;
  TRANSITION  *t_temp;
  Arg wargs[MAX_ARGS];
  int      i;
  Display  *dpy = XtDisplay(drawing);
  Window    win = XtWindow(drawing);

  char  old_label[300],
        new_label[300],
        *buf1,
        *buf2,
        mesg[MSG_LENGTH];

  /* gets the labels from the corresponding widgets */
  buf1 = XmTextGetString(info->label_widget1);
  buf2 = XmTextGetString(info->label_widget2);
  strncpy(old_label,buf1,300);
  strncpy(new_label,buf2,300);
  XtFree(buf1);
  XtFree(buf2);
  /* trims the labels */
  TrimString(old_label); /* TrimString() is defined in utility.c */
  TrimString(new_label);
  if(strlen(old_label) == 0) { /* if the old_label is empty */
    GetEditLabelMsgStr(mesg, "NULL_OLABEL");
```

```
    PutErrorDialog(w,mesg); /* pops up warning dialog box */
    return;
  }
  else if(strlen(new_label) == 0) { /* if the new_label is empty */
    GetEditLabelMsgStr(mesg, "NULL_NLABEL");
    PutErrorDialog(w,mesg);
    return;
  }
  /* searches the old_label in the T_LIST and the P_LIST */
  p_temp = P_LIST->HEAD;
  while((strcmp(p_temp->label,old_label) != 0) && (p_temp != NULL))
    p_temp = p_temp->next;
  t_temp = T_LIST->HEAD;
  while((strcmp(t_temp->label,old_label) != 0) && (t_temp != NULL))
    t_temp = t_temp->next;

  /* the old_label is not found in both the lists */
  if((p_temp == NULL) && (t_temp == NULL)) {
    GetEditLabelMsgStr(mesg, "LB_NOT_FOUND");
    PutErrorDialog(w,mesg);
    return;
  }
  /* searches the new_label in the P_LIST */
  p_temp = P_LIST->HEAD;
  while((strcmp(p_temp->label,new_label) != 0) && (p_temp != NULL))
    p_temp = p_temp->next;
  if(p_temp != NULL) { /* the new_label already present in the P_LIST */
    GetEditLabelMsgStr(mesg, "PLABEL_EXISTS");
    PutErrorDialog(w,mesg);
    return;
  }
  /* searches the new_label in the T_LIST */
  t_temp = T_LIST->HEAD;
  while((strcmp(t_temp->label,new_label) != 0) && (t_temp != NULL))
    t_temp = t_temp->next;
  if(t_temp != NULL) { /* the new_label already present in the T_LIST */
    GetEditLabelMsgStr(mesg, "TLABEL_EXISTS");
    PutErrorDialog(w,mesg);
    return;
  }
  if(!(CheckAndEditPlaceList(old_label,new_label,info->temp_data))) {
    if(!(CheckAndEditTransList(old_label,new_label,info->temp_data))) {
      GetEditLabelMsgStr(mesg, "LB_NOT_FOUND");
      PutErrorDialog(w,mesg);
      return;
    }
  }
  RedrawDrawingBoard(info->temp_data); /* redraws the drawing area */
  strcpy(action_selected,"NOTHING");
  RewriteAction();
  XtUnmanageChild(XtParent(w)); /* pops down the dilaog */
}


/*
CheckAndEditPlaceList() searches the P_LIST and changes the label if
required.
*/

CheckAndEditPlaceList(old_label, new_label,data)
char           *old_label,
               *new_label;
graphics_data  *data;
{
  PLACE    *p_temp;
  int    i;

  p_temp = P_LIST->HEAD; /* searches the P_LIST */
  while((strcmp(p_temp->label,old_label) != 0) && (p_temp != NULL))
    p_temp = p_temp->next;
  if(p_temp != NULL) { /* if found */
    UpdateLabelInLinks(p_temp->label,new_label,data); /* updates the label */
    if(p_temp->cur_preset_no > 0)
      GetLabelFromPlacePreset(p_temp, new_label);

    if(p_temp->cur_postset_no > 0)
      GetLabelFromPlacePostset(p_temp, new_label);

    strcpy(p_temp->label,new_label); /* updates the label */
    i = 0;
    while((strcmp(data->buffer[i].label,old_label) != 0) &&
```

```
          (i < data->next_pos))
      i++;
    if(i == data->next_pos) { /* error in the application */
      printf("failed to find this label in buffer but present in PLACE_LIST\n");
      printf("ERROR\n");
      exit(-1);
    }
    strcpy(data->buffer[i].label,new_label); /* updates the label */
    strcpy(LAST1,old_label); /* stores the information for the undo action */
    strcpy(LAST2,new_label);
    UNDO_ACTION = UNDO_LABEL; /* updates the undo action */
    return(1);
  }
  return(0); /* if the label is not found in the P_LIST */
}


/*
CheckAndEditTranseList() searches the T_LIST and changes the label if
required.
*/

CheckAndEditTransList(old_label, new_label,data)
char                *old_label,
                    *new_label;
graphics_data       *data;
{
  TRANSITION    *t_temp;
  int    i;

  t_temp = T_LIST->HEAD; /* searches the T_LIST */
  while((strcmp(t_temp->label,old_label) != 0) && (t_temp != NULL))
    t_temp = t_temp->next;
  if(t_temp != NULL) { /* if found */
    UpdateLabelInLinks(t_temp->label,new_label,data); /* updates the label */
    if(t_temp->cur_preset_no > 0)
      GetLabelFromTransPreset(t_temp, new_label);
    if(t_temp->cur_postset_no > 0)
      GetLabelFromTransPostset(t_temp, new_label);
    strcpy(t_temp->label,new_label); /* updates the label */

    i = 0;
    while((strcmp(data->buffer[i].label,old_label) != 0) &&
          (i < data->next_pos))
      i++;
    if(i == data->next_pos) { /* error in the application */
      printf("failed to find this label in buffer but present in PLACE_LIST\n");
      printf("ERROR\n");
      exit(-1);
    }
    strcpy(data->buffer[i].label,new_label); /* updates the label */
    strcpy(LAST1,old_label); /* stores the information for the undo action */
    strcpy(LAST2,new_label);
    UNDO_ACTION = UNDO_LABEL; /* updates the undo action */
    return(1);
  }
  return(0); /* if the label is not found in the T_LIST */
}


/*
GetLabelFromTransPreset() picks every node from the preset of a
transition (whose label is to be changed) and updates its occurence
in the postset of these nodes.
*/

void GetLabelFromTransPreset(t_temp, new_label)
TRANSITION  *t_temp;
char   *new_label;
{
  char    label[LABEL_LENGTH];
  int     i;

  for(i = 0; i < t_temp->cur_preset_no; i++) {
    strcpy(label, t_temp->preset[i]);
    if(!UpdatePlaceInPostset(label,t_temp->label, new_label))
      UpdateTransInPostset(label,t_temp->label, new_label);
  }
}
```

```
/*
GetLabelFromTransPostset() picks every node from the postset of a
transition (whose label is to be changed) and updates its occurence
in the preset of these nodes.
*/

void GetLabelFromTransPostset(t_temp, new_label)
TRANSITION  *t_temp;
char   *new_label;
{
  char    label[LABEL_LENGTH];
  int     i;

  for(i = 0; i < t_temp->cur_postset_no; i++) {
    strcpy(label, t_temp->postset[i]);
    if(!UpdatePlaceInPreset(label, t_temp->label, new_label))
      UpdateTransInPreset(label, t_temp->label, new_label);
  }
}


/*
GetLabelFromPlacePreset() picks every node from the preset of a
place (whose label is to be changed) and updates its occurence
in the postset of these nodes.
*/

void GetLabelFromPlacePreset(p_temp, new_label)
PLACE  *p_temp;
char   *new_label;
{
  char    label[LABEL_LENGTH];
  int     i;

  for(i = 0; i < p_temp->cur_preset_no; i++) {
    strcpy(label, p_temp->preset[i]);
    if(!UpdatePlaceInPostset(label,p_temp->label, new_label))
      UpdateTransInPostset(label,p_temp->label, new_label);
  }
}


/*
GetLabelFromPlacePostset() picks every node from the postset of a
place (whose label is to be changed) and updates its occurence
in the preset of these nodes.
*/

void GetLabelFromPlacePostset(p_temp, new_label)
PLACE  *p_temp;
char   *new_label;
{
  char    label[LABEL_LENGTH];
  int     i;

  for(i = 0; i < p_temp->cur_postset_no; i++) {
    strcpy(label, p_temp->postset[i]);
    if(!UpdatePlaceInPreset(label, p_temp->label, new_label))
      UpdateTransInPreset(label,p_temp->label, new_label);
  }
}

/*
UpdatePlaceInPreset() scans the preset of a place and updates the
label if required.
*/

UpdatePlaceInPreset(label, old_label, new_label)
char   *label, /* a place whose preset is to be scaned */
       *old_label,
       *new_label;
{
  PLACE  *p_temp;
  int    flag, i;

  flag = 0;
  p_temp = P_LIST->HEAD;
  while((strcmp(p_temp->label,label) != 0) && (p_temp != NULL))
    p_temp = p_temp->next;
  if(p_temp != NULL) {
    for(i = 0; i < p_temp->cur_preset_no; i++) { /* scans the preset */
```

```
      if((strcmp(p_temp->preset[i],old_label) == 0)) { /* finds the label */
        strcpy(p_temp->preset[i],new_label); /* updates the label */
        flag = 1;
      }
    }
  }
  return(flag); /* returns 1 if the label is updated */
}


/*
UpdateTransInPreset() scans the preset of a transition and
updates the label if required.
*/

UpdateTransInPreset(label, old_label, new_label)
char   *label, /* a transition whose preset is to be scaned */
       *old_label,
       *new_label;
{
  TRANSITION  *t_temp;
  int         flag, i;

  flag = 0;
  t_temp = T_LIST->HEAD;
  while((strcmp(t_temp->label,label) != 0) && (t_temp != NULL))
    t_temp = t_temp->next;
  if(t_temp != NULL) {
    for(i = 0; i < t_temp->cur_preset_no; i++) { /* scans the preset */
      if((strcmp(t_temp->preset[i],old_label) == 0)) { /* finds the label */
        strcpy(t_temp->preset[i],new_label); /* updates the label */
        flag = 1;
      }
    }
  }
  return(flag); /* returns 1 if the label is updated */
}


/*
UpdatePlaceInPostset() scans the postset of a place and
updates the label if required.
*/

UpdatePlaceInPostset(label, old_label, new_label)
char   *label, /* a place whose preset is to be scaned */
       *old_label,
       *new_label;
{
  PLACE  *p_temp;
  int    flag, i;

  flag = 0;
  p_temp = P_LIST->HEAD;
  while((strcmp(p_temp->label,label) != 0) && (p_temp != NULL))
    p_temp = p_temp->next;

  if(p_temp != NULL) {
    for(i = 0; i < p_temp->cur_postset_no; i++) { /* scans the postset */
      if((strcmp(p_temp->postset[i],old_label) == 0)) { /* finds the label */
        strcpy(p_temp->postset[i],new_label); /* updates the label */
        flag = 1;
      }
    }
  }
  return(flag); /* returns 1 if the label is updated */
}


/*
UpdateTransInPostset() updates the postset of a transition and
updates the label if required.
*/

UpdateTransInPostset(label, old_label, new_label)
char   *label, /* a transition whose preset is to be scaned */
       *old_label,
       *new_label;
{
  TRANSITION  *t_temp;
  int         flag, i;
```

```
  flag = 0;
  t_temp = T_LIST->HEAD;
  while((strcmp(t_temp->label,label) != 0) && (t_temp != NULL))
    t_temp = t_temp->next;
  if(t_temp != NULL) {
    for(i = 0; i < t_temp->cur_postset_no; i++) { /* scans the postset */
      if((strcmp(t_temp->postset[i],old_label) == 0)) { /* finds the label */
        strcpy(t_temp->postset[i],new_label); /* updates the label */
        flag = 1;
      }
    }
  }
  return(flag); /* returns 1 if the label is updated */
}


/*
UpdateLabelInLinks() updates the label involved in all of the links.
Every link maintains its own internal label which consists of the
labels of the nodes involved. So whenever a label of a node is
changed it should be changed in the internal link's label. This
function does this task.
*/

void UpdateLabelInLinks(old_label,new_label,data)
char    *old_label;
char    *new_label;
graphics_data  *data;
{
  char  temp_buf[LABEL_LENGTH], /* a temporary storage which is used
                                   by strtok() */
        new_link[LABEL_LENGTH], /* stores the new link label */
        *p;
  int   i;

  /* searches the buffer and updates the link if required */
  for(i = 0; i < data->next_pos; i++) {
    if((strcmp(data->buffer[i].object,"LINK") == 0) ||
       (strcmp(data->buffer[i].object,"START_LINK") == 0)||
       (strcmp(data->buffer[i].object,"END_LINK") == 0)) {
      strcpy(temp_buf,data->buffer[i].label);
      p = strtok(temp_buf,"~~ \n");
      if(strcmp(p,"PLACE=>") == 0) {
        strcpy(new_link,"PLACE=>~~");
        p = strtok(NULL,"~~ \n");
        if(strcmp(p,old_label) == 0) {
          strcat(new_link,new_label);
          strcat(new_link,"~");
          p = strtok(NULL,"\0");
          strcat(new_link,p);
          strcpy(data->buffer[i].label,new_link); /* updates the link label */
        }
        else {
          strcat(new_link,p);
          strcat(new_link,"~~");
        }
      }
      else if(strcmp(p,"TRANSITION=>") == 0) {
        strcpy(new_link,"TRANSITION=>~~");
        p = strtok(NULL,"~~ \n");
        if(strcmp(p,old_label) == 0) {
          strcat(new_link,new_label);
          strcat(new_link,"~");
          p = strtok(NULL,"\0");
          strcat(new_link,p);
          strcpy(data->buffer[i].label,new_link); /* updates the link label */
        }
        else {
          strcat(new_link,p);
          strcat(new_link,"~~");
        }
      }
      p = strtok(NULL,"~~ \n");
      if(strcmp(p,"PLACE=>") == 0) {
        strcat(new_link,"PLACE=>~~");
        p = strtok(NULL,"~~ \n");
        if(strcmp(p,old_label) == 0) {
          strcat(new_link,new_label);
          strcat(new_link,"~");
          p = strtok(NULL,"\0");
          strcat(new_link,p);
```

```
          strcpy(data->buffer[i].label,new_link); /* updates the link label */
        }
      }
      else if(strcmp(p,"TRANSITION=>") == 0) {
        strcat(new_link,"TRANSITION=>~~");
        p = strtok(NULL,"~~ \n");
        if(strcmp(p,old_label) == 0) {
          strcat(new_link,new_label);
          strcat(new_link,"~");
          p = strtok(NULL,"\0");
          strcat(new_link,p);
          strcpy(data->buffer[i].label,new_link); /* updates the link label */
        }
      }
    }
  }
}


/*
File name:   delete.h
Description: It contains the header files required for the file delete.c.
             It also declares the external functions and the functions
             defined in the file delete.c
*/

#include "data_str.h"
#include "global.h"

/*
These externally defined functions are called in the functions
defined in the file delete.c.
*/
extern char *GetDeleteMsgStr(); /* defined in warn_msg.c */
extern void PutErrorDialog();   /* defined in utility.c */
extern void UndoLastDraw();     /* defined in undo.c */

/*
The file delete.c consists of following functions.
For details, refer explanation before the declaration of each
function.
*/
void DeleteObject();
void ParseLinkLabel();


/*
File name:   delete.c
Description: It contains the functions required to implement the option
             Delete Object.
*/

#include "delete.h"

/*
Deleteobject() function is called whenever the mouse is clicked to
select an object for deletion. It checks if any valid object
is selected, if so, deletes it.
*/

void DeleteObject(w, data)
Widget        w;
graphics_data *data;
{
  TRANSITION  *t_temp;  /* used to traverse the list of transitions */
  PLACE       *p_temp;  /* used to traverse the list of places */
  int  x1, x2, y1, y2, xpos, ypos;  /* variables to store coordinates */
  int  i, j,  delete_pos = -1;
  float  floatquant1, floatquant2;
  int    intquant1, intquant2;
  char   mesg[MSG_LENGTH];

  xpos = data->x1;
  ypos = data->y1;
  for(i = 0; i < data->next_pos; i++) {
    x1 = data->buffer[i].x1;
    x2 = data->buffer[i].x2;
    y1 = data->buffer[i].y1;
    y2 = data->buffer[i].y2;
    if(((xpos >= x1) && (xpos <= x2) && (ypos >= y1) && (ypos <= y2))||
       ((xpos <= x1) && (xpos >= x2) && (ypos >= y1) && (ypos <= y2))||
```

```
      ((xpos >= x1) && (xpos <= x2) && (ypos <= y1) && (ypos >= y2))||
      ((xpos <= x1) && (xpos >= x2) && (ypos <= y1) && (ypos >= y2))){

    if((x2 - x1) != 0) {
      floatquant1 = (y2-y1)/((x2-x1)*1.0);
    }
    else
      floatquant1 = y2 - y1;

    if((x2 - xpos != 0) && (y2 - ypos != 0) &&
       (abs(x2 - xpos) > abs(x1 - xpos))) {
      floatquant2 = (1.0*(y2-ypos))/(x2-xpos);
    }
    else if((x1 - xpos != 0) && (y1 - ypos != 0) &&
       (abs(x1 - xpos) > abs(x2 - xpos))) {
      floatquant2 = (1.0*(y1-ypos))/(x1-xpos);
    }
    else
      floatquant2 = floatquant1;

    if((floatquant1 < 1.0) && (floatquant1 > -1.0)) {
      floatquant1 = 100*floatquant1;
      floatquant1 = floatquant1 / 10;
    }
    else if((floatquant1 >= 10.0) || (floatquant1 <= -10.0))
      floatquant1 = floatquant1/10;

    if((floatquant2 < 1.0)  && (floatquant2 > -1.0)){
      floatquant2 = 100*floatquant2;
      floatquant2 = floatquant2 / 10;
    }
    else if((floatquant2 >= 10.0) || (floatquant2 <= -10.0))
      floatquant2 = floatquant2/10;

    intquant1 = floatquant1;
    intquant2 = floatquant2;
    intquant1 = abs(intquant1);
    intquant2 = abs(intquant2);
    /* if the selected object is a link */
    if((strcmp(data->buffer[i].object,"LINK") == 0) ||
       (strcmp(data->buffer[i].object,"START_LINK") == 0)) {
      if((intquant1 == intquant2) || (intquant1 == (intquant2+1)) ||
         (intquant1 == (intquant2-1))) {
        delete_pos = i;
        i = data->next_pos;
      }
    }
    /* if the selected object is other than a link */
    else if(strcmp(data->buffer[i].object,"PUT_TOKEN") != 0) {
      delete_pos = i;
      i = data->next_pos;
    }
  }
}
if (delete_pos != -1) { /* if a valid object is selected */
  /* if the object selected is a transition */
  if(strcmp(data->buffer[delete_pos].object,"TRANSITION") == 0) {
    t_temp = T_LIST->HEAD;
    while((strcmp(t_temp->label,data->buffer[delete_pos].label) != 0) &&
          (t_temp != NULL))
      t_temp = t_temp->next;
    if(t_temp != NULL) {
      /* links should be deleted before deleting a transition */
      if((t_temp->cur_preset_no > 0) || (t_temp->cur_postset_no > 0)) {
        strcpy(mesg,GetDeleteMsgStr("DEL_LINKS"));
        PutErrorDialog(w,mesg);
        return;
      }
    }
    UNDO_ACTION = UNDO_DEL_TRANS;
    strcpy(T_LIST->trans_selected,data->buffer[delete_pos].label);
  }
  /* if the object selected is a place */
  else if(strcmp(data->buffer[delete_pos].object,"PLACE") == 0) {
    p_temp = P_LIST->HEAD;
    while((strcmp(p_temp->label,data->buffer[delete_pos].label) != 0) &&
          (p_temp != NULL))
      p_temp = p_temp->next;
    if(p_temp != NULL) {
      /* links should be deleted before deleting a place */
      if((p_temp->cur_preset_no > 0) || (p_temp->cur_postset_no > 0)) {
```

```
            strcpy(mesg,GetDeleteMsgStr("DEL_LINKS"));
            PutErrorDialog(w,mesg);
            return;
          }
        }
        UNDO_ACTION = UNDO_DEL_PLACE;
        strcpy(P_LIST->place_selected,data->buffer[delete_pos].label);
        P_LIST->last_token = p_temp->no_of_tokens;
      }
      /* if the selected object is a link */
      else if((strcmp(data->buffer[delete_pos].object,"LINK") == 0) ||
              (strcmp(data->buffer[delete_pos].object,"START_LINK") == 0)) {
        UNDO_ACTION = UNDO_DEL_LINK;
        ParseLinkLabel(data->buffer[delete_pos].label);
        UndoLastDraw(w,data);
        return;
      }
      i = delete_pos;
      j = data->next_pos;  /* saves the object for UNDO operation */
      strcpy(data->buffer[j].label,data->buffer[i].label);
      strcpy(data->buffer[j].object,data->buffer[i].object);
      data->buffer[j].x1    = data->buffer[i].x1;
      data->buffer[j].y1    = data->buffer[i].y1;
      data->buffer[j].x2    = data->buffer[i].x2;
      data->buffer[j].y2    = data->buffer[i].y2;
      data->buffer[j].gc    = data->buffer[i].gc;
      data->buffer[j].func = data->buffer[i].func;

      /* if the object is a place then save the number of tokens */
      if(strcmp(data->buffer[delete_pos].object,"PLACE") == 0) {
        strcpy(data->buffer[j+1].label,data->buffer[i+1].label);
        strcpy(data->buffer[j+1].object,data->buffer[i+1].object);
        data->buffer[j+1].x1    = data->buffer[i+1].x1;
        data->buffer[j+1].y1    = data->buffer[i+1].y1;
        data->buffer[j+1].x2    = data->buffer[i+1].x2;
        data->buffer[j+1].y2    = data->buffer[i+1].y2;
        data->buffer[j+1].gc    = data->buffer[i+1].gc;
        data->buffer[j+1].func = data->buffer[i+1].func;
      }
      /* deletes the object */
      if(strcmp(data->buffer[delete_pos].object,"PLACE") == 0) {
        for(i = delete_pos; i < data->next_pos; i++) {
          strcpy(data->buffer[i].label,data->buffer[i+2].label);
          strcpy(data->buffer[i].object,data->buffer[i+2].object);
          data->buffer[i].x1    = data->buffer[i+2].x1;
          data->buffer[i].y1    = data->buffer[i+2].y1;
          data->buffer[i].x2    = data->buffer[i+2].x2;
          data->buffer[i].y2    = data->buffer[i+2].y2;
          data->buffer[i].gc    = data->buffer[i+2].gc;
          data->buffer[i].func = data->buffer[i+2].func;
        }
      }
      else {
        for(i = delete_pos; i < data->next_pos; i++) {
          strcpy(data->buffer[i].label,data->buffer[i+1].label);
          strcpy(data->buffer[i].object,data->buffer[i+1].object);
          data->buffer[i].x1    = data->buffer[i+1].x1;
          data->buffer[i].y1    = data->buffer[i+1].y1;
          data->buffer[i].x2    = data->buffer[i+1].x2;
          data->buffer[i].y2    = data->buffer[i+1].y2;
          data->buffer[i].gc    = data->buffer[i+1].gc;
          data->buffer[i].func = data->buffer[i+1].func;
        }
      }
      UndoLastDraw(w,data,(XtPointer)1);
    }
}


/*
ParseLinkLabel() parses the link label and stores the labels of
a place and a transition involved in this link. This information
is used in undo action.
*/
void ParseLinkLabel(link_label)
char  *link_label;
{
  char  temp_buf[LABEL_LENGTH],
        *p;

  strcpy(temp_buf,link_label);
```

```
    p = strtok(temp_buf,"~~ \n");
    if(strcmp(p,"PLACE=>") == 0) {
      p = strtok(NULL,"~~ \n");
      strcpy(P_LIST->place_selected,p);
      P_LIST->pre_post = POSTSET;
    }
    else if(strcmp(p,"TRANSITION=>") == 0) {
      p = strtok(NULL,"~~ \n");
      strcpy(T_LIST->trans_selected,p);
      T_LIST->pre_post = POSTSET;
    }
    p = strtok(NULL,"~~ \n");
    if(strcmp(p,"PLACE=>") == 0) {
      p = strtok(NULL,"~~ \n");
      strcpy(P_LIST->place_selected,p);
      P_LIST->pre_post = PRESET;
    }
    else if(strcmp(p,"TRANSITION=>") == 0) {
      p = strtok(NULL,"~~ \n");
      strcpy(T_LIST->trans_selected,p);
      T_LIST->pre_post = PRESET;
    }
    p = strtok(NULL,"~~ \n");
    strcpy(T_LIST->del_link,p);
}


/*
File name:   put_token.h
Description: It contains the header files required for the file put_token.c.
             It also declares the external functions and the functions
             defined in the file put_token.c
*/

#include <stdio.h>
#include <string.h>
#include <Xm/Text.h>
#include <Xm/BulletinB.h>
#include <Xm/PushB.h>
#include <Xm/MessageB.h>
#include "data_str.h"
#include "global.h"

/*
These externally defined functions are called in the functions
defined in the file put_token.c.
*/
extern void RewriteAction();       /* defined in utility.c */
extern void PutErrorDialog();      /* defined in utility.c */
extern void RedrawDrawingBoard(); /* defined in utility.c */
extern void TrimString();          /* defined in utility.c */
extern void ResetCursor();         /* defined in cursor.c */
extern char *GetTokMsgStr();       /* defined in warn_msg.c */

/*
The file put_token.c consists of following functions.
For details, refer explanation before the declaration of each
function.
*/
void GetTokenInfo();
void DoneTokenInfo();
void DecrementToken();
void IncrementToken();


/*
File name:   put_token.c
Description: It contains the funstions required to implement the option
             PUT_TOKEN.
*/

#include "put_token.h"


/*
GetTokenInfo() pops up a dialog which prompts the user for the label
of a place where token assignment is to be made. It provides three
buttons, INCREMENT, DECREMENT, and DONE. Tokens can be added or
removed by clicking the INCREMENT or DECREMENT buttons and DONE
is to pop down the dialog box.
*/
```

```
void GetTokenInfo(parent, data)
Widget   parent;
graphics_data  *data;
{
  Widget  board, label1, heading1, increment_button, decrement_button,
          done_button;
  Arg     wargs[10];
  int     i, n;
  XmString  str;

  /* defines the PUT TOKEN dialog box */
  str = XmStringCreate("PUT TOKEN",XmSTRING_DEFAULT_CHARSET);
  n = 0;
  XtSetArg(wargs[n],XmNdialogTitle,str); n++;
  XtSetArg(wargs[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL); n++;
  XtSetArg(wargs[n],XmNautoUnmanage,False); n++;
  XtSetArg(wargs[n],XmNnoResize,True); n++;
  board = XmCreateBulletinBoardDialog(parent, "board", wargs, n);
  XmStringFree(str);
  /* creates a text widget */
  n = 0;
  XtSetArg(wargs[n],XmNx,130); n++;
  XtSetArg(wargs[n],XmNy,10); n++;
  label1 = XtCreateManagedWidget("label1", xmTextWidgetClass, board, wargs, n);
  /* creates a label widget */
  n = 0;
  str = XmStringCreate("PUT TOKEN AT PLACE",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNx,10); n++;
  XtSetArg(wargs[n],XmNy,15); n++;
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNtraversalOn,False); n++;
  heading1 = XtCreateManagedWidget("heading1", xmLabelWidgetClass,
                          board, wargs, n);
  XmStringFree(str);
  /* creates the INCREMENT button */
  n = 0;
  str = XmStringCreate("INCREMENT",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNx,10); n++;
  XtSetArg(wargs[n],XmNy,90); n++;
  XtSetArg(wargs[n],XmNuserData,label1); n++;
  XtSetArg(wargs[n],XmNshadowThickness,4); n++;
  increment_button = XtCreateManagedWidget("increment", xmPushButtonWidgetClass,
                                  board, wargs, n);
  XmStringFree(str);
  /* adds a callback function */
  XtAddCallback(increment_button, XmNactivateCallback, IncrementToken,
                data);
  /* creates the DECREMENT button */
  n = 0;
  str = XmStringCreate("DECREMENT",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNx,90); n++;
  XtSetArg(wargs[n],XmNy,90); n++;
  XtSetArg(wargs[n],XmNuserData,label1); n++;
  XtSetArg(wargs[n],XmNshadowThickness,4); n++;
  decrement_button = XtCreateManagedWidget("decrement",
                                          xmPushButtonWidgetClass,
                                          board, wargs, n);
  XmStringFree(str);
  /* adds a callback function */
  XtAddCallback(decrement_button, XmNactivateCallback, DecrementToken,data);
  /* creates the DONE button */
  n = 0;
  str = XmStringCreate("DONE",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNx,170); n++;
  XtSetArg(wargs[n],XmNy,90); n++;
  XtSetArg(wargs[n],XmNshadowThickness,4); n++;
  done_button = XtCreateManagedWidget("done", xmPushButtonWidgetClass,
                                  board, wargs, n);
  XmStringFree(str);
  /* adds a callback function */
  XtAddCallback(done_button, XmNactivateCallback, DoneTokenInfo,NULL);
  XtManageChild(board);
  XmProcessTraversal(label1,XmTRAVERSE_CURRENT); /* focus to first entry */
  ResetCursor(toplevel);
}

/*
```

```
DoneTokenInfo() pops down the PUT TOKEN dialog box.
*/

void DoneTokenInfo(w, data, call_data)
Widget                  w;
XtPointer               data;
XmAnyCallbackStruct   *call_data;
{
  strcpy(action_selected,"NOTHING");
  RewriteAction();
  XtUnmanageChild(XtParent(w));
}


/*
DecrementToken() decrements the number of tokens present in
a place by one.
*/

void DecrementToken(w, data, call_data)
Widget                  w;
graphics_data           *data;
XmAnyCallbackStruct   *call_data;
{
  Widget     info;
  PLACE      *p_temp;
  TRANSITION *t_temp;
  Arg        wargs[MAX_ARGS];
  char       p_label[300],
             *buf,
             mesg[MSG_LENGTH];

  XtSetArg(wargs[0], XmNuserData, &info);
  XtGetValues(w, wargs, 1);
  buf = XmTextGetString(info);
  strncpy(p_label,buf,300);
  XtFree(buf);
  TrimString(p_label);
  if(strlen(p_label) == 0) { /* a null label entered */
    strcpy(mesg,GetTokMsgStr("NULL_LABEL"));
    PutErrorDialog(w, mesg);
    return;
  }
  t_temp = T_LIST->HEAD;
  while((strcmp(t_temp->label,p_label) != 0) && (t_temp != NULL))
    t_temp = t_temp->next;
  if(t_temp != NULL) { /* the label is a transition */
    strcpy(mesg,GetTokMsgStr("TRANS_LABEL"));
    PutErrorDialog(w, mesg);
    return;
  }
  p_temp = P_LIST->HEAD;
  while((strcmp(p_temp->label,p_label) != 0) && (p_temp != NULL)) {
    p_temp = p_temp->next;
  }

  if(p_temp == NULL) { /* the label is not found */
    strcpy(mesg,GetTokMsgStr("LB_NOT_FOUND"));
    PutErrorDialog(w, mesg);
    return;
  }
  if(p_temp->no_of_tokens == 0) { /* number of tokens cannot be negative */
    strcpy(mesg,GetTokMsgStr("NO_NEG_TOK"));
    PutErrorDialog(w, mesg);
    return;
  }
  p_temp->no_of_tokens--; /* decrements the number of tokens */
  RedrawDrawingBoard(data); /* draws the drawing area */
}


/*
IncrementtToken() increments the number of tokens present in
a place by one.
*/

void IncrementToken(w, data, call_data)
Widget                  w;
graphics_data           *data;
XmAnyCallbackStruct   *call_data;
{
```

```
  Widget      info;
  PLACE       *p_temp;
  TRANSITION *t_temp;
  Arg         wargs[MAX_ARGS];
  char        p_label[300],
              *buf,
              mesg[MSG_LENGTH];

  XtSetArg(wargs[0], XmNuserData, &info);
  XtGetValues(w, wargs, 1);
  buf = XmTextGetString(info);
  strncpy(p_label,buf,300);
  XtFree(buf);
  TrimString(p_label);
  if(strlen(p_label) == 0) { /* a null label entered */
    strcpy(mesg,GetTokMsgStr("NULL_LABEL"));
    PutErrorDialog(w, mesg);
    return;
  }
  t_temp = T_LIST->HEAD;
  while((strcmp(t_temp->label,p_label) != 0) && (t_temp != NULL))
    t_temp = t_temp->next;

  if(t_temp != NULL) { /* the label is a transition */
    strcpy(mesg,GetTokMsgStr("TRANS_LABEL"));
    PutErrorDialog(w, mesg);
    return;
  }
  p_temp = P_LIST->HEAD;
  while((strcmp(p_temp->label,p_label) != 0) && (p_temp != NULL)) {
    p_temp = p_temp->next;
  }
  if(p_temp == NULL) { /* the label is not found */
    strcpy(mesg,GetTokMsgStr("LB_NOT_FOUND"));
    PutErrorDialog(w, mesg);
    return;
  }
  p_temp->no_of_tokens++; /* increments the number of tokens */
  RedrawDrawingBoard(data); /* draws the drawing area */
}


/*
File name:   spec.h
Description: It contains the header files required for the file spec.c.
             It also declares the external functions and the functions
             defined in the file spec.c
*/

#include <string.h>
#include <Xm/Text.h>
#include <Xm/BulletinB.h>
#include <Xm/PushB.h>
#include <Xm/MessageB.h>
#include "data_str.h"
#include "global.h"

/*
These externally defined functions are called in the functions
defined in file spec.c.
*/
extern void SetBusyCursor(); /* defined in cursor.c */
extern void SetHandCursor(); /* defined in cursor.c */
extern void ResetCursor();   /* defined in cursor.c */
extern AllTrim();            /* defined in utility.c */


/*
The file spec.c consists of following functions.
For details, refer explanation before the declaration of each
function.
*/
void SelectNodeForSpec();
void EditSpec();
void CancelEditSpec();
void DoneEditSpec();
void ShowSpec();
void CancelShowSpec();


/*
```

```
File name:   spec.c
Description: It contains the functions required to implement the option
             EDIT_SPEC and SHOW_SPEC.
*/

#include "spec.h"

/*
SelectNodeForSpec() function is called whenever mouse is clicked to
select an object for EDIT SPEC or SHOW SPEC. It checks if any
valid object is selected, if so, then EditSpec() or ShowSpec() is
called.
*/

void SelectNodeForSpec(data,flag)
graphics_data   *data;
int             flag;
{
  PLACE       *p_temp;
  TRANSITION  *t_temp;
  int    x1, x2, y1, y2, xpos, ypos;
  int    i, pos = -1;

  xpos = data->x1;
  ypos = data->y1;
  for(i = 0; i < data->next_pos; i++) {
    x1 = data->buffer[i].x1;
    x2 = data->buffer[i].x2;
    y1 = data->buffer[i].y1;
    y2 = data->buffer[i].y2;
    if(((xpos >= x1) && (xpos <= x2) && (ypos >= y1) && (ypos <= y2))||
       ((xpos <= x1) && (xpos >= x2) && (ypos >= y1) && (ypos <= y2))||
       ((xpos >= x1) && (xpos <= x2) && (ypos <= y1) && (ypos >= y2))||
       ((xpos <= x1) && (xpos >= x2) && (ypos <= y1) && (ypos >= y2))){

      /* checks if the object selected is a place or a transition */
      if((strcmp(data->buffer[i].object,"TRANSITION") == 0) ||
        (strcmp(data->buffer[i].object,"PLACE") == 0)) {
        pos = i;
        i = data->next_pos;
      }
    }
  }
  if (pos != -1) {
    /* if the object is a transition */
    if(strcmp(data->buffer[pos].object,"TRANSITION") == 0) {
      t_temp = T_LIST->HEAD;
      while((strcmp(t_temp->label,data->buffer[pos].label) != 0) &&
            (t_temp != NULL))
        t_temp = t_temp->next;
      if(t_temp != NULL) {
        SetBusyCursor(drawing);
        if(flag == EDIT_SPEC) /* EditSpec() is called */
          EditSpec(t_temp->interpretation, t_temp->label, "TRANSITION");
        else /* ShowSpec() is called */
          ShowSpec(t_temp->interpretation, t_temp->label, "TRANSITION");
      }
    }
    /* if the object is a place */
    else if(strcmp(data->buffer[pos].object,"PLACE") == 0) {
      p_temp = P_LIST->HEAD;
      while((strcmp(p_temp->label, data->buffer[pos].label) != 0) &&
            (p_temp != NULL)) {
        p_temp = p_temp->next;
      }
      if(p_temp != NULL) {
        SetBusyCursor(drawing);
        if(flag == EDIT_SPEC) /* EditSpec() is called */
          EditSpec(p_temp->interpretation, p_temp->label, "PLACE");
        else /* ShowSpec() is called */
          ShowSpec(p_temp->interpretation, p_temp->label, "PLACE");
      }
    }
  }
}


/*
EditSpec() pops up a dialog box displaying the information
about the selected node and allows the user to type any
interpretation for that node.
```

```
*/

void EditSpec(descrip, node_label, node_type)
char     *descrip,
         *node_label,
         *node_type;
{
  static Widget board = NULL,
         type_info = NULL, label_info = NULL, heading1 = NULL,
         heading2  = NULL, heading3   = NULL, cancel_button = NULL,
         text_w    = NULL,
         save_button = NULL;
  Arg     wargs[MAX_ARGS];
  int     n;
  XmString  str;

  if(!board) {
    /* defines the EDIT SPECIFICATION dialog box */
    n = 0;
    str = XmStringCreate("EDIT SPECIFICATION",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNdialogTitle,str); n++;
    XtSetArg(wargs[n],XmNdefaultPosition,False); n++;
    XtSetArg(wargs[n],XmNx,0); n++;
    XtSetArg(wargs[n],XmNy,500); n++;
    XtSetArg(wargs[n],XmNwidth,350); n++;
    XtSetArg(wargs[n],XmNheight,220); n++;
    XtSetArg(wargs[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL); n++;
    XtSetArg(wargs[n],XmNautoUnmanage,False); n++;
    XtSetArg(wargs[n],XmNresizePolicy,XmRESIZE_NONE); n++;
    XtSetArg(wargs[n],XmNnoResize,True); n++;
    board = XmCreateBulletinBoardDialog(drawing, "board", wargs, n);
    XmStringFree(str);
    /* creates a label widget */
    n = 0;
    str = XmStringCreate("TYPE OF NODE:",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNx,10); n++;
    XtSetArg(wargs[n],XmNy,15); n++;
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNtraversalOn,False); n++;
    heading1 = XtCreateManagedWidget("heading1", xmLabelWidgetClass,
                            board, wargs, n);
    XmStringFree(str);
    /* creates a label widget */
    n = 0;
    str = XmStringCreate(node_type,XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNx,110); n++;
    XtSetArg(wargs[n],XmNy,15); n++;
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNtraversalOn,False); n++;
    type_info = XtCreateManagedWidget("type_info",
                                      xmLabelWidgetClass, board, wargs, n);
    XmStringFree(str);
    /* creates a label widget */
    n = 0;
    str = XmStringCreate("NODE LABEL:",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNx,10); n++;
    XtSetArg(wargs[n],XmNy,55); n++;
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNtraversalOn,False); n++;
    heading2 = XtCreateManagedWidget("heading2", xmLabelWidgetClass,
                            board, wargs, n);
    XmStringFree(str);
    /* creates a label widget */
    n = 0;
    str = XmStringCreate(node_label,XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNx,110); n++;
    XtSetArg(wargs[n],XmNy,55); n++;
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNtraversalOn,False); n++;
    label_info = XtCreateManagedWidget("label_info",
                                       xmLabelWidgetClass,
                                       board, wargs, n);
    XmStringFree(str);
    /* creates a label widget */
    n = 0;
    str = XmStringCreate("INTERPRETATION:",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNx,10); n++;
    XtSetArg(wargs[n],XmNy,95); n++;
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNtraversalOn,False); n++;
    heading3 = XtCreateManagedWidget("heading3", xmLabelWidgetClass,
```

```
                                board, wargs, n);
    XmStringFree(str);
    /* creates a widget to edit interpretation */
    n = 0;
    XtSetArg(wargs[n],XmNeditable,True); n++;
    XtSetArg(wargs[n],XmNcursorPositionVisible,True); n++;
    XtSetArg(wargs[n],XmNx,110); n++;
    XtSetArg(wargs[n],XmNy,95); n++;
    XtSetArg(wargs[n],XmNrows,5); n++;
    XtSetArg(wargs[n],XmNcolumns,30); n++;
    XtSetArg(wargs[n],XmNmaxLength,INFO_LENGTH-2); n++;
    XtSetArg(wargs[n],XmNwordWrap,True); n++;
    XtSetArg(wargs[n],XmNscrollHorizontal,False); n++;
    XtSetArg(wargs[n],XmNeditMode,XmMULTI_LINE_EDIT); n++;
    text_w = XmCreateScrolledText(board, "text_w",wargs, n);
    XmTextSetString(text_w, descrip);
    /* creates a button widget */
    n = 0;
    str = XmStringCreate("SAVE",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNx,80); n++;
    XtSetArg(wargs[n],XmNy,185); n++;
    XtSetArg(wargs[n],XmNuserData,descrip); n++;
    XtSetArg(wargs[n],XmNshadowThickness,4); n++;
    save_button = XtCreateManagedWidget("done", xmPushButtonWidgetClass,
                                board, wargs, n);
    XmStringFree(str);
    XtAddCallback(save_button, XmNactivateCallback, DoneEditSpec,
                  text_w);
    /* creates a button widget */
    n = 0;
    str = XmStringCreate("CANCEL",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNx,10); n++;
    XtSetArg(wargs[n],XmNy,185); n++;
    XtSetArg(wargs[n],XmNshadowThickness,4); n++;
    cancel_button = XtCreateManagedWidget("cancel", xmPushButtonWidgetClass,
                                    board, wargs, n);
    XmStringFree(str);
    XtAddCallback(cancel_button, XmNactivateCallback,
                                 CancelEditSpec,save_button);
    XtManageChild(text_w);
    XtManageChild(board);
    XmProcessTraversal(text_w,XmTRAVERSE_CURRENT); /* focus to first entry */
    SetHandCursor(drawing);
  }
  else { /* if the dialog already exists */
    n = 0;
    str = XmStringCreate(node_type,XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetValues(type_info,wargs,n);
    XmStringFree(str);
    n = 0;
    str = XmStringCreate(node_label,XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetValues(label_info,wargs,n);
    XmStringFree(str);
    n = 0;
    XtSetArg(wargs[n],XmNuserData,descrip); n++;
    XtSetValues(save_button,wargs,n);
    XmTextSetString(text_w, descrip);
    XtManageChild(board);
    XmProcessTraversal(text_w,XmTRAVERSE_CURRENT); /* focus to first entry */
    SetHandCursor(drawing);
  }
}


/*
CancelEditSpec(), as a callback function pops down the dialog box.
*/

void CancelEditSpec(w, save_button, call_data)
Widget               w;
Widget            save_button;
XmAnyCallbackStruct  *call_data;
{
  XtUnmanageChild(XtParent(w));
}
```

```
/*
DoneEditSpec() as a callback function saves the interpretation
typed by the user and pops down the dialog box.
*/

void DoneEditSpec(w, t_w, call_data)
Widget               w;
Widget               t_w;
XmAnyCallbackStruct  *call_data;
{
  char  buffer[INFO_LENGTH];
  char  *descrip;
  Arg   wargs[MAX_ARGS];

  XtSetArg(wargs[0], XmNuserData, &descrip);
  XtGetValues(w, wargs, 1);
  strcpy(descrip,XmTextGetString(t_w));
  AllTrim(descrip);
  XtUnmanageChild(XtParent(w));
}


/*
ShowSpec() pops up a dialog box displaying the information
about the selected node. It does not allow the user to change
the interpretation for that node.
*/

void ShowSpec(descrip, node_label, node_type)
char     *descrip,
         *node_label,
         *node_type;
{
  static Widget board = NULL,
         type_info = NULL, label_info = NULL, heading1 = NULL,
         heading2  = NULL, heading3   = NULL, cancel_button = NULL,
         text_w    = NULL,
         save_button = NULL;
  Arg    wargs[MAX_ARGS];
  int    n;
  XmString  str;

  if(!board) {
    /* defines the SHOW SPECIFICATION dialog box */
    n = 0;
    str = XmStringCreate("SHOW SPECIFICATION",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNdialogTitle,str); n++;
    XtSetArg(wargs[n],XmNdefaultPosition,False); n++;
    XtSetArg(wargs[n],XmNx,0); n++;
    XtSetArg(wargs[n],XmNy,500); n++;
    XtSetArg(wargs[n],XmNwidth,350); n++;
    XtSetArg(wargs[n],XmNheight,220); n++;
    XtSetArg(wargs[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL); n++;
    XtSetArg(wargs[n],XmNautoUnmanage,False); n++;
    XtSetArg(wargs[n],XmNresizePolicy,XmRESIZE_NONE); n++;
    XtSetArg(wargs[n],XmNnoResize,True); n++;
    board = XmCreateBulletinBoardDialog(drawing, "board", wargs, n);
    XmStringFree(str);
    /* creates a label widget */
    n = 0;
    str = XmStringCreate("TYPE OF NODE:",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNx,10); n++;
    XtSetArg(wargs[n],XmNy,15); n++;
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNtraversalOn,False); n++;
    heading1 = XtCreateManagedWidget("heading1", xmLabelWidgetClass,
                          board, wargs, n);
    XmStringFree(str);
    /* creates a label widget */
    n = 0;
    str = XmStringCreate(node_type,XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNx,110); n++;
    XtSetArg(wargs[n],XmNy,15); n++;
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNtraversalOn,False); n++;
    type_info = XtCreateManagedWidget("type_info",
                                  xmLabelWidgetClass, board, wargs, n);
    XmStringFree(str);
    /* creates a label widget */
    n = 0;
    str = XmStringCreate("NODE LABEL:",XmSTRING_DEFAULT_CHARSET);
```

```
    XtSetArg(wargs[n],XmNx,10); n++;
    XtSetArg(wargs[n],XmNy,55); n++;
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNtraversalOn,False); n++;
    heading2 = XtCreateManagedWidget("heading2", xmLabelWidgetClass,
                          board, wargs, n);
    XmStringFree(str);
    /* creates a label widget */
    n = 0;
    str = XmStringCreate(node_label,XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNx,110); n++;
    XtSetArg(wargs[n],XmNy,55); n++;
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNtraversalOn,False); n++;
    label_info = XtCreateManagedWidget("label_info",
                                       xmLabelWidgetClass,
                                       board, wargs, n);
    XmStringFree(str);
    /* creates a label widget */
    n = 0;
    str = XmStringCreate("INTERPRETATION:",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNx,10); n++;
    XtSetArg(wargs[n],XmNy,95); n++;
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNtraversalOn,False); n++;
    heading3 = XtCreateManagedWidget("heading3", xmLabelWidgetClass,
                          board, wargs, n);
    XmStringFree(str);
    /* creates a widget to display interpretation */
    n = 0;
    XtSetArg(wargs[n],XmNeditable,False); n++;
    XtSetArg(wargs[n],XmNcursorPositionVisible,False); n++;
    XtSetArg(wargs[n],XmNx,110); n++;
    XtSetArg(wargs[n],XmNy,95); n++;
    XtSetArg(wargs[n],XmNrows,5); n++;
    XtSetArg(wargs[n],XmNcolumns,30); n++;
    XtSetArg(wargs[n],XmNmaxLength,INFO_LENGTH-2); n++;
    XtSetArg(wargs[n],XmNwordWrap,True); n++;
    XtSetArg(wargs[n],XmNscrollHorizontal,False); n++;
    XtSetArg(wargs[n],XmNeditMode,XmMULTI_LINE_EDIT); n++;
    text_w = XmCreateScrolledText(board, "text_w",wargs, n);
    XmTextSetString(text_w, descrip);
    /* creates a button widget */
    n = 0;
    str = XmStringCreate("DONE",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNx,10); n++;
    XtSetArg(wargs[n],XmNy,185); n++;
    XtSetArg(wargs[n],XmNshadowThickness,4); n++;
    cancel_button = XtCreateManagedWidget("cancel", xmPushButtonWidgetClass,
                                  board, wargs, n);
    XmStringFree(str);
    /* adds a callback function */
    XtAddCallback(cancel_button, XmNactivateCallback, CancelShowSpec,NULL);
    XtManageChild(text_w);
    XtManageChild(board);
    XmProcessTraversal(text_w,XmTRAVERSE_CURRENT); /* focus to first entry */
    SetHandCursor(drawing);
  }
  else { /* if widget already exists */
    n = 0;
    str = XmStringCreate(node_type,XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetValues(type_info,wargs,n);
    XmStringFree(str);
    n = 0;
    str = XmStringCreate(node_label,XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetValues(label_info,wargs,n);
    XmStringFree(str);
    XmTextSetString(text_w, descrip);
    XtManageChild(board);
    XmProcessTraversal(text_w,XmTRAVERSE_CURRENT); /* focus to first entry */
    SetHandCursor(drawing);
  }
}


/*
CancelShowSpec(), as a callback function pops down the dialog box.
*/
```

```
void CancelShowSpec(w, client_data, call_data)
Widget               w;
XtPointer            client_data;
XmAnyCallbackStruct  *call_data;
{
  XtUnmanageChild(XtParent(w));
}


/*
File name:   save.h
Description: It contains the header files required for the file save.c.
             It also declares the external functions and the functions
             defined in the file save.c
*/

#include <stdio.h>
#include <string.h>
#include <Xm/Text.h>
#include <Xm/BulletinB.h>
#include <Xm/PushB.h>
#include "data_str.h"
#include "global.h"

/*
These externally defined functions are called in the functions
defined in the file save.c.
*/
extern void DecodeFile();           /* defined in utility.c */
extern void TrimString();           /* defined in utility.c */
extern void RewriteAction();        /* defined in utility.c */
extern void RewriteFileName();      /* defined in utility.c */
extern void RedrawDrawingBoard();   /* defined in utility.c */
extern void ResetCurrentAction();   /* defined in utility.c */
extern void ResetCursor();          /* defined in cursor.c */
extern char *GetSaveMsgStr();       /* defined in warn_msg.c */
extern char *GetFileMsgStr();       /* defined in warn_msg.c */
extern void RemoveIncompleteLink(); /* defined in user_interface.c */


/*
The file save.c consists of following functions.
For details, refer explanation before the declaration of each
function.
*/
void save_object();
void SelectSaveFileOption();
void PreSaveFile();
void CancelSaveFile();
void DoneSaveFile();
void DoneSaveAs();
void SaveFile();
void SaveFileAs();
void CancelSaveAs();
void DoneSaveAs();


/*
File name:   save.c
Description: It contains the functions required to implement the options
             Save and Save As. It also contains the functions needed to
             save the file to the disk.
*/

#include "save.h"

/*
save_object():
Saves the information abount an object. This information is used to
redraw the object.
*/

void save_object(data)
graphics_data  *data;
{
  data->buffer[data->next_pos].x1 = data->x1;
  data->buffer[data->next_pos].y1 = data->y1;
  data->buffer[data->next_pos].x2 = data->x2;
  data->buffer[data->next_pos].y2 = data->y2;
  data->buffer[data->next_pos].gc = data->gc;
  data->buffer[data->next_pos].func = data->current_func;
```

```
  strcpy(data->buffer[data->next_pos].object,data->object);
  strcpy(data->buffer[data->next_pos].label,data->label);
  data->next_pos++;
}


/*
SelectSaveFileOption(), as a callback function calls PreSaveFile()
which saves the file or pops up a dialog if needed.
*/

void SelectSaveFileOption(parent, data, call_data)
Widget         parent;
graphics_data  *data;
XtPointer      call_data;
{
  ResetCurrentAction(data);
  PreSaveFile(parent, data);
}


/*
PreSaveFile() saves the current file. If the filename is not given,
it pops up a dialog box to enter a filename.
*/

void PreSaveFile(parent, data)
Widget         parent;
graphics_data  *data;
{
  Widget  board, label1, heading1, done_button, cancel_button;
  Arg     wargs[MAX_ARGS];
  int     n;
  XmString  str;

  ResetCursor(drawing);
  strcpy(action_selected,"SAVE FILE");
  RewriteAction();
  if(strcmp(GLOBAL_FILE_NAME,"Untitled") != 0) {
    strcpy(action_selected,"Saving...");
    RewriteAction();
    SaveFile(data); /* saves the file */
    strcpy(action_selected,"NOTHING"); /* updates the action selected */
    RewriteAction();
    return;
  }
  /* defines a dialog box titled SAVE FILE */
  str = XmStringCreate("SAVE FILE",XmSTRING_DEFAULT_CHARSET);
  n = 0;
  XtSetArg(wargs[n],XmNdialogTitle,str); n++;
  XtSetArg(wargs[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL); n++;
  XtSetArg(wargs[n],XmNautoUnmanage,False); n++;
  XtSetArg(wargs[n],XmNnoResize,True); n++;
  board = XmCreateBulletinBoardDialog(parent, "board", wargs, n);
  XmStringFree(str);
  /* creates a text widget */
  n = 0;
  XtSetArg(wargs[n],XmNx,130); n++;
  XtSetArg(wargs[n],XmNy,10); n++;
  XtSetArg(wargs[n],XmNvalue,GLOBAL_FILE_NAME); n++;
  label1 = XtCreateManagedWidget("label1", xmTextWidgetClass, board, wargs, n);
  /* creates a label widget */
  n = 0;
  str = XmStringCreate("ENTER FILENAME ",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNx,10); n++;
  XtSetArg(wargs[n],XmNy,15); n++;
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNtraversalOn,False); n++;
  heading1 = XtCreateManagedWidget("heading1", xmLabelWidgetClass,
                              board, wargs, n);
  XmStringFree(str);
  /* creates a button widget */
  n = 0;
  str = XmStringCreate("DONE",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNx,10); n++;
  XtSetArg(wargs[n],XmNy,90); n++;
  XtSetArg(wargs[n],XmNuserData,label1); n++;
  XtSetArg(wargs[n],XmNshadowThickness,4); n++;
  done_button = XtCreateManagedWidget("done", xmPushButtonWidgetClass,
                                      board, wargs, n);
```

```
  XmStringFree(str);
  /* adds a callback function */
  XtAddCallback(done_button, XmNactivateCallback, DoneSaveFile,
                data);
  /* creates a button widget */
  n = 0;
  str = XmStringCreate("CANCEL",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNx,50); n++;
  XtSetArg(wargs[n],XmNy,90); n++;
  XtSetArg(wargs[n],XmNshadowThickness,4); n++;
  cancel_button = XtCreateManagedWidget("cancel", xmPushButtonWidgetClass,
                                        board, wargs, n);
  XmStringFree(str);
  /* adds a callback function */
  XtAddCallback(cancel_button, XmNactivateCallback, CancelSaveFile,NULL);
  XtManageChild(board);
  XmProcessTraversal(label1,XmTRAVERSE_CURRENT); /* focus to first entry */
}


/*
CancelSaveFile(), as a callback function pops down the SAVE FILE
dialog box.
*/

void CancelSaveFile(w, data, call_data)
Widget                w;
XtPointer             data;
XmAnyCallbackStruct   *call_data;
{
  strcpy(action_selected,"NOTHING");
  RewriteAction();
  XtUnmanageChild(XtParent(w));
}


/*
DoneSaveFile() as a callback function saves the file when DONE
button is clicked after entering the filename.
*/

void DoneSaveFile(w, data, call_data)
Widget                w;
graphics_data         *data;
XmAnyCallbackStruct   *call_data;
{
  FILE    *chk_file;
  Widget  info;
  Arg     wargs[MAX_ARGS];
  char    *f_name,
          mesg[MSG_LENGTH];

  XtSetArg(wargs[0], XmNuserData, &info);
  XtGetValues(w, wargs, 1);
  f_name = XmTextGetString(info);
  TrimString(f_name);
  if((strlen(f_name) == 0) ||
     (strcmp(f_name,"Untitled") == 0)) { /* null string for filename */
    strcpy(mesg,GetFileMsgStr("NULL_FILE"));
    PutErrorDialog(w,mesg);
    XtFree(f_name);
    return;
  }
  if((chk_file = fopen(f_name,"r")) != NULL) { /* file already exists */
    strcpy(mesg,GetFileMsgStr("FILE_EXISTS"));
    PutErrorDialog(w,mesg);
    XtFree(f_name);
    fclose(chk_file);
    return;
  }
  strcpy(GLOBAL_FILE_NAME,f_name); /* updates the filename */
  RewriteFileName(); /* displays the filename */
  strcpy(action_selected,"Saving...");
  RewriteAction();
  SaveFile(data); /* saves the file */
  XtFree(f_name);
  XtUnmanageChild(XtParent(w));
  strcpy(action_selected,"NOTHING"); /* updates the action selected */
  RewriteAction();
}
```

```
/*
SaveFile actually writes the information into a file.
*/

void SaveFile(data)
graphics_data  *data;
{
  FILE   *file_ptr;
  FILE   *fp1;
  PLACE       *p_temp;
  TRANSITION  *t_temp;
  int i;

  if(!(file_ptr = fopen("PETRINETTEMPFILE","w+"))) {
    printf("Cannot open file %s \n",GLOBAL_FILE_NAME);
    exit(0);
  }
  /* saves the data into a file */
  fprintf(file_ptr,"BUFFER_INFO\n");
  for(i = 0; i < data->next_pos; i++) {
    fprintf(file_ptr,"%s\n",data->buffer[i].object);
    fprintf(file_ptr,"%s\n",data->buffer[i].label);
    fprintf(file_ptr,"%d\n",data->buffer[i].x1);
    fprintf(file_ptr,"%d\n",data->buffer[i].x2);
    fprintf(file_ptr,"%d\n",data->buffer[i].y1);
    fprintf(file_ptr,"%d\n",data->buffer[i].y2);
  }
  /* saves the information from the P_LIST */
  fprintf(file_ptr,"PLACE_LIST_INFO\n");
  p_temp = P_LIST->HEAD;
  while(p_temp != NULL) {
    fprintf(file_ptr,"%s\n",p_temp->label); /* saves the label */
    fprintf(file_ptr,"PRESET_LIST\n");
    fprintf(file_ptr,"%d\n",p_temp->cur_preset_no);
    for(i = 0; i < p_temp->cur_preset_no; i++)     /* saves the preset */
      fprintf(file_ptr,"%s\n",p_temp->preset[i]);

    fprintf(file_ptr,"POSTSET_LIST\n");
    fprintf(file_ptr,"%d\n",p_temp->cur_postset_no);
    for(i = 0; i < p_temp->cur_postset_no; i++)    /* saves the postset */
      fprintf(file_ptr,"%s\n",p_temp->postset[i]);

    fprintf(file_ptr,"%d\n",p_temp->no_of_tokens); /* saves the number of tokens */
    if(strlen(p_temp->interpretation) > 0)     /* saves the interpretation */
      fprintf(file_ptr,"%s\n",p_temp->interpretation);
    fprintf(file_ptr,"END_INTERPRETATION\n");
    p_temp = p_temp->next;
  }
  /* saves the information from the P_LIST */
  fprintf(file_ptr,"TRANSITION_LIST_INFO\n");
  t_temp = T_LIST->HEAD;
  while(t_temp != NULL) {
    fprintf(file_ptr,"%s\n",t_temp->label);   /* saves the label */

    fprintf(file_ptr,"PRESET_LIST\n");
    fprintf(file_ptr,"%d\n",t_temp->cur_preset_no);
    for(i = 0; i < t_temp->cur_preset_no; i++)   /* saves the preset */
      fprintf(file_ptr,"%s\n",t_temp->preset[i]);

    fprintf(file_ptr,"POSTSET_LIST\n");
    fprintf(file_ptr,"%d\n",t_temp->cur_postset_no);
    for(i = 0; i < t_temp->cur_postset_no; i++)   /* saves the postset */
      fprintf(file_ptr,"%s\n",t_temp->postset[i]);

    fprintf(file_ptr,"%d\n",t_temp->enabled);
    if(strlen(t_temp->interpretation) > 0)   /* saves the interpretation */
      fprintf(file_ptr,"%s\n",t_temp->interpretation);
    fprintf(file_ptr,"END_INTERPRETATION\n");
    t_temp = t_temp->next;
  }
  fprintf(file_ptr,"GLOBAL_PTL_COUNTS\n");
  fprintf(file_ptr,"%d\n",PLACE_CNT);
  fprintf(file_ptr,"%d\n",TRANSITION_CNT);
  fprintf(file_ptr,"%d\n",LINK_CNT);
  fprintf(file_ptr,"END_OF_FILE\n");
  fclose(file_ptr); /* closes the file */
  if(!(file_ptr = fopen(GLOBAL_FILE_NAME,"w+"))) {
    printf("Cannot open file %s \n",GLOBAL_FILE_NAME);
    exit(0);
  }
```

```
  if(!(fp1 = fopen("PETRINETTEMPFILE","r"))) {
    printf("Cannot open file %s \n",GLOBAL_FILE_NAME);
    exit(0);
  }
  DecodeFile(file_ptr,fp1); /* decodes the file */
  fclose(file_ptr);
  fclose(fp1);
  system("rm PETRINETTEMPFILE"); /* removes the temporary file */
}


/*
SaveFileAs() pops up a dialog box titled SAVE AS FILE. It prompts
the user to enter a new filename. The current file is saved by
this name, without updating the current file.
*/

void SaveFileAs(parent, data, call_data)
Widget   parent;
graphics_data  *data;
XtPointer      call_data;
{
  Widget  board, label1,label2, heading1, heading2, done_button,
          cancel_button;
  Arg     wargs[10];
  int     i, n;
  XmString  str;
  EDIT_LABEL_STRUCT *info_to_send;

  ResetCurrentAction(data);
  ResetCursor(drawing);
  strcpy(action_selected,"SAVE AS FILE");
  RewriteAction();
  /* creates a dialog box */
  str = XmStringCreate("SAVE AS FILE",XmSTRING_DEFAULT_CHARSET);
  n = 0;
  XtSetArg(wargs[n],XmNdialogTitle,str); n++;
  XtSetArg(wargs[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL); n++;
  XtSetArg(wargs[n],XmNautoUnmanage,False); n++;
  XtSetArg(wargs[n],XmNnoResize,True); n++;
  board = XmCreateBulletinBoardDialog(parent, "board", wargs, n);
  XmStringFree(str);
  /* creates a text widget */
  n = 0;
  XtSetArg(wargs[n],XmNx,140); n++;
  XtSetArg(wargs[n],XmNy,10); n++;
  XtSetArg(wargs[n],XmNvalue,GLOBAL_FILE_NAME); n++;
  XtSetArg(wargs[n],XmNeditable,False); n++;
  label1 = XtCreateManagedWidget("label1", xmTextWidgetClass, board, wargs, n);
  /* creates a text widget */
  n = 0;
  XtSetArg(wargs[n],XmNx,140); n++;
  XtSetArg(wargs[n],XmNy,50); n++;
  label2 = XtCreateManagedWidget("label2", xmTextWidgetClass, board, wargs, n);
  /* creates a label widget */
  n = 0;
  str = XmStringCreate("CURRENT FILE NAME",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNx,10); n++;
  XtSetArg(wargs[n],XmNy,15); n++;
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNtraversalOn,False); n++;
  heading1 = XtCreateManagedWidget("heading1", xmLabelWidgetClass,
                            board, wargs, n);
  XmStringFree(str);
  /* creates a label widget */
  n = 0;
  str = XmStringCreate("SAVE AS ",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNx,10); n++;
  XtSetArg(wargs[n],XmNy,55); n++;
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNtraversalOn,False); n++;
  heading2 = XtCreateManagedWidget("heading2", xmLabelWidgetClass,
                            board, wargs, n);
  XmStringFree(str);
  /* callback data */
  info_to_send = (EDIT_LABEL_STRUCT *)malloc(sizeof(EDIT_LABEL_STRUCT));
  if(info_to_send == NULL) {
    printf("Out of memory\n");
    exit(0);
  }
  info_to_send->label_widget1 = label1;
```

```
  info_to_send->label_widget2 = label2;
  /* creates a button widget */
  n = 0;
  str = XmStringCreate("DONE",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNx,10); n++;
  XtSetArg(wargs[n],XmNy,90); n++;
  XtSetArg(wargs[n],XmNuserData,info_to_send); n++;
  XtSetArg(wargs[n],XmNshadowThickness,4); n++;
  done_button = XtCreateManagedWidget("done", xmPushButtonWidgetClass,
                                      board, wargs, n);
  XmStringFree(str);
  /* adds a callback function */
  XtAddCallback(done_button, XmNactivateCallback, DoneSaveAs,
                data);
  /* creates a button widget */
  n = 0;
  str = XmStringCreate("CANCEL",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNx,50); n++;
  XtSetArg(wargs[n],XmNy,90); n++;
  XtSetArg(wargs[n],XmNshadowThickness,4); n++;
  cancel_button = XtCreateManagedWidget("cancel", xmPushButtonWidgetClass,
                                      board, wargs, n);
  XmStringFree(str);
  /* adds a callback function */
  XtAddCallback(cancel_button, XmNactivateCallback, CancelSaveAs,NULL);
  XtManageChild(board);
  XmProcessTraversal(label2,XmTRAVERSE_CURRENT); /* focus to first entry */
}


/*
CancelSaveAs(), as a callback function pops down the SAVE AS FILE
dialog box.
*/

void CancelSaveAs(w, data, call_data)
Widget                w;
XtPointer             data;
XmAnyCallbackStruct   *call_data;
{
  strcpy(action_selected,"NOTHING");
  RewriteAction(); /* displays the action selected */
  XtUnmanageChild(XtParent(w)); /* pops down the dialog */
}


/*
DoneSaveAs() as a callback function saves the file when DONE
button is clicked after entering the filename.
*/

void DoneSaveAs(w, data, call_data)
Widget                w;
graphics_data         *data;
XmAnyCallbackStruct   *call_data;
{
  FILE   *chk_file;
  EDIT_LABEL_STRUCT  *info;
  Arg wargs[MAX_ARGS];
  char  *old_label,
        *new_label,
        mesg[MSG_LENGTH];

  XtSetArg(wargs[0], XmNuserData, &info);
  XtGetValues(w, wargs, 1);
  /* gets the labels */
  old_label = XmTextGetString(info->label_widget1);
  new_label = XmTextGetString(info->label_widget2);
  TrimString(new_label);
  if(strlen(new_label) == 0) { /* a null string is entered for the filename */
    strcpy(mesg,GetFileMsgStr("NULL_FILE"));
    PutErrorDialog(w,mesg);
    return;
  }
  if((chk_file = fopen(new_label,"r"))) { /* file already exists */
    strcpy(mesg,GetFileMsgStr("FILE_EXISTS"));
    PutErrorDialog(w,mesg);
    XtFree(new_label);
    XtFree(old_label);
```

```
    fclose(chk_file);
    return;
  }
  strcpy(GLOBAL_FILE_NAME,new_label); /* updates the filename */
  RewriteFileName(); /* displays the filename */
  strcpy(action_selected,"Saving...");
  RewriteAction();
  SaveFile(data); /* saves the file */
  XtFree(new_label);
  XtFree(old_label);
  strcpy(action_selected,"NOTHING"); /* updates the action selected */
  RewriteAction();
  XtUnmanageChild(XtParent(w)); /* pops down the dialog box */
}


/*
File name:   cursor.h
Description: It contains the header files required for the file cursor.c.
             It also declares the functions defined in the file cursor.c
*/

#include <X11/cursorfont.h>
#include "data_str.h"
#include "global.h"

/*
The file cursor.c consists of following functions.
For details, refer explanation before the declaration of each
function.
*/
Cursor  CreateCursor();
void InitCursors();
void ResetCursor();
void SetPirateCursor();
void SetHandCursor();
void SetBusyCursor();


/*
File name:   cursor.c .
Description: It contains the functions required to create different
             shapes of the cursor.
*/

#include "cursor.h"

/*
CreateCursor() creates a cursor of a desired shape.
*/

Cursor CreateCursor(w, shape)
Widget  w;
unsigned int shape;
{
  Cursor cursor;
  Display *dpy = XtDisplay(w);
  cursor =  XCreateFontCursor(dpy,shape);  /* creates a cursor */
  return(cursor);  /* returns the created cursor */
}


/*
InitCursors() initializes cursors of different shapes.
*/

void InitCursors()
{
  busy_cursor   = CreateCursor(toplevel,XC_watch);
  pirate_cursor = CreateCursor(toplevel,XC_pirate);
  hand_cursor   = CreateCursor(toplevel,XC_hand1);
}


/*
ResetCursor() undefines the cursor for a widget. When current
cursor is undefined, default shape of the cursor is restored.
*/

void ResetCursor(w)
```

```
Widget  w;
{
  XUndefineCursor(XtDisplay(w),XtWindow(w));  /* undefines the cursor for
                                                 a widget */
}


/*
SetPirateCursor() defines the XC_pirate shape for the widget. This
is used when DeleteObject option is selected from the pull down
menu Edit.
*/

void SetPirateCursor(w)
Widget      w;
{
  extern Cursor pirate_cursor;
  Display *dpy = XtDisplay(w);
  XDefineCursor(dpy,XtWindow(w),pirate_cursor);
}


/*
SetHandCursor() defines the XC_hand1 shape for the widget. This
is used for pointing and selecting an object, i.e., for EDIT_SPEC,
stepped firing, etc.
*/

void SetHandCursor(w)
Widget      w;
{
  extern Cursor hand_cursor;
  Display *dpy = XtDisplay(w);
  XDefineCursor(dpy,XtWindow(w),hand_cursor);
}


/*
SetBusyCursor() defines the XC_watch shape for the widget. This
is used when system is busy doing some processing, i.e., during
the process of bringing up a dialog box.
*/

void SetBusyCursor(w)
Widget      w;
{
  extern Cursor busy_cursor;
  Display *dpy = XtDisplay(w);
  XDefineCursor(dpy,XtWindow(w),busy_cursor);
}



/*
File name: help.h
Description: It contains the header files required for the file help.c.
             It also declares the functions defined in the file help.c
*/

#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/Label.h>
#include <Xm/MessageB.h>
#include "data_str.h"
#include "global.h"

/*
The file help.c consists of following functions.
For details, refer explanation before the declaration of each
function.
*/
void PlaceHelp();
void TransHelp();
void LinkHelp();
void PutTokenHelp();
void EditSpecHelp();
void ShowSpecHelp();
void FireHelp();
void UndoHelp();
void DeleteHelp();
void GenHelp();
```

```
static void DoneHelp();
XmString  StringToXmstring();


/*
File name:   help.c
Description: Its contains the functions which displays a help about an
             option.
*/

#include "help.h"

/*
PlaceHelp() as a callback function provides a help about the option
PLACE.
*/

void PlaceHelp(w, client_data, call_data)
Widget    w;
XtPointer client_data,
          call_data;
{
static char *p_help_str[] = {
  "To draw PLACE, select the PLACE button and then click on",
  "the drawing area where you want to draw PLACE.",
  "",
  "The click on the drawing area will be taken as the center",
  "of the circle representing PLACE.",
  "Whenever PLACE is drawn, a default label is given to it,",
  "i.e., p1, p2, etc.",
  "","};
  GenHelp(w, p_help_str);
}


/*
TransHelp(), as a callback function provides a help about the option
TRANSITION.
*/

void TransHelp(w, client_data, call_data)
Widget    w;
XtPointer client_data,
          call_data;
{
static char *t_help_str[] = {
  "To draw TRANSITION, select the TRANSITION button and then click on",
  "the drawing area where you want to draw TRANSITION. Move the mouse",
  "to get the desired size of the TRANSITION, click button 2 to end it.",
  "",
  "A click on the drawing area will be taken as one of the corners of",
  "the rectangle representing the TRANSITION.",
  "Whenever a TRANSITION is drawn, a default label is given to it,",
  "i.e., t1, t2, etc.",
  "If any other option is selected from the menu during the creation of",
  "a TRANSITOIN, the current selection will be lost and the incomplete",
  "TRANSITION will be erased.",
  "","};
  GenHelp(w, t_help_str);
}


/*
Linkhelp(), as a callback function provides a help about the option
LINK.
*/

void LinkHelp(w, client_data, call_data)
Widget    w;
XtPointer client_data,
          call_data;
{
static char *l_help_str[] = {
  "To draw a LINK, select the LINK button. A window will pop-up. Enter",
  "the SOURCE and the DESTINATION of the link. Press DONE and the window",
  "will pop-down. Now select the SOURCE by clicking inside it and move",
  "the mouse to hit DESTINATION.",
  "As soon as the DESTINATION is hit, a link will be established.",
  "",
  "The correct selection of SOURCE will be indicated by a bell sound.",
```

```
  "To change the path of a link, press button 1 and change the path.",
  "Pressing button 2 before reaching DESTINATION will cancel the link",
  "drawing process. A partially-made link is not undoable.",
  "If any other option is selected from menu during the process of",
  "creating a link, the current selection will be lost and the incomplete",
  "LINK will be erased.",
  "","");
  GenHelp(w, l_help_str);
}


/*
PutTokenhelp(), as a callback function provides a help about the option
PUT_TOKENS.
*/

void PutTokenHelp(w, client_data, call_data)
Widget    w;
XtPointer client_data,
          call_data;
{
static char *pt_help_str[] = {
  "To put TOKENS in a PLACE, select the PUT_TOKEN button . A window",
  "will pop-up. Enter the label of the PLACE where you want to put",
  "the token(s). Pressing the 'increment' or 'decrement' button changes",
  "the number of tokens accordingly.",
  "",
  "The maximum number of tokens that can be displayed at any PLACE is 99,",
  "w will be displayed for more than 99 tokens.",
  "","");
  GenHelp(w, pt_help_str);
}


/*
EditSpecHelp(), as a callback function provides a help about the option
EDIT_SPEC.
*/

void EditSpecHelp(w, client_data, call_data)
Widget    w;
XtPointer client_data,
          call_data;
{
static char *es_help_str[] = {
  "To edit the interpratation of a node, select the EDIT_SPEC button.",
  "Select the node drawn on the drawing area by pointing and clicking",
  "on it. A window will pop-up in which the interpratation of the ",
  "selected node can be edited.",
  "",
  "The maximum of 200 characters can be entered for each node interpratation.",
  "","");

  GenHelp(w, es_help_str);
}


/*
ShowSpecHelp(), as a callback function provides a help about the option
SHOW_SPEC.
*/

void ShowSpecHelp(w, client_data, call_data)
Widget    w;
XtPointer client_data,
          call_data;
{
static char *ss_help_str[] = {
  "To view the interpratation of a node, select the SHOW_SPEC button.",
  "Select the node drawn on the drawing area by pointing and clicking",
  "on it. A window will pop-up where the interpratation of the selected",
  "node can be viewed.",
  "","");

  GenHelp(w, ss_help_str);
}


/*
FireHelp(), as a callback function provides a help about the option
Stepped Firing.
```

```
*/

void FireHelp(w, client_data, call_data)
Widget    w;
XtPointer client_data,
          call_data;
{
static char *f_help_str[] = {
  "Pull-down menu Firing contains options Stepped Firing",
  "and History.",
  "To fire a transition, select Stepped Firing and click",
  "on the transition to be fired.",
  "Select History to display list of the last 20 fired",
  "transitions.",
  "","");

  GenHelp(w, f_help_str);
}


/*
UndoHelp(), as a callback function provides a help about the option
UNDO.
*/

void UndoHelp(w, client_data, call_data)
Widget    w;
XtPointer client_data,
          call_data;
{
static char *u_help_str[] = {
  "To UNDO the last draw action on the drawing area, select UNDO from",
  "the pull-down menu Edit or click button 3 of the mouse while inside",
  "the drawing area.",
  "Any partially-drawn LINK or TRANSITION is not undoable, rather it ",
  "would UNDO the last action before this incomplete action.",
  "","");

  GenHelp(w, u_help_str);
}


/*
DeleteHelp(), as a callback function provides a help about the option
DELETE_OBJECT.
*/

void DeleteHelp(w, client_data, call_data)
Widget    w;
XtPointer client_data,
          call_data;
{
static char *d_help_str[] = {
  "To delete any object, first select DELETE_OBJECT from the pull-down",
  "menu Edit and then click on the object to be deleted.",
  "Whenever DELETE_OBJECT is selected, the cursor changes the pirate sign.",
  "","");

  GenHelp(w, d_help_str);
}


/*
GenHelp() generates a pop up help window. The help string is passed
as an argument from the calling function.
*/

void GenHelp(w,str)
Widget     w;
char       *str[];
{
  int        i, n;
  Widget     dialog, info;
  XmString   xmstr,
             label_xmstr;
  Arg        wargs[MAX_ARGS];

  n = 0;
  label_xmstr = XmStringCreateLtoR("HELP WINDOW",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n], XmNdialogTitle, label_xmstr); n++;
  XtSetArg(wargs[n], XmNautoUnmanage, FALSE); n++;
```

```
  XtSetArg(wargs[n], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL); n++;
  XtSetArg(wargs[n], XmNdefaultPosition, FALSE); n++;
  XtSetArg(wargs[n], XmNx, 350); n++;
  XtSetArg(wargs[n], XmNy, 300); n++;
  dialog = XmCreateMessageDialog(w, "Help", wargs, n);
  XmStringFree(label_xmstr);
  XtUnmanageChild(XmMessageBoxGetChild(dialog, XmDIALOG_CANCEL_BUTTON));
  info = XmMessageBoxGetChild(dialog, XmDIALOG_MESSAGE_LABEL);
  n = 0;
  XtSetArg(wargs[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
  XtSetValues(info, wargs, n);
  XtAddCallback(dialog, XmNokCallback, DoneHelp, NULL);
  /* count the text up to the first NULL string */
  for(i = 0; str[i][0] != '\0'; i++)
    ;
  /* converts the string array to an XmString array and sets it as
     the label text */
  xmstr = StringToXmstring(str, i);
  n = 0;
  XtSetArg(wargs[n], XmNmessageString, xmstr); n++;
  XtSetValues(dialog, wargs, n);

  /*
  If the next entry in the help string array is also NULL,
  then this is the last message. Unmanage the help button.
   */
  if(str[++i][0] == '\0') {
    XtUnmanageChild(XmMessageBoxGetChild(dialog, XmDIALOG_HELP_BUTTON));
  }
  /* adds a help callback function */
  else {
    XtAddCallback(dialog, XmNhelpCallback, GenHelp, &str[i]);
  }
  /* displays the dialog */
  XtManageChild(dialog);
}


/*
StringToXmString() converts a character string to a
compound string (XmString) with separator components between
each line.
*/

XmString  StringToXmstring(str, n)
char     *str[];
int      n; /* represents length of the character array */
{
  XmString xmstr;
  int      i;

  /* If the array is empty just return an empty string */
  if(n <= 0)
    return(XmStringCreate("", XmSTRING_DEFAULT_CHARSET));

  xmstr = (XmString) NULL;
  for(i = 0; i < n; i++) {
    if(i > 0)
      xmstr = XmStringConcat(xmstr, XmStringSeparatorCreate());
    xmstr = XmStringConcat(xmstr, XmStringCreate(str[i],
                           XmSTRING_DEFAULT_CHARSET));
  }
  return(xmstr);
}


/*
DoneHelp() as a callback function unmanages a widget which is
passed as an argument.
*/
static void DoneHelp(w, client_data, call_data)
Widget          w;
caddr_t         client_data;
XmAnyCallbackStruct  *call_data;
{
  XtUnmanageChild(w);
}
```

```
/*
File name:   utility.h
Description: It contains the header files required for the file utility.c.
             It also declares the external functions and the functions
             defined in the file utility.c
*/

#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <math.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/Label.h>
#include <Xm/MessageB.h>
#include "data_str.h"
#include "global.h"

/*
These externally defined functions are called in the functions
defined in the file utility.c.
*/
extern DrawTransition();       /* defined in draw_object.s */
extern DrawPlace();            /* defined in draw_object.s */
extern DrawToken();            /* defined in draw_object.s */
extern MarkEnabledTrans();     /* defined in fire.c */
extern RemoveIncompleteLink(); /* defined in user_interface.c */

/*
The file utility.c consists of following functions.
For details, refer explanation before the declaration of each
function.
*/
void RewriteAction();
void RewriteFileName();
void RedrawDrawingBoard();
void ResetCurrentAction();
void TrimString();
void PutErrorDialog();
void AllTrim();
void DecodeFile();
void EncodeFile();


/*
File name:   utility.c
Description: It contains the functions which are used by different
             files of the application.
*/

#include "utility.h"


/*
RewriteAction() displays the selected action at the bottom bar.
*/

void RewriteAction()
{
  int pos;
  char head[20];
  GC  gc;
  Display *dpy = XtDisplay(action_display);
  Window   win = XtWindow(action_display);

  strcpy(head,"ACTION SELECTED:");
  pos = 5;
  gc  = XtGetGC(action_display, NULL, NULL);
  XClearWindow(dpy,win);
  XDrawString(dpy, win, gc, pos, 15, head, strlen(head));
  XDrawString(dpy, win, gc, pos+150, 15, action_selected,
              strlen(action_selected));
}


/*
RewriteFileName() displays the filename at the top bar.
*/

void RewriteFileName()
{
```

```
  int pos;
  GC  gc;
  Display *dpy = XtDisplay(filename_display);
  Window   win = XtWindow(filename_display);

  pos = 355 - 5 * strlen(GLOBAL_FILE_NAME);
  gc  = XtGetGC(filename_display, NULL, NULL);
  XClearWindow(dpy,win);
  XDrawString(dpy, win, gc, pos, 20, GLOBAL_FILE_NAME,
              strlen(GLOBAL_FILE_NAME));
}


/*
RedrawDrawingBoard() draws the object on the drawing area.
*/

void RedrawDrawingBoard(data)
graphics_data  *data;
{
  Display     *dpy = XtDisplay(drawing);
  Window       win = XtWindow(drawing);
  int  i;

  MarkEnabledTrans(data);
  XClearWindow(dpy,win);
  if(data->next_pos > 0) {
    for(i = 0; i < data->next_pos; i++) {
      if(strcmp(data->buffer[i].object,"TRANSITION") == 0) {
        strcpy(data->label, data->buffer[i].label);
        DrawTransition(drawing, data->gc,
                       data->buffer[i].x1,
                       data->buffer[i].y1,
                       data->buffer[i].x2,
                       data->buffer[i].y2,
                       data, "PUT_LABEL");
      }
      else if(strcmp(data->buffer[i].object,"PLACE") == 0) {
        strcpy(data->label, data->buffer[i].label);
        DrawPlace(drawing, data->buffer[i].gc,
                  data->buffer[i].x1,
                  data->buffer[i].y1,
                  data->buffer[i].x2,
                  data->buffer[i].y2,
                  data);
      }
      else if(strcmp(data->buffer[i].object,"PUT_TOKEN") == 0) {
        DrawToken(data->buffer[i].x1,
                  data->buffer[i].y1,
                  data);
      }
      else {
        (*(data->buffer[i].func)) (drawing, data->gc,
                                   data->buffer[i].x1,
                                   data->buffer[i].y1,
                                   data->buffer[i].x2,
                                   data->buffer[i].y2);
      }
    }
  }
}


/*
ResetCurrentAction() resets the current action, i.e., any incomplete
action is taken care of and nothing is assigned.
*/

void ResetCurrentAction(data)
graphics_data  *data;
{
  data->current_func = NULL;
  if(strcmp(data->object,"START_LINK") == 0) {
    strcpy(data->object,"");
    RemoveIncompleteLink(data);
    RedrawDrawingBoard(data);
  }
  else if(strcmp(data->object,"START_TRANS") == 0) {
    strcpy(data->object,"");
    RedrawDrawingBoard(data);
  }
```

```
  else if(strcmp(data->object,"START_PLACE") == 0) {
    strcpy(data->object,"");
    RedrawDrawingBoard(data);
  }
}


/*
TrimString() accepts a character string and removes any blanks
from it.
*/

void TrimString(str)
char  *str;
{
  int  i,j;
  char temp_str[300];

  strcpy(temp_str,str);
  str[0] = '\0';
  j = 0;
  for(i = 0; i < strlen(temp_str); i++) {
    if(temp_str[i] != ' ')
      str[j++] = temp_str[i];
  }
  str[j] = '\0';
}


/*
PutErrorDialog() pops up a warning dialog box. The warning message
is passed as an argument by the calling function.
*/

void PutErrorDialog(w,reason)
Widget    w;
char      *reason;
{
  static Widget   warning = NULL;
  Arg       wargs[20];
  int       n;
  XmString str1, str2, str3;

  if(!warning) {
  /* creates a warning message box */
  n = 0;
  str1 = XmStringCreateLtoR(" WARNING ",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNdialogTitle,str1); n++;
  str2 = XmStringCreateLtoR(reason,XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNmessageString,str2); n++;
  str3 = XmStringCreateLtoR("CLICK HERE",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNokLabelString,str3); n++;
  XtSetArg(wargs[n],XmNmessageAlignment,XmALIGNMENT_BEGINNING);
  XtSetArg(wargs[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL); n++;
  XtSetArg(wargs[n],XmNnoResize,True); n++;
  XtSetArg(wargs[n],XmNdefaultPosition,False); n++;
  XtSetArg(wargs[n],XmNx,350); n++;
  XtSetArg(wargs[n],XmNy,300); n++;
  warning = XmCreateErrorDialog(drawing, "warning",wargs,n);
  XmStringFree(str1);
  XmStringFree(str2);
  XtUnmanageChild(XmMessageBoxGetChild(warning, XmDIALOG_CANCEL_BUTTON));
  XtUnmanageChild(XmMessageBoxGetChild(warning, XmDIALOG_HELP_BUTTON));
  XtManageChild(warning);
  }
  else { /* warning message box already exists */
    n = 0;
    str2 = XmStringCreateLtoR(reason,XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNmessageString,str2); n++;
    XtSetValues(warning,wargs,n);
    XtManageChild(warning);
    XmStringFree(str2);
  }
}


/*
AllTrim() accepts a character string and removes any non printable
characters from both ends.
*/
```

```
void AllTrim(str)
char  *str;
{
  int  i, j;

  char  temp_buff[200];

  for(i = strlen(str)-1; i > 0; i--) {
    if((isspace(str[i]) == 0)) {
      str[i+1] = '\0';
      break;
    }
  }
  for(i = 0; i < strlen(str); i++) {
    if((isspace(str[i]) == 0)) {
      break;
    }
  }
  j = 0;
  for(i = i; i < strlen(str); i++) {
    temp_buff[j++] = str[i];
  }
  temp_buff[j] = '\0';
  strcpy(str,"");
  strcpy(str,temp_buff);
}


/*
DecodeFile() accepts two file pointers and writes second file
into first file in decoded form.
*/

void DecodeFile(fp1, fp2)
FILE    *fp1,
        *fp2;
{
  char ch;
  while((!feof(fp2)) && ((ch=fgetc(fp2)) != EOF)) {
    if(ch != '\n')
      fprintf(fp1, "%c",ch+10);
    else
      fprintf(fp1, "%c",ch);
  }
}


/*
EncodeFile() accepts two file pointers and writes second file
into first file in encoded form.
*/

void EncodeFile(fp1, fp2)
FILE    *fp1,
        *fp2;
{

  char ch;
  while((!feof(fp2)) && ((ch=fgetc(fp2)) != EOF)) {
    if(ch != '\n')
      fprintf(fp1, "%c",ch-10);
    else
      fprintf(fp1, "%c",ch);
  }
}


/*
File name:   undo.h
Description: It contains the header files required for the file undo.c.
             It also declares the external functions and the functions
             defined in the file undo.c
*/

#include "data_str.h"
#include "global.h"

/*
These externally defined functions are called in the functions
defined in the file undo.c.
*/
```

```
extern void PutErrorDialog();        /* defined in utility.c */
extern void ResetCurrentAction();    /* defined in utility.c */
extern void RewriteAction();         /* defined in utility.c */
extern void RedrawDrawingBoard();    /* defined in utility.c */
extern char *GetUndoMsgStr();        /* defined in warn_msg.c */
extern void RemoveIncompleteLink(); /* defined in user_interface.c */

/*
The file undo.c consists of following functions.
For details, refer explanation before the declaration of each
function.
*/
void  UndoLastDraw();
void  PutInDataBuf();
void  DelFromDataBuf();
void  RemoveLinkInfo();
void  UndoLinkInfo();
void  RemovePlaceInfo();
void  UndeletePlaceInfo();
void  RemoveTransInfo();
void  UndeleteTransInfo();


/*
File name:   undo.c
Description: It contains the functions required to implement the option
             Undo.
*/

#include "undo.h"


/*
UndoLastDraw() undoes the last drawing action. It checks the flag UNDO_ACTION
and depending upon its value, it calls the appropriate function to undo
the last action.
*/

void UndoLastDraw(w, data, call_data)
Widget  w;
graphics_data  *data;
XtPointer      call_data;
{
  char  local_str1[30];
  char  local_str2[30];
  char  mesg[MSG_LENGTH];

  if((strcmp(data->object,"START_LINK") == 0) ||
   (strcmp(data->object,"START_TRANS") == 0)  ||
   (strcmp(data->object,"START_PLACE") == 0))
    ResetCurrentAction(data);

  if(UNDO_ACTION == NO) { /* there is nothing to undo */
    strcpy(mesg,GetUndoMsgStr("NO_UNDO"));
    PutErrorDialog(drawing,mesg);
    return;
  }
  switch(UNDO_ACTION) {
    case UNDO_LABEL: /* undo action to change the label */
      strcpy(local_str1, LAST1);
      strcpy(local_str2, LAST2);
      if(!(CheckAndEditPlaceList(local_str2,local_str1,data))) {
        CheckAndEditTransList(local_str2,local_str1,data);
      }
      break;
    case UNDO_DEL_PLACE: /* undo action to delete a place */
      RemovePlaceInfo();
      data->next_pos -= 2;
      UNDO_ACTION = UNDO_UNDEL_PLACE;
      break;
    case UNDO_DEL_TRANS: /* undo action to delete a transition */
      RemoveTransInfo();
      data->next_pos -= 1;
      UNDO_ACTION = UNDO_UNDEL_TRANS;
      break;
    case UNDO_DEL_LINK: /* undo action to delete a link */
      RemoveLinkInfo();
      DelFromDataBuf(data);
      UNDO_ACTION = UNDO_UNDEL_LINK;
      break;
    case UNDO_UNDEL_PLACE: /* undo action to undelete a place */
```

```
      UndeletePlaceInfo(data);
      data->next_pos += 2;
      UNDO_ACTION = UNDO_DEL_PLACE;
      break;
    case UNDO_UNDEL_TRANS: /* undo action to undelete a transition */
      UndeleteTransInfo(data);
      data->next_pos += 1;
      UNDO_ACTION = UNDO_DEL_TRANS;
      break;
    case UNDO_UNDEL_LINK: /* undo action to undelete a link */
      UndoLinkInfo();
      PutInDataBuf(data);
      UNDO_ACTION = UNDO_DEL_LINK;
      break;
    default:
      return;
  }
  RedrawDrawingBoard(data); /* draws the drawing area */
}


/*
PutInDataBuf() puts the previously deleted link information into
the data buffer and frees the memory.
*/

void  PutInDataBuf(data)
graphics_data  *data;
{
  int i;
  DEL_ITEM  *temp_item,
            *free_item;

  temp_item = DEL_LIST;
  while(temp_item != NULL) {
    strcpy(data->buffer[data->next_pos].object,temp_item->buffer.object);
    strcpy(data->buffer[data->next_pos].label,temp_item->buffer.label);
    data->buffer[data->next_pos].x1   = temp_item->buffer.x1;
    data->buffer[data->next_pos].x2   = temp_item->buffer.x2;
    data->buffer[data->next_pos].y1   = temp_item->buffer.y1;
    data->buffer[data->next_pos].y2   = temp_item->buffer.y2;
    data->buffer[data->next_pos].func = temp_item->buffer.func;
    data->buffer[data->next_pos].gc   = temp_item->buffer.gc;
    free_item = temp_item;
    temp_item = temp_item->next;
    data->next_pos++;
    free(free_item);
  }
  DEL_LIST = NULL;
}


/*
DelFromDataBuf() removes a link from the data buffer and puts in the
linked list of the deleted link for the undo action.
*/

void  DelFromDataBuf(data)
graphics_data  *data;
{
  int i, del_pos, no_of_sublinks;
  DEL_ITEM  *temp_item,
            *free_item;

  temp_item = DEL_LIST;
  while(temp_item != NULL) { /* removes any previously deleted link */
    free_item = temp_item;
    temp_item = temp_item->next;
    free(free_item);
  }
  DEL_LIST = NULL;
  /* searches for the link to be deleted */
  for(i = 0; i < data->next_pos; i++) {
    if(strstr(data->buffer[i].label,T_LIST->del_link)) {
      del_pos = i;
      break;
    }
  }
  no_of_sublinks = 0;
  /* puts parts of a link into a list */
  for(i = del_pos; i < data->next_pos; i++) {
```

```
    if(!(strstr(data->buffer[i].label,T_LIST->del_link))) {
      break;
    }
    else { /* creates an item for the list */
      if((temp_item = (DEL_ITEM *)malloc(sizeof(DEL_ITEM))) == NULL) {
        printf("Out of memory\n");
        exit(0);
      }
      strcpy(temp_item->buffer.object,data->buffer[i].object);
      strcpy(temp_item->buffer.label,data->buffer[i].label);
      temp_item->buffer.x1 = data->buffer[i].x1;
      temp_item->buffer.x2 = data->buffer[i].x2;
      temp_item->buffer.y1 = data->buffer[i].y1;
      temp_item->buffer.y2 = data->buffer[i].y2;
      temp_item->buffer.func = data->buffer[i].func;
      temp_item->buffer.gc = data->buffer[i].gc;
      temp_item->next = NULL;

      if(DEL_LIST == NULL) /* puts in the list */
        DEL_LIST = temp_item;
      else {
        temp_item->next = DEL_LIST;
        DEL_LIST = temp_item;
      }
      no_of_sublinks++; /* counts the number of sub-links */
    }
  }
  /* removes the link from the data buffer */
  for(i = del_pos; i < data->next_pos-no_of_sublinks; i++) {
    strcpy(data->buffer[i].label,data->buffer[i+no_of_sublinks].label);
    strcpy(data->buffer[i].object,data->buffer[i+no_of_sublinks].object);
    data->buffer[i].x1    = data->buffer[i+no_of_sublinks].x1;
    data->buffer[i].y1    = data->buffer[i+no_of_sublinks].y1;
    data->buffer[i].x2    = data->buffer[i+no_of_sublinks].x2;
    data->buffer[i].y2    = data->buffer[i+no_of_sublinks].y2;
    data->buffer[i].gc    = data->buffer[i+no_of_sublinks].gc;
    data->buffer[i].func = data->buffer[i+no_of_sublinks].func;
  }
  data->next_pos -= no_of_sublinks; /* updates the next position */
}


/*
RemoveLinkInfo() removes the information of a link from the linked
list of the places and the transitions.
*/

void RemoveLinkInfo()
{
  PLACE      *p_temp;
  TRANSITION *t_temp;
  int  i, delete_pos;

  /* finds and removes from the P_LIST */
  p_temp = P_LIST->HEAD;
  while((strcmp(p_temp->label,P_LIST->place_selected) != 0) && (p_temp != NULL))
    p_temp = p_temp->next;
  if(p_temp == NULL) {
    printf("ERROR: place for the deleted link not found\n");
    exit(0);
  }
  delete_pos = -1;
  if(P_LIST->pre_post == PRESET) {
    for(i = 0; i < p_temp->cur_preset_no; i++) {
      if(strcmp(p_temp->preset[i],T_LIST->trans_selected) == 0) {
        delete_pos = i;
        i = p_temp->cur_preset_no;
      }
    }
    for(i = delete_pos; i < p_temp->cur_preset_no; i++)
      strcpy(p_temp->preset[i],p_temp->preset[i+1]);
    p_temp->cur_preset_no -= 1;
  }
  else if(P_LIST->pre_post == POSTSET) {
    delete_pos = -1;
    for(i = 0; i < p_temp->cur_postset_no; i++) {
      if(strcmp(p_temp->postset[i],T_LIST->trans_selected) == 0) {
        delete_pos = i;
        i = p_temp->cur_postset_no;
      }
    }
```

```
    for(i = delete_pos; i < p_temp->cur_postset_no; i++)
      strcpy(p_temp->postset[i],p_temp->postset[i+1]);
    p_temp->cur_postset_no -= 1;
  }
  else {
    printf("ERROR: pre post undefined\n");
    exit(0);
  }

  /* finds and removes from the T_LIST */
  t_temp = T_LIST->HEAD;
  while((strcmp(t_temp->label,T_LIST->trans_selected) != 0) && (t_temp != NULL))
    t_temp = t_temp->next;

  if(t_temp == NULL) {
    printf("ERROR: transition for the deleted link not found\n");
    exit(0);
  }
  delete_pos = -1;
  if(T_LIST->pre_post == PRESET) {
    for(i = 0; i < t_temp->cur_preset_no; i++) {
      if(strcmp(t_temp->preset[i],P_LIST->place_selected) == 0) {
        delete_pos = i;
        i = t_temp->cur_preset_no;
      }
    }
    for(i = delete_pos; i < t_temp->cur_preset_no; i++)
      strcpy(t_temp->preset[i],t_temp->preset[i+1]);
    t_temp->cur_preset_no -= 1;
  }
  else if(T_LIST->pre_post == POSTSET) {
    delete_pos = -1;
    for(i = 0; i < t_temp->cur_postset_no; i++) {
      if(strcmp(t_temp->postset[i],P_LIST->place_selected) == 0) {
        delete_pos = i;
        i = t_temp->cur_postset_no;
      }
    }
    for(i = delete_pos; i < t_temp->cur_postset_no; i++)
      strcpy(t_temp->postset[i],t_temp->postset[i+1]);
    t_temp->cur_postset_no -= 1;
  }
  else {
    printf("ERROR: pre post undefined\n");
    exit(0);
  }
}


/*
UndoLinkInfo() undeletes a link. It gets the information of the previously
deleted link and puts back into the P_LIST and the T_LIST.
*/

void UndoLinkInfo()
{
  PLACE      *p_temp;
  TRANSITION *t_temp;

  p_temp = P_LIST->HEAD;
  while((strcmp(p_temp->label,P_LIST->place_selected) != 0) && (p_temp != NULL))
    p_temp = p_temp->next;
  if(p_temp == NULL) {
    printf("ERROR: place for the deleted link not found\n");
    exit(0);
  }
  /* puts back the link information */
  if(P_LIST->pre_post == PRESET) {
    strcpy(p_temp->preset[p_temp->cur_preset_no],T_LIST->trans_selected);
    p_temp->cur_preset_no += 1;
  }
  else if(P_LIST->pre_post == POSTSET) {
    strcpy(p_temp->postset[p_temp->cur_postset_no],T_LIST->trans_selected);
    p_temp->cur_postset_no += 1;
  }
  else {
    printf("ERROR: preset and postset undefined\n");
    exit(0);
  }

  t_temp = T_LIST->HEAD;
```

```
  while((strcmp(t_temp->label,T_LIST->trans_selected) != 0) && (t_temp != NULL))
    t_temp = t_temp->next;
  if(t_temp == NULL) {
    printf("ERROR: transition for the deleted link not found\n");
    exit(0);
  }
  /* puts back the link information */
  if(T_LIST->pre_post == PRESET) {
    strcpy(t_temp->preset[t_temp->cur_preset_no],P_LIST->place_selected);
    t_temp->cur_preset_no += 1;
  }
  else if(T_LIST->pre_post == POSTSET) {
    strcpy(t_temp->postset[t_temp->cur_postset_no],P_LIST->place_selected);
    t_temp->cur_postset_no += 1;
  }
  else {
    printf("ERROR: preset and postset undefined\n");
    exit(0);
  }
}


/*
RemovePlaceInfo() removes the information of a place to be deleted
from the place list, i.e., P_LIST.
*/

void RemovePlaceInfo()
{
  PLACE      *p_temp;

  p_temp = P_LIST->HEAD;
  while((strcmp(p_temp->label,P_LIST->place_selected) != 0) &&
        (p_temp != NULL))
    p_temp = p_temp->next;
  if(p_temp == NULL) {
    printf("ERROR: place to be deleted not found\n");
    exit(0);
  }
  P_LIST->last_token = p_temp->no_of_tokens;
  if(P_LIST->HEAD == p_temp) { /* updates the list */
    P_LIST->HEAD = p_temp->next;
    if(P_LIST->HEAD != NULL)
      P_LIST->HEAD->prev = NULL;
  }
  else {
    p_temp->prev->next = p_temp->next;
    if(p_temp->next != NULL)
      p_temp->next->prev = p_temp->prev;
  }
  free(p_temp); /* frees the memory */
}


/*
UndeletePlaceInfo() puts back the information of a place to be undeleted,
into the P_LIST.
*/

void UndeletePlaceInfo(data)
graphics_data  *data;
{
  PLACE      *p_temp;

  if((p_temp = (PLACE *)malloc(sizeof(PLACE))) == NULL) {
    printf("Out Of Memory\n");
    exit(0);
  }
  p_temp->next = NULL;
  p_temp->prev = NULL;
  p_temp->cur_preset_no  = 0;
  p_temp->cur_postset_no = 0;
  p_temp->no_of_tokens = P_LIST->last_token;
  strcpy(p_temp->label,data->buffer[data->next_pos].label);
  strcpy(P_LIST->place_selected,p_temp->label);
  if(P_LIST->HEAD == NULL)
    P_LIST->HEAD = p_temp;
  else {
    p_temp->next = P_LIST->HEAD;
    P_LIST->HEAD->prev = p_temp;
    P_LIST->HEAD = p_temp;
```

```
  }
}


/*
RemoveTransInfo() removes the information of a transition to be deleted
from the transition list, i.e., T_LIST.
*/

void RemoveTransInfo()
{
  TRANSITION      *t_temp;

  t_temp = T_LIST->HEAD;
  while((strcmp(t_temp->label,T_LIST->trans_selected) != 0) &&
        (t_temp != NULL))
    t_temp = t_temp->next;

  if(t_temp == NULL) {
    printf("ERROR: place to be deleted not found\n");
    exit(0);
  }
  if(T_LIST->HEAD == t_temp) {
    T_LIST->HEAD = t_temp->next;
    if(T_LIST->HEAD != NULL)
      T_LIST->HEAD->prev = NULL;
  }
  else {
    t_temp->prev->next = t_temp->next;
    if(t_temp->next != NULL)
      t_temp->next->prev = t_temp->prev;
  }

  free(t_temp);
}


/*
UndeleteTransInfo() puts back the information of a transition to be
undeleted into the T_LIST.
*/

void UndeleteTransInfo(data)
graphics_data  *data;
{
  TRANSITION      *t_temp;

  if((t_temp = (TRANSITION *)malloc(sizeof(TRANSITION))) == NULL) {
    printf("Out Of Memory\n");
    exit(0);
  }
  t_temp->next = NULL;
  t_temp->prev = NULL;
  t_temp->cur_preset_no  = 0;
  t_temp->cur_postset_no = 0;
  t_temp->enabled        = NO;
  strcpy(t_temp->label,data->buffer[data->next_pos].label);
  strcpy(T_LIST->trans_selected,t_temp->label);
  if(T_LIST->HEAD == NULL)
    T_LIST->HEAD = t_temp;
  else {
    t_temp->next = T_LIST->HEAD;
    T_LIST->HEAD->prev = t_temp;
    T_LIST->HEAD = t_temp;
  }
}


/*
File name:   warn_msg.c
Description: It contains function which select a warning message.
*/


#include "data_str.h"
#include "global.h"


/*
GetLinkMsgStr() selects an apropriate message, related to the
option LINK, depending upon the value of the flag.
```

```
*/

char  *GetLinkMsgStr(flag)
char   *flag;
{
  char  mesg[MSG_LENGTH];

  /* Message for incorrect source (TRANSITION) selected for a link. */
  if(strcmp(flag,"TSRC_ERR_MSG") == 0) {
    strcpy(mesg,"REASON: INCORRECT SOURCE!\n  \n");
    strcat(mesg,"LINK DEFINITION SHOULD BE AS FOLLOWS:\n\nSOURCE     : ");
    strcat(mesg,T_LIST->trans_pre_selected);
    strcat(mesg,"\nDESTINATION: ");
    strcat(mesg,P_LIST->place_pre_selected);
    strcat(mesg,"\nINSTRUCTION: Try to touch the boundary of the source. ");
    strcat(mesg,
          "\n              Then drag the mouse to touch the boundary of ");
    strcat(mesg,
          "\n              the destination.");
    strcat(mesg,
          "\nNOTE        : A bell sound will indicate a correct touch.\n");
    return(mesg);
  }
  /* Message for incorrect source (PLACE) selected for a link. */
  else if(strcmp(flag,"PSRC_ERR_MSG") == 0) {
    strcpy(mesg,"REASON: INCORRECT SOURCE!\n  \n");
    strcat(mesg,"LINK DEFINITION SHOULD BE AS FOLLOWS:\n\nSOURCE     : ");
    strcat(mesg,P_LIST->place_pre_selected);
    strcat(mesg,"\nDESTINATION: ");
    strcat(mesg,T_LIST->trans_pre_selected);
    strcat(mesg,"\nINSTRUCTION: Try to touch the boundary of the source. ");
    strcat(mesg,
          "\n              Then drag the mouse to touch the boundary of ");
    strcat(mesg,
          "\n              the destination.");
    strcat(mesg,
          "\nNOTE        : A bell sound will indicate a correct touch.\n");
    return(mesg);
  }
  else if(strcmp(flag,"NULL_DST") == 0) {
    strcpy(mesg,"REASON: Invalid entry for the destination label.");
    return(mesg);
  }
  else if(strcmp(flag,"NULL_SRC") == 0) {
    strcpy(mesg,"REASON: Invalid entry for the source label.");
    return(mesg);
  }
  else if(strcmp(flag,"NO_SRC") == 0) {
    strcpy(mesg,"REASON: The source label does not exist.");
    return(mesg);
  }
  else if(strcmp(flag,"NO_DST") == 0) {
    strcpy(mesg,"REASON: The destination label does not exist.");
    return(mesg);
  }
  else if(strcmp(flag,"SAME_TYPE_NODE") == 0) {
    strcpy(mesg,"REASON: Source and destination labels are same type nodes.");
    return(mesg);
  }
}


/*
GetTokMsgStr() selects an apropriate message, related to the
option PUT_TOKEN, depending upon the value of the flag.
*/

char  *GetTokMsgStr(flag)
char   *flag;
{
  char  mesg[MSG_LENGTH];

  if(strcmp(flag,"NULL_LABEL") == 0) {
    strcpy(mesg,"REASON: Invalid entry for label.");
    return(mesg);
  }
  else if(strcmp(flag,"TRANS_LABEL") == 0) {
    strcpy(mesg,"REASON: Invalid entry for label.\n");
    strcat(mesg,"        Tokens can be put in PLACEs only.");
    return(mesg);
  }
```

```
  else if(strcmp(flag,"LB_NOT_FOUND") == 0) {
    strcpy(mesg,"REASON: This label does not exist.");
    return(mesg);
  }
  else if(strcmp(flag,"NO_NEG_TOK") == 0) {
    strcpy(mesg,"REASON: PLACEs cannot have negative tokens.");
    return(mesg);
  }
}


/*
GetEditLabelMsgStr() selects an apropriate message, related to the
option EditLabel, depending upon the value of the flag.
*/

void GetEditLabelMsgStr(mesg, flag)
char  *mesg;
char  *flag;
{

  if(strcmp(flag,"NULL_OLABEL") == 0) {
    strcpy(mesg,"REASON: Invalid entry for old_label.");
    return;
  }
  else if(strcmp(flag,"NULL_NLABEL") == 0) {
    strcpy(mesg,"REASON: Invalid entry for new_label.");
    return;
  }
  else if(strcmp(flag,"PLABEL_EXISTS") == 0) {
    strcpy(mesg,"REASON: The new label already exists for a PLACE.");
    return;
  }
  else if(strcmp(flag,"TLABEL_EXISTS") == 0) {
    strcpy(mesg,"REASON: The new label already exists for a TRANSITION.");
    return;
  }
  else if(strcmp(flag,"LB_NOT_FOUND") == 0) {
    strcpy(mesg,"REASON: The old label does not exist.");
    return;
  }
}


/*
GetDeleteMsgStr() selects an apropriate message, related to the
option DeleteObject, depending upon the value of the flag.
*/

char  *GetDeleteMsgStr(flag)
char   *flag;
{
  char  mesg[MSG_LENGTH];

  if(strcmp(flag,"DEL_LINKS") == 0) {
    strcpy(mesg,"REASON: You must first delete the links.");
    return(mesg);
  }
}


/*
GetFireMsgStr() selects an apropriate message, related to the
option SteppedFire, depending upon the value of the flag.
*/

char  *GetFireMsgStr(flag)
char   *flag;
{
  char  mesg[MSG_LENGTH];

  if(strcmp(flag,"DISABLE") == 0) {
    strcpy(mesg,"REASON: This transition is not enabled.");
    return(mesg);
  }
  else if(strcmp(flag,"NO_PRESET") == 0) {
    strcpy(mesg,"REASON: This transition is not enabled.\n");
    strcat(mesg,"        It has no input PLACE(s).");
    return(mesg);
  }
}
```

```
/*
GetUndoMsgStr() selects an apropriate message, related to the
option Undo, depending upon the value of the flag.
*/

char  *GetUndoMsgStr(flag)
char   *flag;
{
  char  mesg[MSG_LENGTH];

  if(strcmp(flag,"NO_UNDO") == 0) {
    strcpy(mesg,"REASON: There is nothing to Undo.");
    return(mesg);
  }
}


/*
GetFileMsgStr() selects an apropriate message, related to the
option Save, Save As, Open, and Quit.
*/

char  *GetFileMsgStr(flag)
char   *flag;
{
  char  mesg[MSG_LENGTH];

  if(strcmp(flag,"NULL_FILE") == 0) {
    strcpy(mesg,"REASON: Invalid file name.");
    return(mesg);
  }
  else if(strcmp(flag,"FILE_EXISTS") == 0) {
    strcpy(mesg,"REASON: File already exists.");
    return(mesg);
  }
  else if(strcmp(flag,"FILE_CANNOT_OPEN") == 0) {
    strcpy(mesg,"REASON: File cannot be opened.");
    return(mesg);
  }
  else if(strcmp(flag,"FILE_FORMAT_ERR") == 0) {
    strcpy(mesg,"REASON: File not in correct format.");
    return(mesg);
  }
}


/*
File name:   fire.h
Description: It contains the header files required for the file fire.c.
             It also declares the external functions and the functions
             defined in the file fire.c
*/

#include <stdio.h>
#include <string.h>
#include <Xm/Text.h>
#include <Xm/BulletinB.h>
#include <Xm/PushB.h>
#include <Xm/List.h>
#include <Xm/PanedW.h>
#include <Xm/RowColumn.h>
#include <Xm/Label.h>
#include <Xm/ScrolledW.h>
#include <Xm/ToggleBG.h>
#include "data_str.h"
#include "global.h"
#define  HISTORY_COUNT  20
extern char  trace_filename[FILE_NAME_LENGTH];
extern int   HISTORY_SWITCH;

/*
These externally defined functions are called in the functions
defined in the file fire.c.
*/
extern  void RedrawDrawingBoard(); /* defined in utility.c */
extern  void ResetCurrentAction(); /* defined in utility.c */
extern  void RewriteAction();      /* defined in utility.c */
extern  void PutErrorDialog();     /* defined in utility.c */
extern  void SetHandCursor();      /* defined in cursor.c */
extern  char *GetFireMsgStr();     /* defined in warn_msg.c */
```

```
extern  char *GetFileMsgStr();      /* defined in warn_msg.c */

/*
The file fire.c consists of following functions. These functions are
used to implement the Fire and related options present in pull
down menu Firing. For details, refer explanation before the declaration
of each function.
*/

void MarkEnabledTrans();
void SteppedFire();
void TransSelected();
void FireTrans();
int  IsTransEnable();
void UpdateH_LIST();
void FiringHistory();
void DoneExeSeq();
void DoneHistorySwitch();
void toggled();
void CancelHistorySwitch();
void UpdateTraceFile();



/*
File name:   fire.c
Description: It contains the function required to implement the options
             present in the pull down menu Firing.
*/

#include "fire.h"

/*
MarkEnabledTrans() checks the presence of the enabled transitions due
to any design change. If a transition is found to be enabled, it
is marked.
*/

void MarkEnabledTrans(data)
graphics_data  *data;
{
  TRANSITION  *t_temp;

  t_temp = T_LIST->HEAD;
  while(t_temp != NULL) {
    if(IsTransEnable(t_temp)) /* checks if the transition is enabled */
      t_temp->enabled = YES;
    else
      t_temp->enabled = NO;  /* if the transition is not enabled */
    t_temp = t_temp->next;
  }
}


/*
SteppedFire() changes the current action to STEPPED FIRING, it
registers the function TransSelected() and defines a hand-shaped
cursor for drawing area. After this, a user can select any transition
to fire.
*/

void SteppedFire(w, data, call_data)
Widget          w;
graphics_data  *data;
XtPointer       call_data;
{
  ResetCurrentAction(data);  /* resets any previous incomplete action */
  strcpy(data->object,"STEPPED_FIRE");
  data->current_func = TransSelected;  /* registers function TransSelected() */
  strcpy(action_selected,"STEPPED FIRING");  /* STEPPED FIRING as current
                                                action */
  RewriteAction();  /* displays the action selected */
  SetHandCursor(drawing);
}


/*
TransSelected() decides which transition is selected to fire. It
takes the current position where mouse is clicked and checks if
it selects any transition. If any enabled transition is selected,
it is fired.
```

```
*/

void TransSelected(data)
graphics_data  *data;
{
  TRANSITION  *t_temp;  /* used to traverse the T_LIST */
  int    x1, x2, y1, y2, xpos, ypos;  /* temporary variables to hold
                                         coordinates */
  int    i, pos = -1;
  char   mesg[500];  /* stores a warning message */

  xpos = data->x1;
  ypos = data->y1;
  for(i = 0; i < data->next_pos; i++) {
    x1 = data->buffer[i].x1;
    x2 = data->buffer[i].x2;
    y1 = data->buffer[i].y1;
    y2 = data->buffer[i].y2;
    if(((xpos >= x1) && (xpos <= x2) && (ypos >= y1) && (ypos <= y2))||
       ((xpos <= x1) && (xpos >= x2) && (ypos >= y1) && (ypos <= y2))||
       ((xpos >= x1) && (xpos <= x2) && (ypos <= y1) && (ypos >= y2))||
       ((xpos <= x1) && (xpos >= x2) && (ypos <= y1) && (ypos >= y2))){

      /* if the selected object is a transition */
      if(strcmp(data->buffer[i].object,"TRANSITION") == 0) {
        pos = i;
        i = data->next_pos;
      }
    }
  }
  /* if a transition is selected, i.e., pos != -1 */
  if (pos != -1) {
    t_temp = T_LIST->HEAD;  /* searches the required transition in the T_LIST */
    while((strcmp(t_temp->label,data->buffer[pos].label) != 0) &&
          (t_temp != NULL))
      t_temp = t_temp->next;
    if(t_temp != NULL) {  /* finds the  transition */
      if(t_temp->cur_preset_no == 0) {  /* if the transition has no preset */
        strcpy(mesg,GetFireMsgStr("NO_PRESET"));
        PutErrorDialog(drawing, mesg);  /* displays a warning message */
        return;
      }
      if(!IsTransEnable(t_temp)) {  /* if the transition is not enabled */
        strcpy(mesg,GetFireMsgStr("DISABLE"));
        PutErrorDialog(drawing, mesg);  /* displays a warning message */
        return;
      }
      FireTrans(t_temp);  /* if the transition is enabled, it fires */
      MarkEnabledTrans(data); /* updates the list of enabled transitions */
      RedrawDrawingBoard(data); /* draws the objects */
      UpdateH_LIST(t_temp->label); /* updates the history list, i.e., H_LIST */
      UpdateTraceFile(t_temp->label); /* updates the trace file */
    }
  }
}


/*
FireTrans() simulates the firing of the selected transition. It
updates the preset and the postset of the transition and updates the
number of tokens in the places involved in the process.
*/

void FireTrans(t_temp)
TRANSITION  *t_temp;
{
  PLACE  *p_temp;
  int    i;

  /*
  picks a place from the preset of the selected transition, searches in the
  P_LIST and decrements the number of tokens by one
  */
  for(i = 0; i < t_temp->cur_preset_no; i++) {
    p_temp = P_LIST->HEAD;
    while((strcmp(p_temp->label,t_temp->preset[i]) != 0) &&
          (p_temp != NULL))
      p_temp = p_temp->next;
    if(p_temp == NULL) {
      printf("ERROR: preset of transition not found for firing\n");
      exit(0);
```

```
    }
    p_temp->no_of_tokens-- ;
  }

  /*
  picks a place from the postset of the selected transition, searches in the
  P_LIST and increments the number of tokens by one
  */
  for(i = 0; i < t_temp->cur_postset_no; i++) {
    p_temp = P_LIST->HEAD;
    while((strcmp(p_temp->label,t_temp->postset[i]) != 0) &&
          (p_temp != NULL))
      p_temp = p_temp->next;
    if(p_temp == NULL) {
      printf("ERROR: postset of transition not found for firing\n");
      exit(0);
    }
    p_temp->no_of_tokens++ ;
  }
}


/*
IsTransEnable() decides if the selected transition is enabled.
*/

int IsTransEnable(t_temp)
TRANSITION  *t_temp;
{
  PLACE  *p_temp;
  int    i;

  if(t_temp->cur_preset_no == 0) return(0); /* not enabled */
  p_temp = P_LIST->HEAD;
  while(p_temp != NULL) {
    p_temp->freq_in_preset = 0;
    p_temp = p_temp->next;
  }
  for(i = 0; i < t_temp->cur_preset_no; i++) {
    p_temp = P_LIST->HEAD;
    while((strcmp(p_temp->label,t_temp->preset[i]) != 0) &&
          (p_temp != NULL))
      p_temp = p_temp->next;
    if(p_temp == NULL) { /* error in the application */
      printf("ERROR: preset of transition not found for firing\n");
      exit(0);
    }
    p_temp->freq_in_preset++;
    if(p_temp->no_of_tokens < p_temp->freq_in_preset)
      return(0); /* transition is not enabled */
  }
  return(1); /* transition is enabled */
}


/*
UpdateH_LIST() updates the entry in the history list.
*/

void UpdateH_LIST(trans_label)
char     *trans_label;
{
  HISTORY_ITEM    *h_temp;

  /* creates a new entry */
  if((h_temp = (HISTORY_ITEM *)malloc(sizeof(HISTORY_ITEM))) == NULL) {
    printf("Out Of Memory\n");
    exit(0);
  }
  h_temp->next = NULL;
  h_temp->prev = NULL;
  strcpy(h_temp->trans_label, trans_label);
  /* updates the list */
  if(H_LIST->HEAD == NULL) {
    H_LIST->HEAD = h_temp;
    H_LIST->TAIL = h_temp;
  }
  else {
    if (H_LIST->count >= HISTORY_COUNT) {
      H_LIST->TAIL = H_LIST->TAIL->prev;
      free(H_LIST->TAIL->next);
```

```
      H_LIST->TAIL->next = NULL;
    }
    h_temp->next = H_LIST->HEAD;
    H_LIST->HEAD->prev = h_temp;
    H_LIST->HEAD = h_temp;
  }
  H_LIST->count++;
}


/*
FiringHistory() displays a history of the firing.
*/

void FiringHistory(parent, client_data, call_data)
Widget     parent;
XtPointer  client_data;
XmToggleButtonCallbackStruct  *call_data;
{
  static Widget  board = NULL, radio_box, history_on,
                 history_off, label1, label2, text_w,
                 done_button, cancel_button;
  int            n;
  Arg            wargs[20];
  XmString       str;

  if(!call_data->set) {
    strcpy(trace_filename,"Untitled");
    return;
  }
  /* creates a user-interface for the SET HISTORY dialog box */
  if(!board) {
    n = 0;
    str = XmStringCreate("SET HISTORY",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNdefaultPosition,False); n++;
    XtSetArg(wargs[n],XmNdialogTitle,str); n++;
    XtSetArg(wargs[n],XmNallowOverlap,False); n++;
    XtSetArg(wargs[n],XmNx,300); n++;
    XtSetArg(wargs[n],XmNy,200); n++;
    XtSetArg(wargs[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL); n++;
    XtSetArg(wargs[n],XmNautoUnmanage,False); n++;
    board = XmCreateBulletinBoardDialog(drawing, "board", wargs, n);
    XmStringFree(str);
    /* creates a label widget */
    n = 0;
    XtSetArg(wargs[n],XmNx,10); n++;
    XtSetArg(wargs[n],XmNy,15); n++;
    str = XmStringCreate("Filename",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    label1 = XtCreateManagedWidget("label1", xmLabelWidgetClass, board,
                                   wargs, n);
    XmStringFree(str);
    /* creates a text widget */
    n = 0;
    XtSetArg(wargs[n],XmNx,70); n++;
    XtSetArg(wargs[n],XmNy,10); n++;
    XtSetArg(wargs[n],XmNvalue,trace_filename); n++;
    XtSetArg(wargs[n],XmNcursorPosition,0); n++;
    text_w = XtCreateManagedWidget("text_w", xmTextWidgetClass, board,
                                   wargs, n);
    /* creates a button */
    n = 0;
    XtSetArg(wargs[n],XmNx,10); n++;
    XtSetArg(wargs[n],XmNy,85); n++;
    str = XmStringCreate("DONE",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNrecomputeSize,False); n++;
    XtSetArg(wargs[n],XmNshadowThickness,4); n++;
    done_button = XtCreateManagedWidget("done", xmPushButtonWidgetClass,
                                        board, wargs, n);
    XmStringFree(str);
    /* adds a callback function */
    XtAddCallback(done_button, XmNactivateCallback, DoneHistorySwitch,
                  text_w);
    /* creates a button */
    n = 0;
    XtSetArg(wargs[n],XmNx,50); n++;
    XtSetArg(wargs[n],XmNy,85); n++;
    str = XmStringCreate("CANCEL",XmSTRING_DEFAULT_CHARSET);
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNrecomputeSize,False); n++;
```

```
    XtSetArg(wargs[n],XmNshadowThickness,4); n++;
    cancel_button = XtCreateManagedWidget("cancel", xmPushButtonWidgetClass,
                                          board, wargs, n);
    XmStringFree(str);
    /* adds a callback function */
    XtAddCallback(cancel_button, XmNactivateCallback, CancelHistorySwitch,
                  parent);
    XtManageChild(board);
    XmProcessTraversal(text_w,XmTRAVERSE_CURRENT); /* focus to text entry */
  }
  else { /* if the widget already exists */
    n = 0;
    XtSetArg(wargs[n],XmNcursorPosition,strlen(trace_filename)); n++;
    XtSetArg(wargs[n],XmNvalue,trace_filename); n++;
    XtSetValues(text_w,wargs,n);
    XtManageChild(board);
    XmProcessTraversal(text_w,XmTRAVERSE_CURRENT); /* focus to text entry */
  }
}


/*
CancelHistorySwitch(), as a callback function pops down the dialog box.
*/

void CancelHistorySwitch(w, client_data, call_data)
Widget    w;
Widget    client_data;
XtPointer call_data;
{
  Arg  wargs[5];

  XtSetArg(wargs[0],XmNset,False);
  XtSetValues(client_data, wargs,1);
  XtUnmanageChild(XtParent(w));
}


/*
DoneHistorySwitch(), as a callback function pops down the SET HISTORY
dialog box and saves the filename for the tarce of firing history.
*/

void DoneHistorySwitch(w, text_w, call_data)
Widget    w;
Widget    text_w;
XtPointer call_data;
{
  FILE *chk_file;
  char *buffer1,
       buffer2[FILE_NAME_LENGTH],
       mesg[MSG_LENGTH];

  /* gets the filename */
  buffer1 = XmTextGetString(text_w);
  strncpy(buffer2,buffer1,FILE_NAME_LENGTH);
  XtFree(buffer1);
  TrimString(buffer2);
  /* checks if filename is valid */
  if((strlen(buffer2) == 0) ||
     (strcmp(buffer2,"Untitled") == 0)) { /* invalid filename */
    strcpy(mesg,GetFileMsgStr("NULL_FILE"));
    PutErrorDialog(w,mesg);
    return;
  }
  if((chk_file = fopen(buffer2,"r")) != NULL) { /* file already exists */
    strcpy(mesg,GetFileMsgStr("FILE_EXISTS"));
    PutErrorDialog(w,mesg);
    fclose(chk_file);
    return;
  }
  strcpy(trace_filename,buffer2); /* updates the filename */
  XtUnmanageChild(XtParent(w));
}


/*
UpdateTraceFile() updates the tarce file of the firing history.
*/

void UpdateTraceFile(trans_label)
```

```
char   *trans_label;
{
  FILE    *file_ptr;
  PLACE   *p_temp;
  static  fire_count = 0;

  if(HISTORY_SWITCH == NO) return;
  /* writes in the history file */
  if(!(file_ptr = fopen(trace_filename,"a+"))) {
    printf("Error: Cannot open tarce file '%s'.\n",trace_filename);
    exit(0);
  }
  fire_count++;
  fprintf(file_ptr,"%d. ",fire_count);
  fprintf(file_ptr,"TRANSITION FIRED: ");
  fprintf(file_ptr,"%s   (Token assignment are as follows:)\n",trans_label);
  p_temp = P_LIST->HEAD;
  while(p_temp != NULL) {
    fprintf(file_ptr,"%s =  ",p_temp->label);
    fprintf(file_ptr,"%d\n",p_temp->no_of_tokens);
    p_temp = p_temp->next;
  }
  fprintf(file_ptr,"\n");
  fclose(file_ptr);
}


/*
ExeSequence() displays the execution sequence.
*/

void ExeSequence(parent, client_data, call_data)
Widget     parent;
XtPointer  client_data;
XtPointer  call_data;
{
  HISTORY_ITEM    *h_temp;
  Widget          board, scroll_w, q_button, rowcol, scroll_area;
  XmString        str;
  XmStringTable   t_list;
  Arg             wargs[10];
  int             n, count;

  h_temp = H_LIST->TAIL;
  /* creates a user-interface */
  str = XmStringCreate("EXECUTION SEQUENCE",XmSTRING_DEFAULT_CHARSET);
  n = 0;
  XtSetArg(wargs[n],XmNdialogTitle,str); n++;
  XtSetArg(wargs[n],XmNdefaultPosition,False); n++;
  XtSetArg(wargs[n],XmNallowOverlap,False); n++;
  XtSetArg(wargs[n],XmNx,200); n++;
  XtSetArg(wargs[n],XmNy,100); n++;
  XtSetArg(wargs[n],XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL); n++;
  XtSetArg(wargs[n],XmNautoUnmanage,False); n++;
  board = XmCreateBulletinBoardDialog(parent, "board", wargs, n);
  XmStringFree(str);
  /* creates a row column widget */
  rowcol = XtCreateManagedWidget("rowcol", xmRowColumnWidgetClass,
                          board, NULL, 0);
  /* creates a scroll window */
  n = 0;
  XtSetArg(wargs[n],XmNheight,200); n++;
  XtSetArg(wargs[n],XmNwidth,120); n++;
  XtSetArg(wargs[n],XmNscrollingPolicy,XmAUTOMATIC); n++;
  XtSetArg(wargs[n],XmNscrollBarDisplayPolicy,XmAS_NEEDED); n++;
  scroll_w = XtCreateManagedWidget("scroll_w", xmScrolledWindowWidgetClass,
                          rowcol, wargs, n);
  scroll_area = XtCreateWidget("scroll_area", xmRowColumnWidgetClass,
                          scroll_w, NULL, 0);
  /* creates entries for the history list */
  t_list = (XmStringTable)XtMalloc(HISTORY_COUNT * sizeof(XmString *));
  h_temp = H_LIST->TAIL;
  if(h_temp == NULL) {
    str = XmStringCreateSimple("NO ENTRY");
    n = 0;
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNtraversalOn,False); n++;
    XtSetArg(wargs[n],XmNautoUnmanage,True); n++;
    XtCreateManagedWidget(str, xmLabelWidgetClass,
                          scroll_area, wargs, n);
    XmStringFree(str);
```

```
  }
  for (count = 0; (count < HISTORY_COUNT) && (h_temp != NULL); count++) {
    str = XmStringCreateSimple(h_temp->trans_label);
    h_temp = h_temp->prev;
    n = 0;
    XtSetArg(wargs[n],XmNlabelString,str); n++;
    XtSetArg(wargs[n],XmNtraversalOn,True); n++;
    XtSetArg(wargs[n],XmNautoUnmanage,True); n++;
    XtCreateManagedWidget(str, xmLabelWidgetClass,
                          scroll_area, wargs, n);
    XmStringFree(str);
  }
  /* creates a button widget */
  n = 0;
  str = XmStringCreate("DONE",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n],XmNlabelString,str); n++;
  XtSetArg(wargs[n],XmNrecomputeSize,False); n++;
  XtSetArg(wargs[n],XmNshadowThickness,4); n++;
  q_button = XtCreateManagedWidget("done", xmPushButtonWidgetClass,
                                   rowcol, wargs, n);
  XmStringFree(str);
  XtAddCallback(q_button, XmNactivateCallback, DoneExeSeq,
                board);
  XtManageChild(scroll_area);
  XtManageChild(board);
}


/*
DoneExeSeq() as a callback function pops down the display window.
*/

void DoneExeSeq(w, widget, call_data)
Widget    w;
Widget    widget;
XtPointer call_data;
{
  XtUnmanageChild(widget);
}


/*
ViewHistoryFile(), as a callback function displays the history file.
*/

void ViewHistoryFile(w, client_data, call_data)
Widget    w;
XtPointer client_data;
XtPointer call_data;
{

  char  string[100];

  if(strcmp(trace_filename,"Untitled") != 0 ) {
    sprintf(string,"xterm -e view %s &",trace_filename);
    system(string);
  }
  else {
    PutErrorDialog(drawing, "History file is not created yet.");
  }
}
```

VITA

Muhammad Tariq Hassan

Candidate for the Degree of

Master of Science

Thesis: TOWARDS A GRAPHICAL PETRI NET TOOL

Major Field: Computer Science

Biographical:

Personal Data: Born in Karachi, Pakistan, December 22, 1965, son of Muhammad Badrul Hassan and Sajeda Hassan.

Education: Graduated from Adamjee Science College, Karachi, Pakistan, in October 1984; received Bachelor of Engineering Degree in Computer Systems Engineering from N.E.D. University of Engineering and Technology, Karachi, Pakistan, in October 1990; completed the requirements for the Master of Science degree in Computer Science at Oklahoma State University in July 1993.

Professional Experience: Graduate Research Assistant, University Computer Center, Oklahoma State University, August 1991 to July 1993.