

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

GRAPHICAL USER INTERFACE TOOL KIT FOR PATH- BASED NETWORK POLICY LANGUAGE

by

Tufan Ekin

March 2002

Thesis Advisor:
Second Reader:

Geoffrey Xie
James Bret Michael

Approved for public release; distribution is unlimited.

Report Documentation Page		
Report Date 29 Mar 2002	Report Type N/A	Dates Covered (from... to) -
Title and Subtitle Graphical User Interface Tool Kit for Path-Based Network Policy Language	Contract Number	
	Grant Number	
	Program Element Number	
Author(s) Ekin, Tufan	Project Number	
	Task Number	
	Work Unit Number	
Performing Organization Name(s) and Address(es) Naval Postgraduate School Monterey, California	Performing Organization Report Number	
Sponsoring/Monitoring Agency Name(s) and Address(es)	Sponsor/Monitor's Acronym(s)	
	Sponsor/Monitor's Report Number(s)	
Distribution/Availability Statement Approved for public release, distribution unlimited		
Supplementary Notes The original document contains color images.		
Abstract		
Subject Terms		
Report Classification unclassified	Classification of this page unclassified	
Classification of Abstract unclassified	Limitation of Abstract UU	
Number of Pages 415		

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE March 2002		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE: Graphical User Interface Tool Kit For Path-Based Network Policy Language			5. FUNDING NUMBERS	
6. AUTHOR (S) Tufan Ekin				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA and NASA			10. SPONSORING/MONITORING AGENCY REPORT NUMBER G417	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the U.S. Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) The Path-based Policy Language (PPL) is a formal network policy language for constructing models of Internet service and access control. Seven changes have been made to the LEXER and YACC code of PPL. Five of the changes are related to the syntax of policy rules in PPL. Two of the changes are related to the semantics of the language. A graphical user interface tool kit for creating, validating, archiving and compiling policies represented in PPL has been developed. The tool kit has a field-by-field interface that allows a policy maker to input and update PPL compliant policy rules, while hiding the subtle details of the PPL syntax from the user. Prior to the work reported here, policy files were created separately and the PPL compiler had to be invoked manually from a command line interface. The GUI combines the processes of forming a policy file and compiling. These processes are performed automatically from the menu items of the tool kit. The GUI itself is password protected, permitting only authorized users access to the system. Protection of the policy rules is also provided via the tool kit.				
14. SUBJECT TERMS Policy Language, Path-Based, Graphical User Interface, Network Management, Conflict Detection, Conflict Resolution, GUI, Network Policy Language			15. NUMBER OF PAGES 413	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

**GRAPHICAL USER INTERFACE TOOL KIT FOR PATH-BASED POLICY
LANGUAGE**

Tufan Ekin
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1996

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
March 2001**

Author: Tufan Ekin

Approved by: Geoffrey Xie
Thesis Advisor

James Bret Michael
Second Reader

Chris Eagle
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Path-based Policy Language (PPL) is a formal network policy language for constructing models of Internet service and access control. Seven changes have been made to the LEXER and YACC code of PPL. Five of the changes are related to the syntax of policy rules in PPL. Two of the changes are related to the semantics of the language.

A graphical user interface tool kit for creating, validating, archiving and compiling policies represented in PPL has been developed. The tool kit has a field-by-field interface that allows a policy maker to input and update PPL compliant policy rules, while hiding the subtle details of the PPL syntax from the user. Prior to the work reported here, policy files were created separately and the PPL compiler had to be invoked manually from a command line interface. The GUI combines the processes of forming a policy file and compiling. These processes are performed automatically from the menu items of the tool kit.

The GUI itself is password protected, permitting only authorized users access to the system. Protection of the policy rules is also provided via the tool kit.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	NETWORK POLICIES IN GENERAL	1
B.	PATH-BASED NETWORK POLICY LANGUAGE	2
C.	CONTRIBUTIONS OF THIS THESIS	3
II.	CHANGES MADE TO PATH-BASED NETWORK POLICY LANGUAGE ..	5
A.	PPL POLICY FORMAT AND SEMANTICS	5
B.	DESCRIPTION OF CHANGES	5
1.	Change 1	6
2.	Change 2	6
3.	Change 3 and Change 4	6
4.	Change 5	7
5.	Change 6	7
6.	Change 7	8
III.	GRAPHICAL USER INTERFACE TOOL KIT FOR PATH-BASED NETWORK POLICY LANGUAGE	9
A.	LOGIN PROCESS	13
B.	USER ACCOUNT MANAGEMENT	14
1.	User Account Creation	14
2.	User Account Deletion	16
C.	NETWORK DEFINITION	17
1.	Node And Node Parameter Creation	17
2.	Node Deletion	19
3.	Link and Link Parameter Creation	20
4.	Link Deletion	21
5.	Path and Path Parameter Creation	22
6.	Path Deletion	25
7.	Class Creation	26
8.	Class Deletion	28
9.	Type Creation	28
10.	Type Deletion	30
D.	POLICY DEFINITION	31
1.	Policy Creation	31
a.	Policy ID	33
b.	Policy Path	34
c.	Policy Target	35
d.	Policy Condition	37
e.	Policy Action	46
2.	Policy Modification	48
3.	Policy Deletion	50

E.	FILE MANAGEMENT	51
1.	Creating a New File	51
2.	Saving and Opening a File	52
a.	<i>Saving a File</i>	52
b.	<i>Opening a File</i>	53
3.	Saving and Importing Policies	55
a.	<i>Saving Policies</i>	55
b.	<i>Importing Policies</i>	56
4.	Compiling	58
V.	CONCLUSION	61
A.	SUMMARY	61
B.	FUTURE WORK	61
	APPENDIX A. CHANGED VERSION OF PPL SCAN.FLEX FILE	63
	APPENDIX B. CHANGED VERSION OF PPL PARSER.YACC FILE	69
	APPENDIX C. PPL MANAGER SOURCE CODE	89
	LIST OF REFERENCES	399
	INITIAL DISTRIBUTION LIST	401

LIST OF FIGURES

Figure 1.1	Path-based Network Policy Language Working Steps ..	3
Figure 3.1	Network Configuration Utilizing PPL	12
Figure 3.2	Login Screen	13
Figure 3.3	GUI After Login Process	14
Figure 3.4	User Account Creation Window	15
Figure 3.5	User Account Display	16
Figure 3.6	User Account Deletion Window	17
Figure 3.7	Node Creation Window	18
Figure 3.8	Node and Node Parameter Display	19
Figure 3.9	Node Deletion Window	19
Figure 3.10	Link Creation Window	21
Figure 3.11	Link and Link Parameter Display	21
Figure 3.12	Link Deletion Window	22
Figure 3.13	Example Network for Wild Card Character	23
Figure 3.14	Path Creation Window	24
Figure 3.15	Path and Path Parameter Display	25
Figure 3.16	Path Deletion Window	25
Figure 3.17	Class Creation Window	27
Figure 3.18	Class Display	27
Figure 3.19	Class Deletion Window	28
Figure 3.20	Type Creation Window	29
Figure 3.21	Type Display	30
Figure 3.22	Type Deletion Window	30
Figure 3.23	Policy Creation Window	31
Figure 3.24	Policy Path Definition Display	32
Figure 3.25	Policy Display	33
Figure 3.26	Policy ID Creation Window	34
Figure 3.27	Policy Path Creation Window	35
Figure 3.28	Policy Target Creation Window	36
Figure 3.29	Policy Condition Creation Window	37
Figure 3.30	Priority Condition Creation Window	38
Figure 3.31	Hop Count Condition Creation Window	39
Figure 3.32	Time Condition Creation Window	40
Figure 3.33	Source IP Address Condition Creation Window	41
Figure 3.34	Bandwidth Condition Creation Window	42
Figure 3.35	User ID Condition Creation Window	43
Figure 3.36	Type Condition Creation Window	44
Figure 3.37	Parameter Condition Creation Window	45
Figure 3.38	Policy Action Creation Window	47
Figure 3.39	Policy Modification Window - 1	49
Figure 3.40	Policy Modification Window - 2	50
Figure 3.41	Policy Deletion Window	51

Figure 3.42	Confirmation Window When Creating a New File	52
Figure 3.43	File Chooser for Saving Files	53
Figure 3.44	File Chooser for Opening Files	54
Figure 3.45	Error Message in Opening a File	54
Figure 3.46	File Chooser for Saving Policies	56
Figure 3.47	File Chooser for Importing Policies	57
Figure 3.48	Error Message in Importing Policies	57
Figure 3.49	Setting Compilation Mode	59
Figure 3.50	Compiler Output	59

ACKNOWLEDGEMENTS

I would like to thank to Professor Geoffrey Xie and Professor James Bret Michael for their guidance and my friend Cihat Eryigit for his help throughout this thesis research.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. NETWORK POLICIES IN GENERAL

In the general sense, policies represent business goals and objectives, and describe how resources are allocated to meet these goals and objectives. With respect to networking, policy refers to the ability to administer and manage network elements in order to provide a set of services to all clients of the network. “Clients” in this case refers to users as well as applications and services. [Ref.1].

Strassner and Ellessen [Ref.1] define a network policy as an aggregation of policy rules. According to their definition, each policy rule is comprised of a set of conditions and a corresponding set of actions. The conditions define when the policy rule is applicable. Once a policy rule is so activated, one or more actions contained by that policy rule may then be executed. These actions are associated with either meeting or not meeting the set of conditions specified by the policy rule.

Policy-based networking is the control of networking environments via creation and enforcement of policies. It provides a formal framework for managing network quality of service (including security rights) and client priority. With the convergence of video, voice and data traffic on networks, network management has become a very complex task to perform without a well-defined framework. That is why there has been growing interest on policy-based networking in recent years.

This thesis addresses some of the advantages of network policies, identified as follows. Network policies allow administrators to control the use of the network. Priority can be given by means of network policies to business-critical application traffic like IP telephony during congestion times. Without policies, network performance experienced by a client would be unpredictable. Such performance is detrimental to real-time applications such as video conferencing. Network policies also provide the ability to restrict the use of some parts of the network by denying access to those parts by unwanted traffic.

B. PATH-BASED NETWORK POLICY LANGUAGE

Path-based network policy language (PPL) is a network policy language where all attributes associated with the policy, which include the service type of the traffic, conditions used to trigger the policy and the actions executed when the policy is triggered, are all bound to a predefined set of paths. Using the path as the fundamental building block of a policy statement provides some degree of control and flexibility. The ability to specify an explicit path, which identifies each node from source to destination, enables policy makers to create virtual channels where resources are reserved to support real-time applications. [Ref.2].

The main focus of the previous PPL work in [Ref.2] is the verification of the policies that are going to be applied throughout the network. Stone states in [Ref.2] that the policies specified by different people at different times cannot be enforced in a network if they have conflicts with each other. The PPL compiler parses policy rules specified in PPL and detects conflicting policies before they are distributed to the policy enforcement points.

The steps of how PPL-based tools may be used are depicted in Figure 1.1. The policy maker first creates an ASCII based PPL source file that contains network topology information and policy rules in PPL. Next, the PPL compiler is used to parse the source file, generating PROLOG logic statements. Then, the compiler invokes the PROLOG interpreter to detect conflicts among the generated logic statements. The results are presented to the policy maker in an ASCII based log file. According to the results obtained, the policy maker repeats the process of modifying policy rules and validating them until a conflict-free set of policies is obtained. Then the validated policies are distributed to a policy server where they can be used to regulate the network.

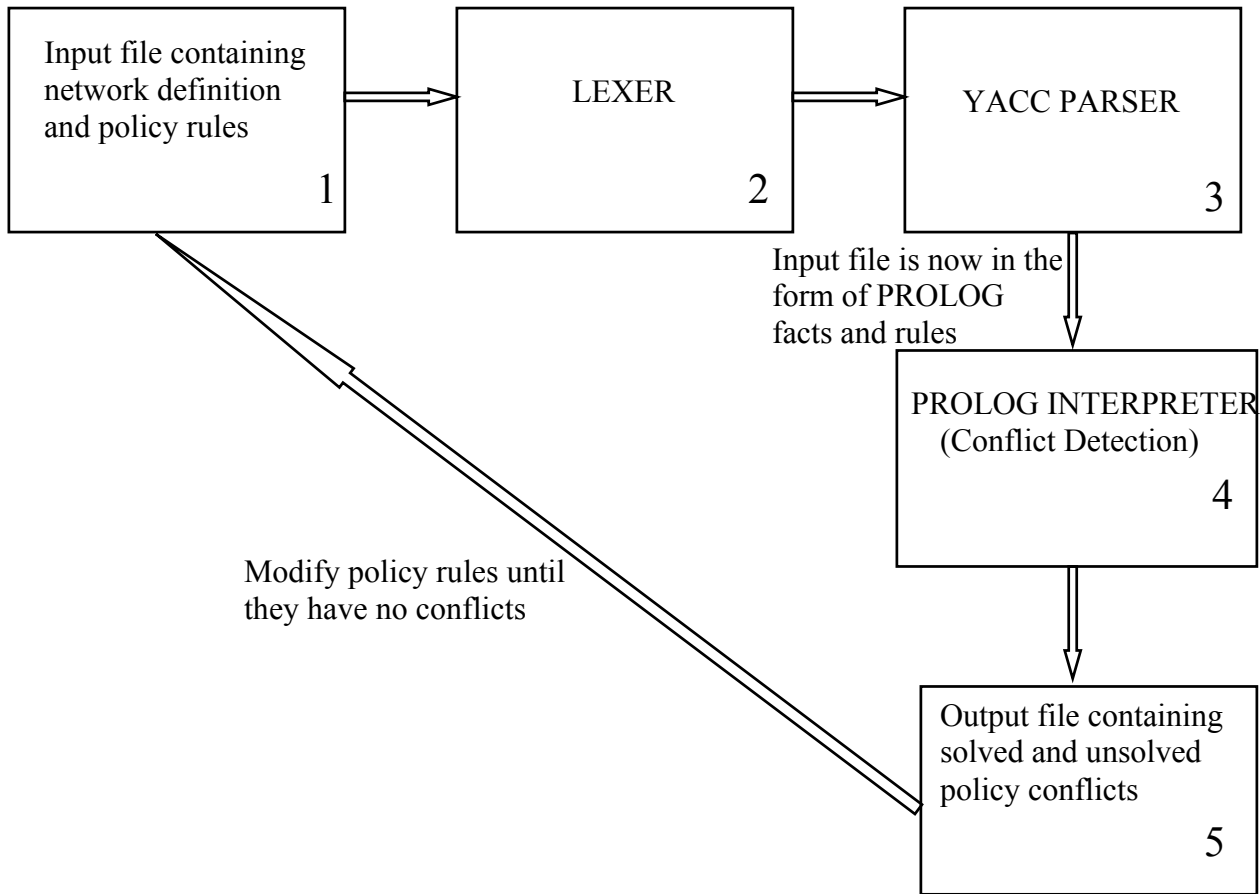


Figure 1.1 Path-based Network Policy Language Working Steps

C. CONTRIBUTIONS OF THIS THESIS

The first part of this thesis develops seven changes to the LEXER (lexical analyzer) and the YACC code of path-based network policy language. Most of these changes are related to the syntax of policy rules in PPL. The purpose of the changes is to put policy rules in the conventional format of high-level programming languages.

The second and primary contribution of this thesis provides a graphical user interface (GUI) tool kit for creating, validating, archiving, and compiling policies. The JAVA programming language was used for developing the GUI tool kit. In [Ref.2], network topology definition and policy rules associated with that network topology are typed into an ASCII file, compiled, and scanned with the command line interface of the Bourne Again SHell (BASH). Policy creators have to know all the rules, keywords, and

syntax of the language to create the policy file. They have to spend a considerable amount of time learning the rules and the syntax of the language, as well as writing the policy file. When they make a syntax mistake, they have to return to the policy file to correct it after compilation. In contrast, the GUI provided by this thesis has a natural field-by-field interface that provides a syntax-free policy rule creation. Only valid choices are presented to the policy creator for completion of a policy rule by using input from the pre-existing fields. The policy creator invokes compilation via the GUI. The GUI tool kit also provides for semantic checking and aggregating multiple policy files. As a result, the detailed rules and the syntax of the language are made transparent to the user of the GUI. It is expected that this will result in a reduction in the effort and time spent learning the PPL grammar and policy creation processes, as well as fewer errors during the creation process. However, experiments with human subjects need to be conducted to verify these expectations.

The GUI is password protected. All users must log on to the system to be able to create or modify policies. Policies are associated with the user currently logged on. This prevents an ordinary user from assigning a “network manager” identifier to a policy in hopes of guaranteeing its implementation.

II. CHANGES MADE TO PATH-BASED NETWORK POLICY LANGUAGE

A. PPL POLICY FORMAT AND SEMANTICS

In PPL, each policy rule has the following format as documented in [Ref. 3]:

*Policy ID Policy Creator @ {Target Paths} {Target Traffic} {Path Conditions}
{Action Items};*

Where:

Policy ID is a unique policy identification token (“policy1”, “catch21”, etc.) associated with the rule.

Policy Creator is the identification of the policy creator who has made this rule. (One may assume that each policy maker must first obtain a unique login ID from the network administrator.)

Target Paths defines the set of network paths that this rule affects.

Target Traffic defines the set of user packets that this rule affects. Items are represented in disjunctive normal form.

Path Conditions are any conditions associated with a target path. Some conditions are global, like time or day. Items are represented in conjunctive normal form.

Action Items are used for explicit accept/deny or for setting parameters such as policy priority.

The underlying semantic of a policy rule is: *Action Items* are performed for *Target Traffic*, which traverse the *Target Paths*, only if *Path Conditions* are true.

B. DESCRIPTION OF CHANGES

Seven changes have been made to the PPL language and compiler. The first five changes deal with the syntax of PPL policy rules, making the syntax similar to that of high-level languages. The sixth change adds support for a domain suffix to node

definitions in PPL. The final change provides for setting four additional parameters in the *Action Items* element. These changes are explained in detail in the remainder of this section.

1. Change 1

The sign “@” was added before the *Target Paths* element of a policy rule to mark the start of target paths definition.

The PPL rule format before the change was:

```
Policy ID Policy Creator {Target Paths} {Target Traffic} {Path Conditions}
{Action Items};
```

The format after the change is:

```
Policy ID Policy Creator @ {Target Paths} {Target Traffic} {Path Conditions}
{Action Items};
```

2. Change 2

The signs “{}” around user defined class members used in the *Target Traffic* element of a policy rule were changed to “<>” since each of these traffic class members is an element of a set. In PPL, “<>” is the standard notation for defining a set.

An example rule in the old format was:

```
Policy1 Net_Manager @ {<1, 2, 5>} {userGroup == {Faculty}} {*} {permit};
```

After the change the rule is:

```
Policy1 Net_Manager @ {<1, 2, 5>} {userGroup == <Faculty>} {*} {permit};
```

3. Change 3 and Change 4

In the *Target Traffic* element of a policy, the comma used to separate items was changed to “||”. This reflects the logic that all attributes in the *Target Traffic* element of a policy are OR’ed together. In the *Path Conditions* element of a policy, the comma

separating items was changed to “&&”to reflect that all attributes in the *Path Conditions* element of a policy are AND’ed together.

The reason for these two changes was to follow the syntax conventions of widely used programming languages like JAVA.

An example rule before the changes was:

```
Policy5 Net_Manager @ {nps_darpa} {traffic_type == <university> , traffic_class == <video>} {time >= 1100 , BW <= 120.0 MBPS , day == Monday} {deny};
```

After the changes the rule is:

```
Policy5 Net_Manager @ {nps_darpa} {traffic_type == <university> || traffic_class == <video>} {time >= 1100 && BW <= 120.0 MBPS && day == Monday} {deny};
```

4. Change 5

The “*hostIP*” keyword used in the *Path Conditions* element of a policy rule was changed to “*srcIPAddress*”.

An example rule in the old format was:

```
Policy7 Net_Manager @ {nsf} {*} {hostIP != 131.*.*} {deny};
```

With the change the rule becomes:

```
Policy7 Net_Manager @ {nsf} {*} {srcIPAddress != 131.*.*} {deny};
```

5. Change 6

This change enables a domain suffix to be added to the definition of a node.

An example node definition before the change was:

```
define node cs;
```

The new definition is:

```
define node cs_nps;
```

6. Change 7

Policy creators can explicitly deny or permit target traffic in the *Action Items* element of a policy rule. They can also set *priority* and/or *hop count* parameters for target traffic in this element. The change permits the user of PPL to set *allocated bandwidth*, *maximum loss rate*, *delay bound* and *security level* parameters in the *Action Items* element of a policy rule.

An example rule in the old format was:

```
Policy11      Net_Manager @      {nps_darpa}  {traffic_class == <video>}  {time  
>= 1601}      {permit, priority:=5, hopCount:=19};
```

With the format change the rule becomes:

```
Policy11      Net_Manager @      {nps_darpa}  {traffic_class == <video>}  {time  
>= 1601}      {permit, maxLossRate:=0.01%, delayBound:=5 msec, securityLevel:=2,  
allocated_bw:=100.0 MBPS};
```

III. GRAPHICAL USER INTERFACE TOOL KIT FOR PATH-BASED NETWORK POLICY LANGUAGE

The need for efficiency and productivity in the process of creating, validating, archiving, and compiling network definitions and policies in PPL led to the development of the GUI tool kit, “PPL Manager.” Shneiderman [Ref. 4] defines usability of a computer program as a combination of the following user-oriented characteristics:

- Ease of learning,
- High speed of user task performance,
- Low user error rate,
- Subjective user satisfaction, and
- User retention over time.

The PPL Manager described in this thesis is designed to add value in all of these aspects. Conventional GUI components such as menus, menu items, text fields, text areas, and buttons have been used in PPL Manager. Several steps, such as creating a policy file and compilation, are all performed from PPL Manager. Only valid choices are presented to the policy creator for completion of a policy rule by using input from the previous fields. All the choices and the functionality of PPL are ready to be used easily and quickly. Hix & Hartson [Ref. 5] state that communication is at least as important as computation to the users of an interactive system. The PPL Manager is intended to facilitate the communication between policy makers and the PPL increasing usability.

Using the PPL Manager, policy makers do not have to know the detailed syntax of PPL. When a policy maker wants to create a network element or a policy, the validity of input elements is checked during the creation process. As soon as an error in the user input is detected, the policy maker is notified of the occurrence and is not allowed to proceed further before correcting the error. It is expected that this will result in users of the tool making fewer errors and saving on effort and time in both learning the PPL

grammar and policy creation processes. Experiments with human subjects need to be conducted to verify the preceding claims.

The PPL Manager requires all users to login via a user ID and a password. Policies are associated with the login IDs of users. Each login ID is assigned a priority value. Therefore, the login process serves to prevent both the impersonation of an authorized user by an intruder and impersonation of another user with higher priority by an authorized user. This step is very important since conflicts between policies are first resolved according to the priority of the policy makers. Priority is part of a user account and specified for each user at the time of user account creation. It is an integer number. One is the highest priority value that can be given to a policy maker.

With the PPL Manager, policy rules created by different users for a particular network topology can be aggregated into one set and compiled by a network administrator to detect possible conflicts. When conflicts are detected, the administrator will request the creators of these conflicting policies to make revisions. The administrator will then compile the entire policy set again, and will repeat the above process until a conflict free set of policies is obtained.

A similar application for policy management is the “Policy Tool” [Ref. 6] GUI tool kit. “Policy Tool” was developed by Sun Microsystems for a JAVA runtime environment, specifying which permissions are available for code from various sources. The default policy implementation obtains its information from static ASCII policy configuration files. A policy file can be composed via a simple text editor, or via the graphical “Policy Tool” utility that is very similar to the PPL Manager. Using the “Policy Tool” saves typing and eliminates the need for users to know the required syntax of policy files, thus reducing errors [Ref. 6].

A typical usage scenario of the PPL Manager is shown in Figure 3.1. PPL network definition and policies can be created, revised, saved to a file, or opened from a file using the PPL Manager. At any time, the PPL Compiler may be invoked from the PPL Manager to detect conflicts among the current set of loaded policy rules. This cycle of revising and compiling policies should be repeated until the policies do not generate

conflicts. Once a conflict-free set of policies has been obtained, they may be loaded into a policy server where they are enacted to regulate the network. PPL Manager does not currently implement loading policies into the policy server. Network devices will provide the policy server information about the current state of the network, such as delay or loss rate of a link or a path, and other such information as required by dynamic PPL policies [Ref. 2:p. 39]. Each one of the numbered flows in Figure 3.1 are explained below:

- 1 - Network definition and policies are saved to PPL policy file
- 2 - Network definition and policies are opened from PPL policy file
- 3 - Compiler is invoked
- 4 - The result of the compiler output is displayed in PPL Manager
- 5 - Conflict-free set of policies is distributed to the policy server. This step is not currently implemented in PPL Manager
- 6 - Network devices provide information about the current state of the network, such as delay or loss rate
- 7 – Policy server responds to the requests for network resources

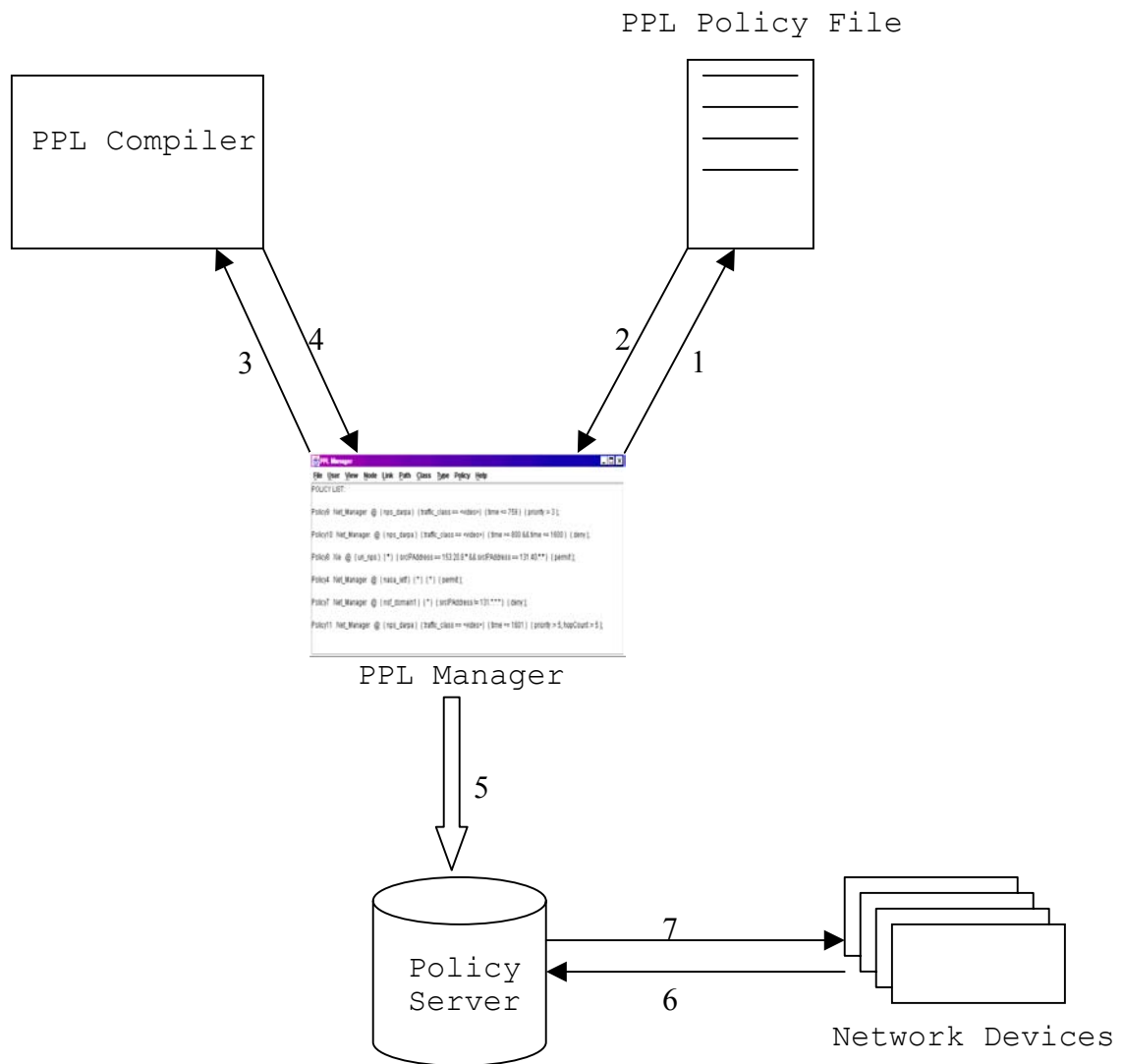


Figure 3.1 Network Configuration Utilizing PPL

The components of the PPL Manager and how the first four of the above tasks are accomplished are explained in detail in the remainder of this chapter.

A. LOGIN PROCESS

The login screen of PPL Manager is shown in Figure 3.2. The program checks the login ID and the password entered by the user against the current list of login IDs and passwords of authorized users. If the login information is correct, then the GUI used for defining network topology and updating policies is launched. A screen shot of the GUI is depicted in Figure 3.3. Otherwise, the user is not permitted access to the PPL Manager.

Management of user (policy creator) accounts is supported by the “User” pull-down menu and the “View Users” menu item under the “View” pull-down menu of the GUI in Figure 3.3. These menu choices are only available to the administrator. They are grayed out if the user is not the administrator. Thus, only the administrator can create, delete, or view user accounts. User account management is explained in detail in the next section of this chapter.

After a user successfully logs on, the *Policy Creator* element of every policy the user creates will automatically be set to the login ID of the user. The reason for this is to prevent an opportunistic user from impersonating a higher priority policy creator to guarantee the implementation of his/her policies.

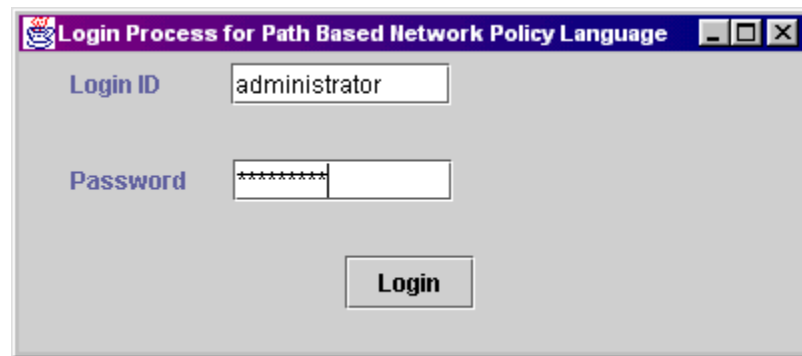


Figure 3.2 Login Screen

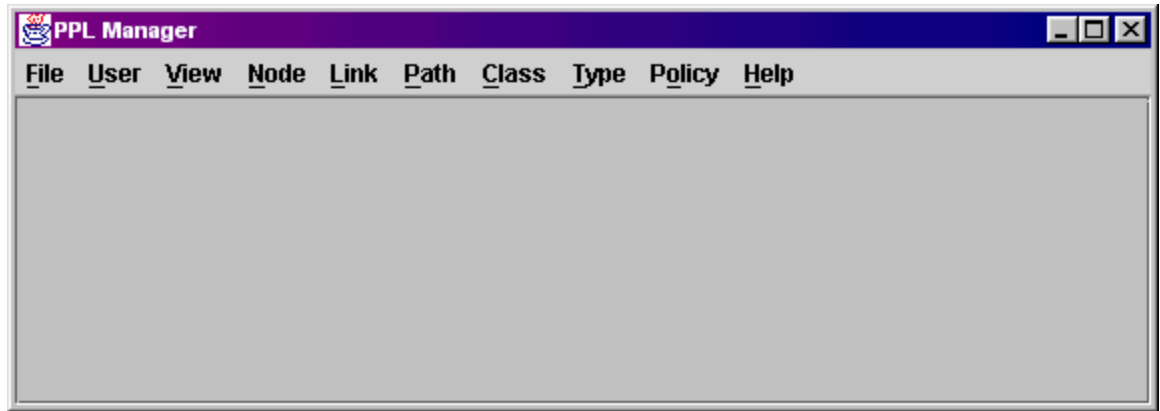


Figure 3.3 GUI After Login Process

B. USER ACCOUNT MANAGEMENT

Each user account is composed of a login ID, a password and a priority value. Login ID and password are used to differentiate among different users and for access control as in many other computer systems. Priority is a very important attribute of a PPL policy maker. This is because, the policy created by the user who has higher priority value will override the policy created by the user with a lower priority value, if the two policies conflict. If two policy makers have the same priority, the conflicts will not be solved automatically by the compiler but will be displayed to the policy maker.

User account information is saved in a text file by using the JAVA *ObjectOutputStream* class. Even though this file can be opened without restriction, text in this file is not in a readable form. This provides some degree of security, but additional security measures, like file access control, are needed to restrict access to this file. Such security measures are beyond the scope of this thesis.

The GUI support for creating, deleting, and displaying user accounts are explained in detail in the remainder of this section.

1. User Account Creation

The administrator creates a user account by selecting the “Create User” menu item under the “User” menu. A new dialog, shown in Figure 3.4, is invoked upon this

selection. The administrator writes the login ID, password, and priority of the user in the text fields of “User Account Creation Window”.

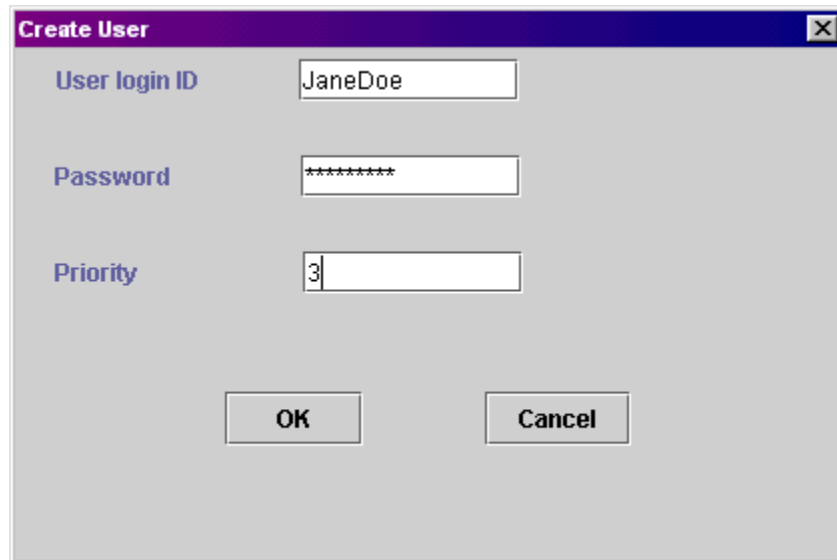
The image shows a standard Windows-style dialog box titled "Create User". It has a purple title bar with a close button (X) in the top right corner. The main area is light gray and contains three text input fields, each with a label to its left. The first field is labeled "User login ID" and contains the text "JaneDoe". The second field is labeled "Password" and contains seven asterisks "*****". The third field is labeled "Priority" and contains the number "3". At the bottom of the dialog, there are two buttons: "OK" on the left and "Cancel" on the right.

Figure 3.4 User Account Creation Window

White space characters are not allowed in a login ID string because the PPL compiler does not accept white space characters in this string. Another check for preventing duplicate user account creation is also made: no two users are allowed to have the same login ID.

The value that the administrator enters as user priority is checked so that it will be an integer greater than or equal to one.

The administrator can view all current user accounts by selecting the “View Users” menu item under the “View” pull-down menu. Figure 3.5 displays how user accounts are displayed when the administrator selects this option. A PPL policy creator definition statement is generated and displayed for each user.

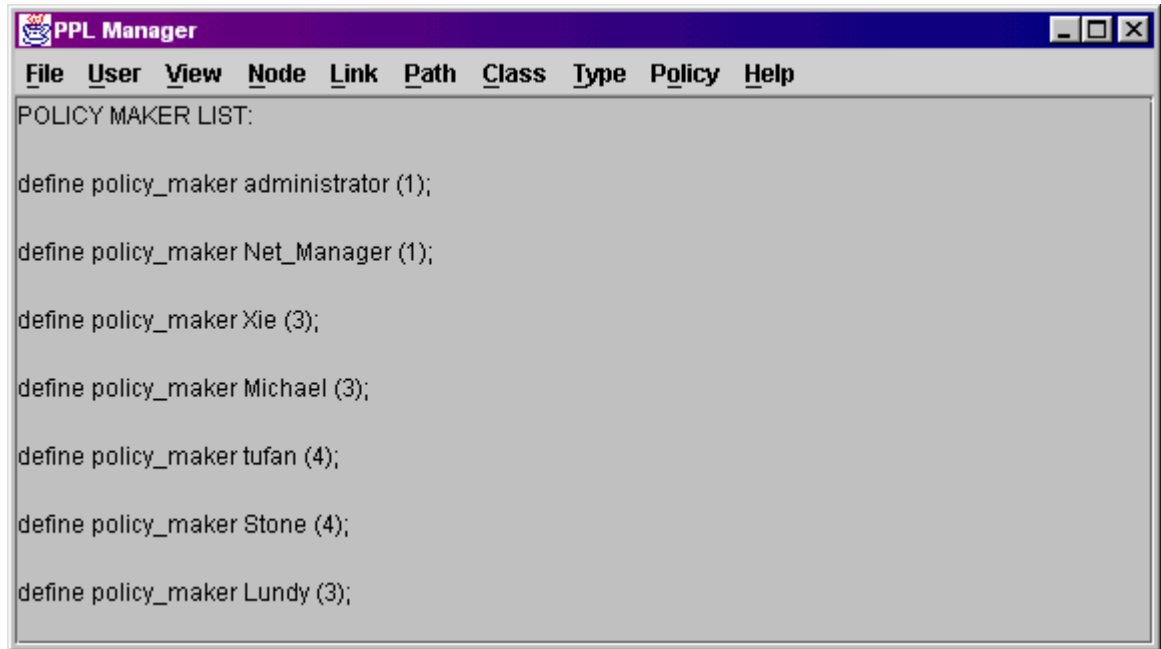


Figure 3.5 User Account Display

The administrator can save the user accounts by selecting the “Save current users” menu item under the “User” menu. When the administrator exits the program by closing the PPL Manager or by selecting “Exit” menu item under “File” menu, user accounts are also saved automatically.

2. User Account Deletion

The administrator deletes a user account by selecting the “Delete User” menu item under the “User” menu. Then, the “Delete User” modal window, which is shown in Figure 3.6, is launched. In this window, the *Login IDs* of all users are presented to the administrator in a combo box. After selecting the *Login ID* of the user to be deleted from this combo box, the administrator clicks the “Delete” button to delete the selected user account. The administrator account cannot be deleted.

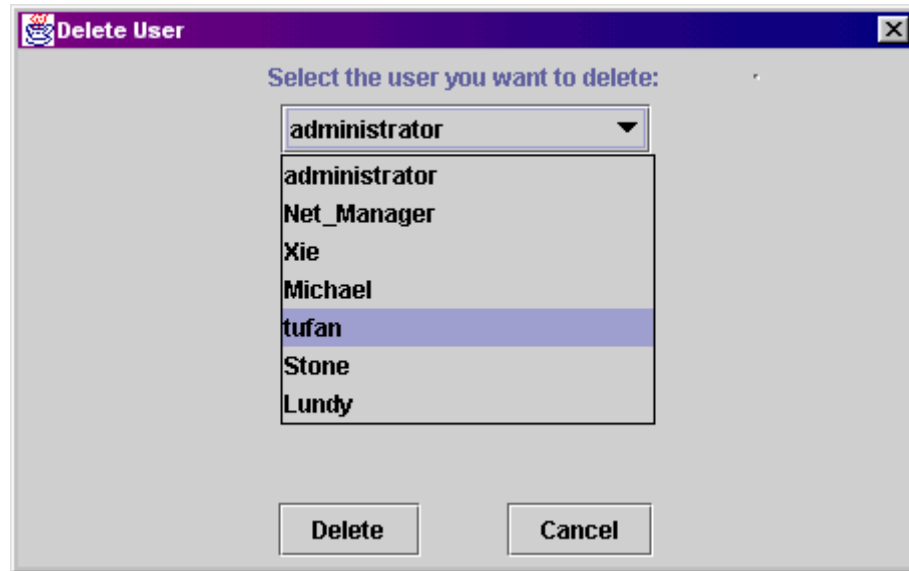


Figure 3.6 User Account Deletion Window

C. NETWORK DEFINITION

Definition of the nodes, links, and paths of a network topology, as well as user defined classes and types used in policy creation and modification, are explained in this section.

1. Node And Node Parameter Creation

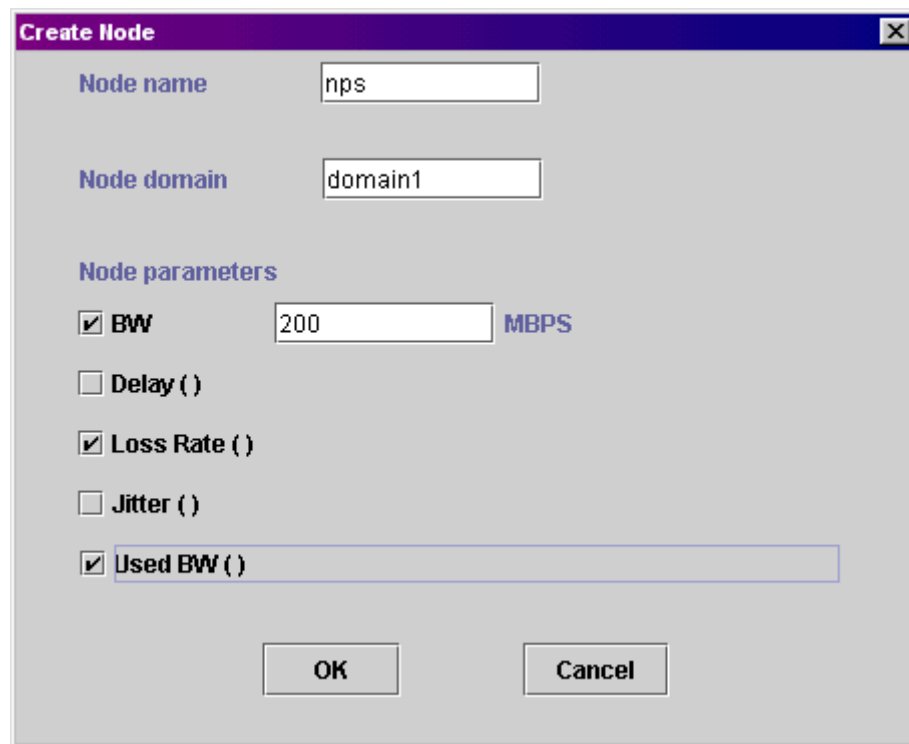
The policy maker defines the nodes in the network topology and the parameters associated with those nodes from the “Node” menu. Parameters that can be associated with a node are: bandwidth, delay, loss rate, jitter and used bandwidth. When the policy maker chooses the “Create Node” menu item under the “Node” menu, a new modal window is created as shown in Figure 3.7. In this window, “Node Name” and “Node Domain” parts are mandatory to be populated since each node in the network must have a name and a domain.

White-space-character checks are made for the “Node Name” and “Node Domain” strings after the user clicks on the “OK” button. The PPL compiler does not accept white space characters in these strings.

The name and domain of the current node are checked against the node name and domain of all the nodes defined so far for the network topology so that there will not be a redefined node. The node redefinition problem occurs when two nodes have the same name and domain.

If the policy maker defines a bandwidth parameter for a node, then this value is checked to make sure it is a positive number.

The policy creator can view the nodes and their parameters by choosing the “View Nodes” menu item under the “View” menu. The display of the nodes is updated as the user adds or deletes nodes. Figure 3.8 shows how the node created in Figure 3.7 is displayed to the user when the “View Nodes” menu item is selected. Node and node parameter definition statements are in the same format as they are in a PPL policy file.



The image shows a "Create Node" dialog box with the following fields and options:

- Node name:** nps
- Node domain:** domain1
- Node parameters:**
 - ☒ **BW** 200 MBPS
 - ☐ **Delay ()**
 - ☒ **Loss Rate ()**
 - ☐ **Jitter ()**
 - ☒ **Used BW ()**
- Buttons:** OK, Cancel

Figure 3.7 Node Creation Window

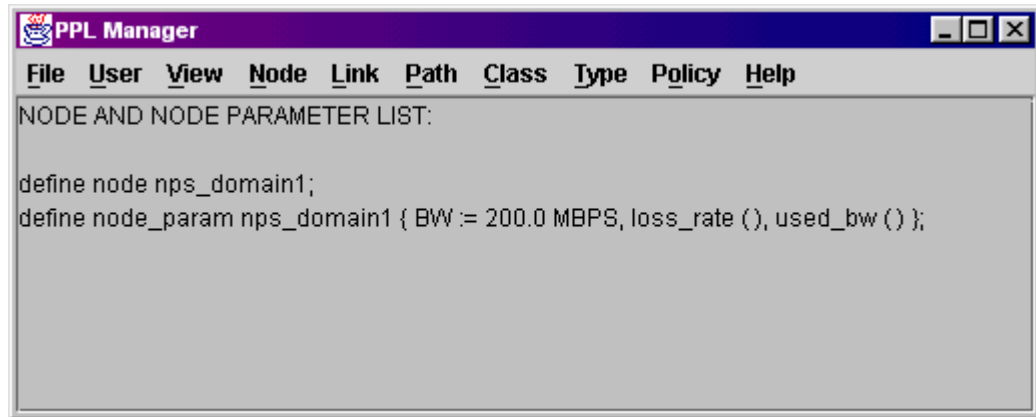


Figure 3.8 Node and Node Parameter Display

2. Node Deletion

The policy creator can delete a node by selecting the “Delete Node” menu item under the “Node” menu in Figure 3.8. Then the “Delete Node” modal window, which is shown in Figure 3.9, is displayed. In this window, the names and the domains of all the nodes are presented to the user in a combo box. After selecting the node to be deleted from the combo box, the “Delete” button is clicked to remove that node.

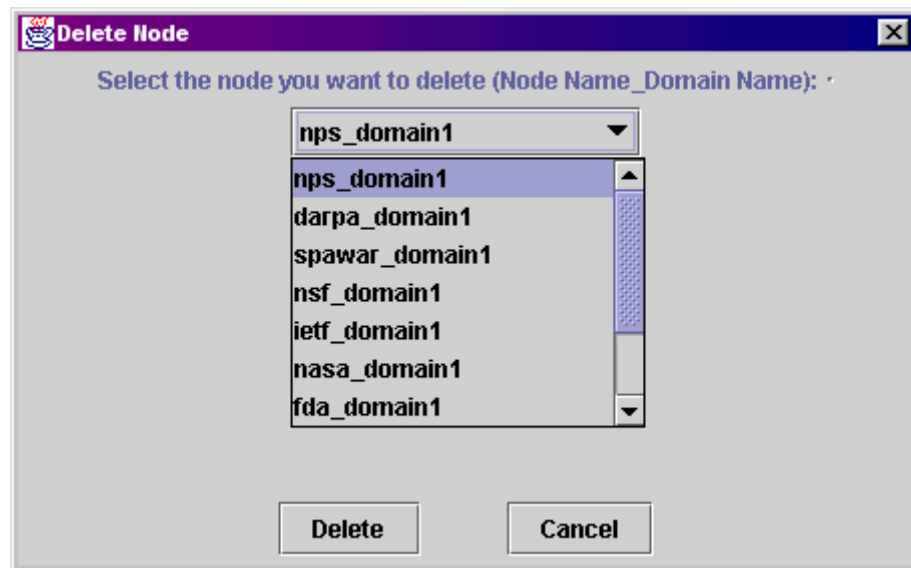


Figure 3.9 Node Deletion Window

3. Link and Link Parameter Creation

The policy maker defines the links and the parameters associated with these links from the “Link” menu. Parameters that can be associated with a link are the same as the parameters of a node. When the policy maker chooses the “Create Link” menu item under the “Link” menu, a new modal window is created as shown in Figure 3.10. In this window, the “Link Name” must be entered since each link in the network must have a name. The two end-nodes of the link are selected from different combo boxes in this “Create Link” window. Each node combo box lists all the defined nodes. Since two nodes of the link cannot be the same, the policy maker is not allowed to select the same node as the two end-points.

The following validation steps are performed to ensure a correct link definition is made. A white-space-character check is made for the “Link Name” string after the user clicks on the “OK” button. If the policy maker defines a bandwidth parameter for a link, then this value is checked to make sure it is a positive real number. The name of the link that is being created is checked against the names of all the existing links of the network topology to ensure that there will not be a redefined link.

The policy creator can see the links and the parameters of the links by choosing the “View Links” menu item under the “View” menu. The display of the links is updated as the user adds or deletes links. Figure 3.11 shows how the link created in Figure 3.10 is displayed to the user when the “View Links” menu item is selected. Link and link parameter definition statements are presented in the same format as in a PPL policy file.

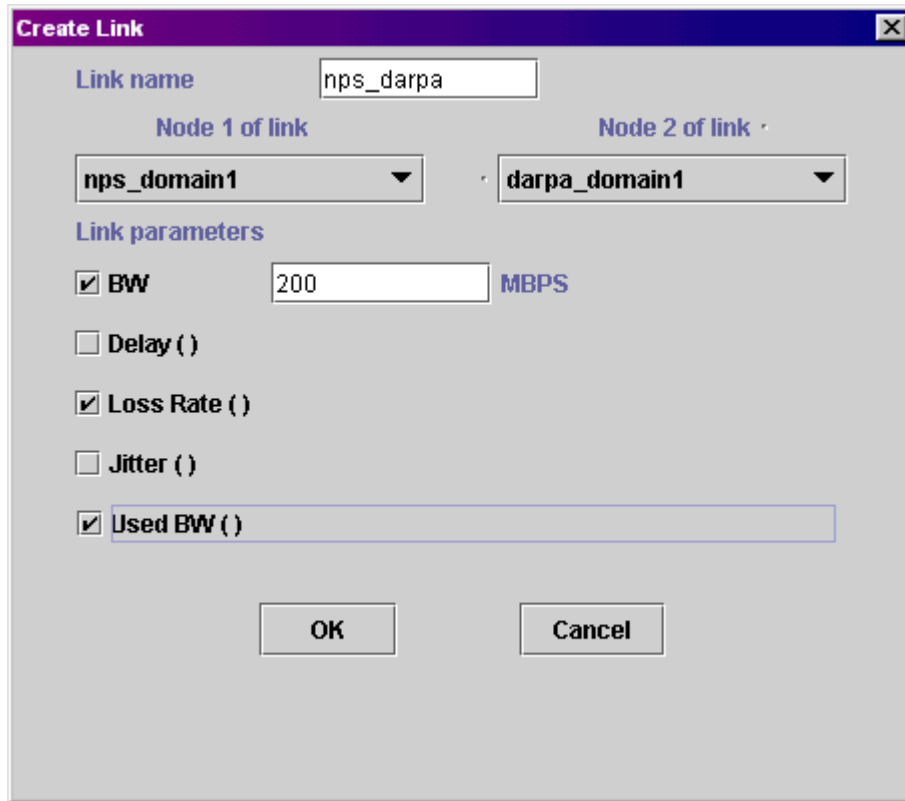


Figure 3.10 Link Creation Window

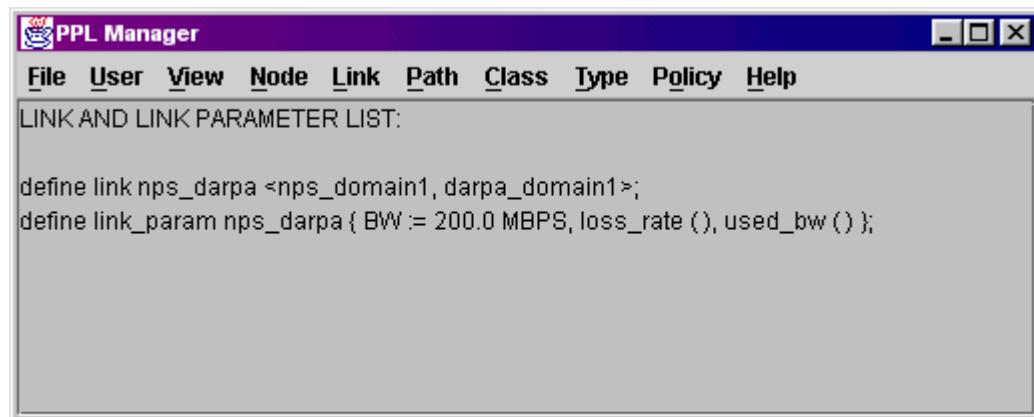


Figure 3.11 Link and Link Parameter Display

4. Link Deletion

The policy maker can delete a link by selecting the “Delete Link” menu item under the “Link” menu. Then, “Delete Link” modal window, which is shown in Figure 3.12, is displayed. In this window, the names of all the links are presented to the user in a

combo box. After selecting the link to be deleted from this combo box, the user selects the “Delete” button to delete that link.

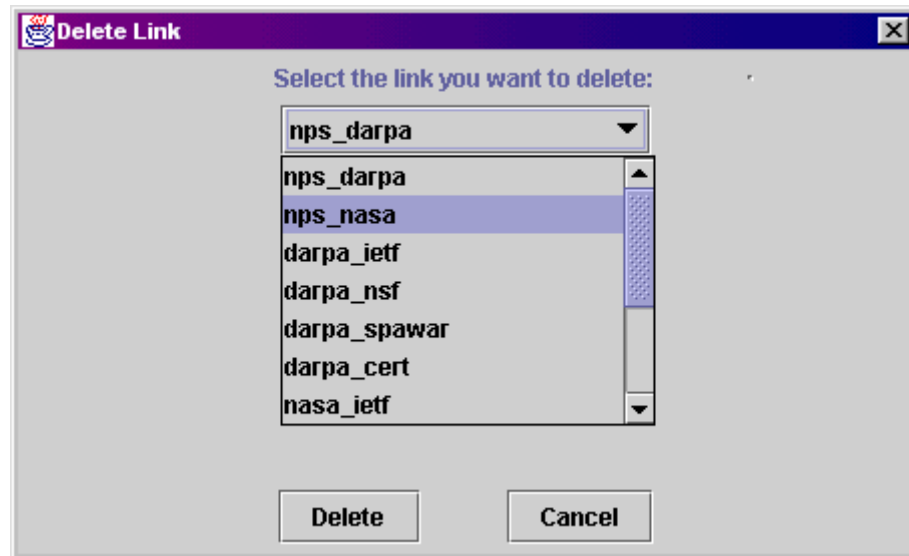


Figure 3.12 Link Deletion Window

5. Path and Path Parameter Creation

A path is composed of at least two nodes. The policy maker is not allowed to create a path that has less than two nodes.

The wild card character, “*,” can be combined with existing nodes to create a path. When the wild card character is used between two nodes in a path, for example node A and node B, then all possible paths connecting the two nodes are defined by the statement. For the network in Figure 3.13, “define path A_B {A, *, B};” statement would include two paths. These paths are {A, C, B} and {A, D, B}.

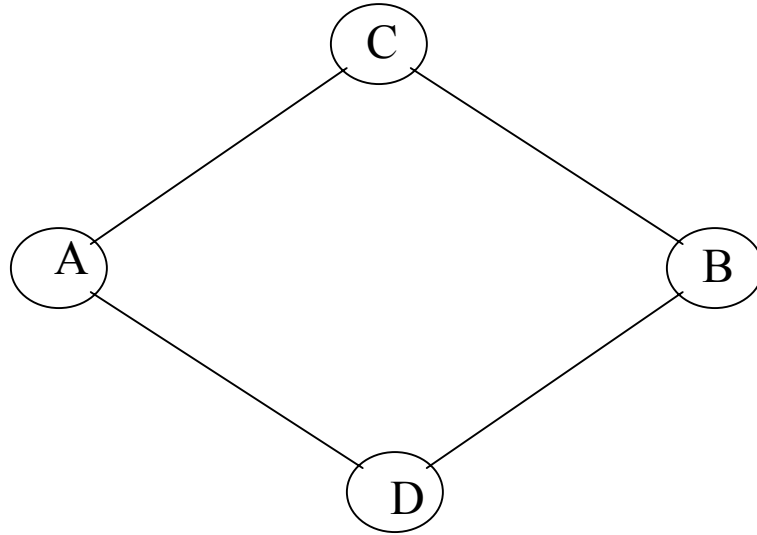


Figure 3.13 Example Network for Wild Card Character

The policy maker defines the paths and the parameters associated with these paths from the “Path” menu. The parameters that can be associated with a path are the same as the parameters for a node. When the policy maker chooses the “Create Path” menu item under the “Path” menu, a new modal window is displayed as shown in Figure 3.14. In this window, the “Path Name” is required to be entered.

The nodes of a path are selected from a combo box in the “Create Path” window. This combo box lists all the nodes defined so far and a wild card character. The same node cannot be included more than one time in a path. Otherwise, that path would have a loop, an unauthorized condition. As the policy creator adds more nodes to a path by clicking the “Add selected node to path” button, the previous nodes and the node the user has just selected are displayed to the user in the order of selection in the display area of the “Create Path” window. An example of the result is shown in the circle in Figure 3.14.

As with the create node process, the following checks are performed by the GUI driver. A white-space-character check is made for “Path Name” string after the user clicks on the “OK” button. If the policy maker defines a bandwidth parameter for a path, then this value is checked to make sure it is a positive real number. The name of the path that is being created is checked against the names of all the paths defined so far for that network topology so that there will not be a redefined path.

The policy maker can view the paths and their parameters by choosing the “View Paths” menu item under the “View” menu. The display of the paths is updated as the user adds or deletes paths. Figure 3.15 shows how the path created in Figure 3.14 is displayed to the user. Path and path parameter definition statements are in the same format as they are in a PPL policy file.

Create Path

Path name:

Select nodes of the path:

Nodes of the path are:

- nps_domain1
- *
- cert_domain1

Path parameters:

☒ BW MBPS

☐ Delay ()

☐ Loss Rate ()

☒ Jitter ()

☐ Used BW ()

Figure 3.14 Path Creation Window

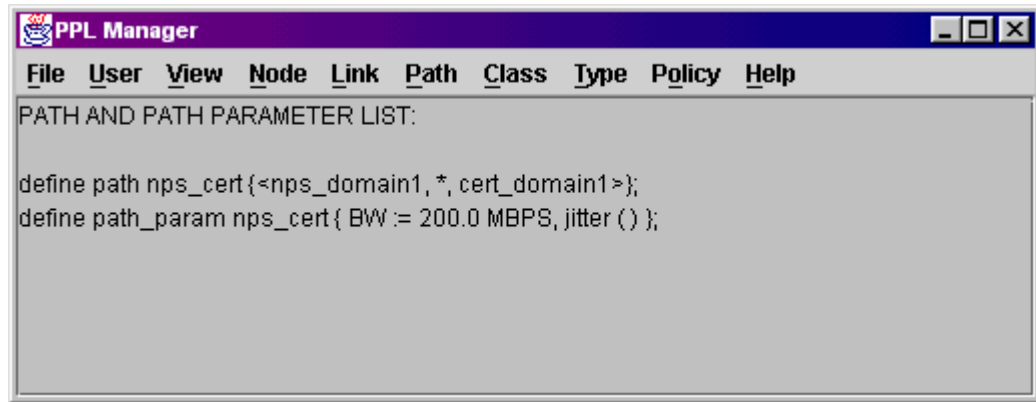


Figure 3.15 Path and Path Parameter Display

6. Path Deletion

The policy creator can delete a path by selecting the “Delete path” menu item under “Path” menu. Then, the “Delete Path” modal window, which is shown in Figure 3.16, is displayed. In this window, the names of all the paths are presented to the user in a combo box. After selecting the path to be deleted from this combo box, the “Delete” button is clicked to delete that path.

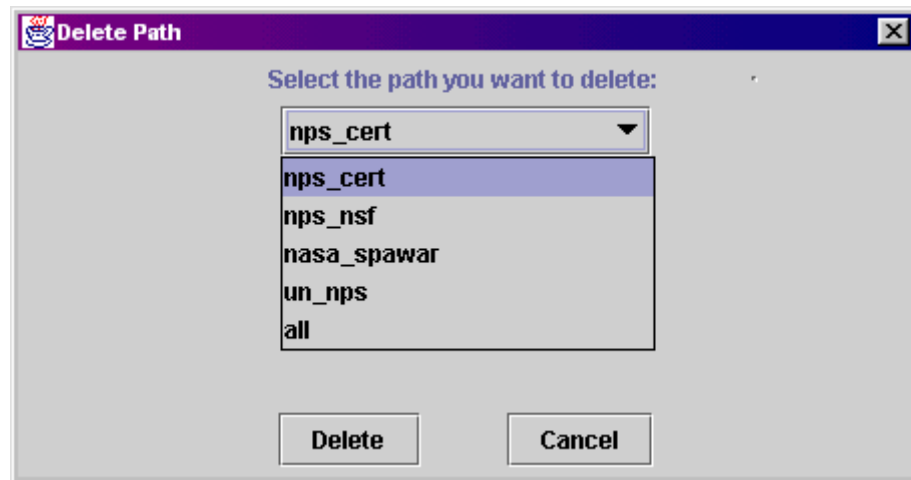


Figure 3.16 Path Deletion Window

7. Class Creation

In differentiated services, packets are marked differently to create several packet classes. Packets in different classes receive different services [Ref. 7]. This allows organizations to receive a better QoS, possibly because they are willing to pay more money. To support policies based on differentiated services, a PPL policy contains a *Target Traffic* field. To utilize this field, a traffic class has to be created prior to its use in a policy [Ref. 2].

Classes are user-defined statements. They are the only statements that can be used in the *Target Traffic* element of policies. Each class has one or more class members. Either equality or inequality statements may be used to set class values. For example, the name of a class could be *traffic_priority* and the members of that class could be *{high, med, low}*. Then, the *Target Traffic* element of a policy would be *traffic_priority == low* or *traffic_priority != high* or two of the previous class members together, represented in a disjunctive normal form.

The policy maker defines the classes and the class members from the “Class” menu. When the policy maker chooses the “Create class” menu item under “Class” menu, a modal window is displayed, as shown in Figure 3.17. In this window, the “Class Name” part is obligatory to be completed because each class must have a name. There is also a check to verify a class has at least one member. Another check is made so to ensure no class members are the same, if there is more than one class member for a particular class.

Class members are written to a text field and added to the class by clicking the “Add class member” button. After a member is added, the user can see it in the “Create Class” window. An example is shown in the circled region in Figure 3.17.

White-space-character check is made for “Class name” and “Class member” strings. The name of the class that is being created is checked against the names of all the classes defined so far for that network topology so that there will not be a redefined class.

The policy creator can see classes by choosing the “View Classes” menu item under the “View” menu. The display of classes is updated as the user adds or deletes classes. Figure 3.18 shows how the class created in Figure 3.17 was added to the

previously created classes and then displayed to the user when the “View Classes” menu item was selected. Class definition statements are in the exact format as they are in a PPL policy file.

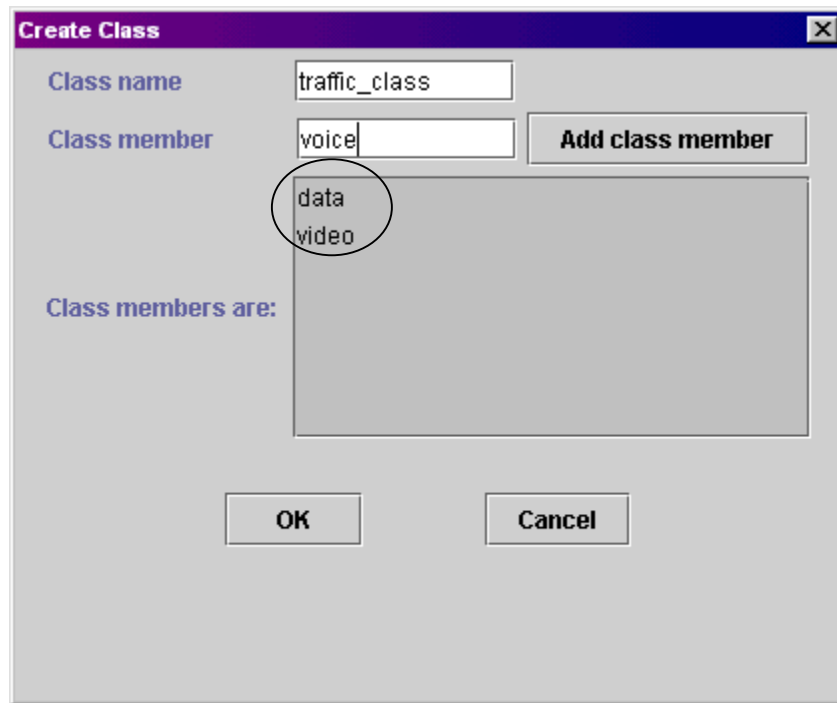


Figure 3.17 Class Creation Window

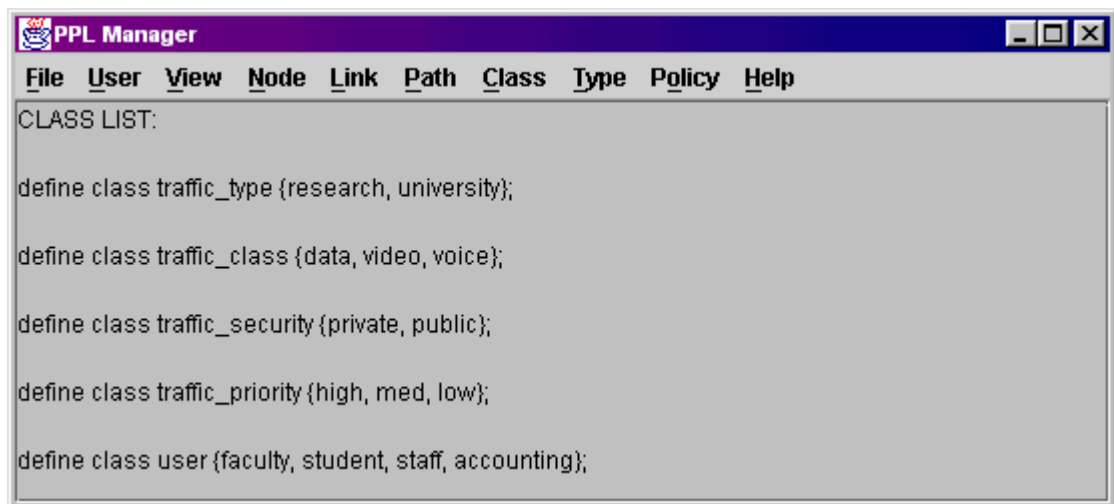


Figure 3.18 Class Display

8. Class Deletion

The policy creator can delete a class by selecting the “Delete class” menu item under the “Class” menu. Then the “Delete Class” modal window, shown in Figure 3.19, is displayed. In this window, the names of all the classes are presented to the user in a combo box. After selecting the class to be deleted in this combo box, the “Delete” button is clicked to delete that class.

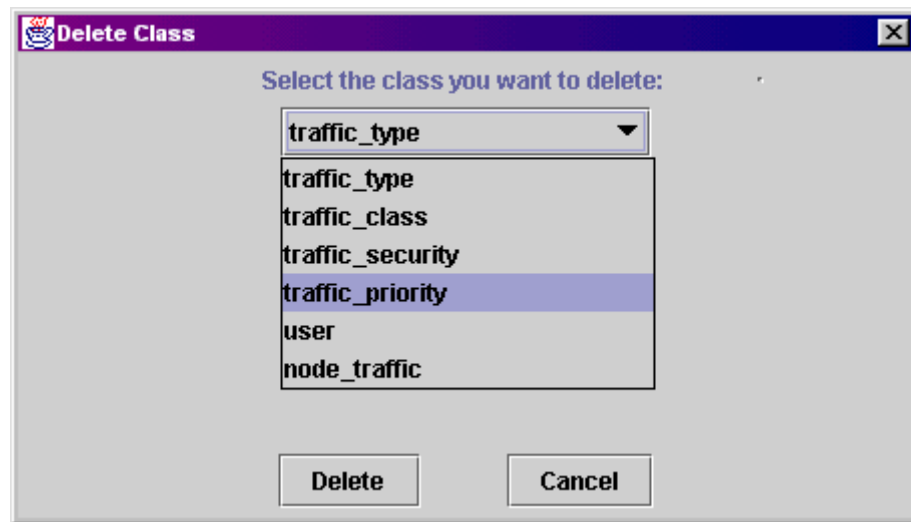


Figure 3.19 Class Deletion Window

9. Type Creation

Types are user-defined statements that can be used in the *Path Condition* element of policy rules. Each type has at least one type member.

The policy maker defines the types and the type members from the “Type” menu. When the policy maker chooses the “Create type” menu item under the “Type” menu, a modal window is displayed, as shown in Figure 3.20. In this window, the “Type Name” must be entered. A check is made to ensure a type has at least one member. Another check is made to verify no type members are the same.

Type members are written to a text field and added to the type by clicking on “Add type member” button. After a member is added, the user can see it in the “Create

Type” window. An example of the result of this displaying process is shown in the circled region of Figure 3.20.

A white-space-character check is made for “Type name” and “Type member” strings as well as a check to verify that the name of the type created is different than the names of all the types defined previously.

The policy maker can view types by choosing the “View Types” menu item under the “View” menu. The display of the types is updated as the user adds or deletes types. Figure 3.21 shows how the type created in Figure 3.20 is displayed to the user when the “View Types” menu item is selected. Type definition statements are presented in the same format as in a PPL policy file.

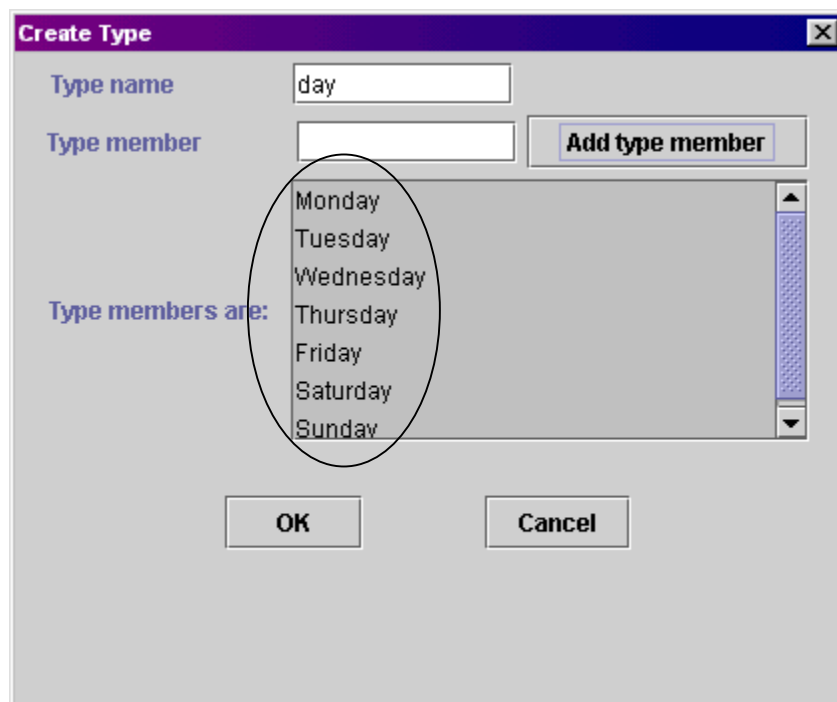


Figure 3.20 Type Creation Window

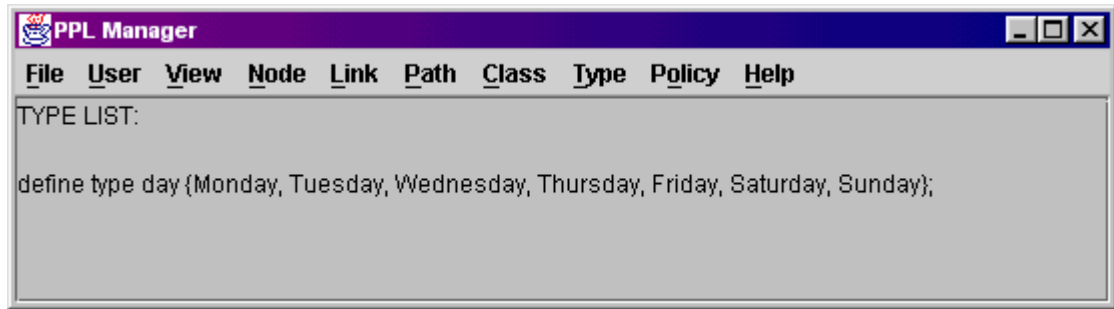


Figure 3.21 Type Display

10. Type Deletion

The policy creator can delete a type by selecting the “Delete type” menu item under the “Type” menu. Then the “Delete Type” modal window, which is shown in Figure 3.22, is displayed. In this window, the names of all the types are presented to the user in a combo box. After selecting the type to be deleted, the “Delete” button is clicked to delete that type.

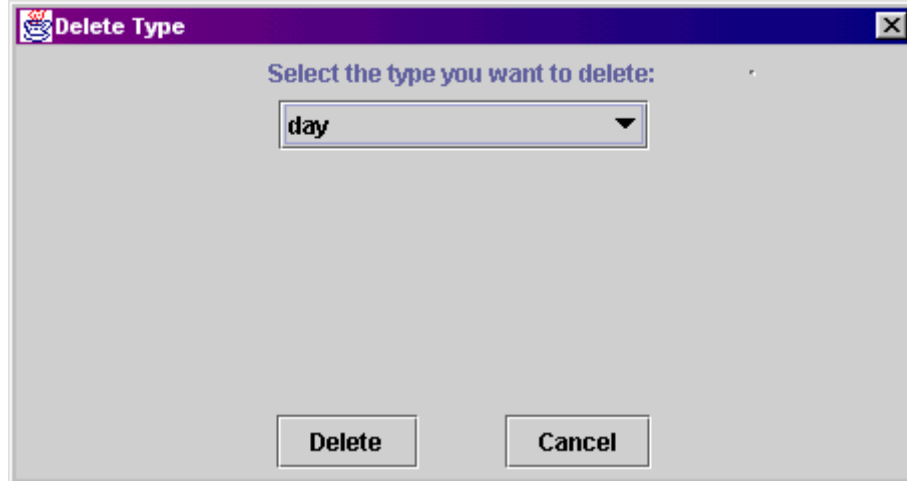


Figure 3.22 Type Deletion Window

D. POLICY DEFINITION

1. Policy Creation

The policy maker defines policies from the “Policy” menu. When the user selects the “Create Policy” menu item under the “Policy” menu, a modal window is displayed, as shown in Figure 3.23.

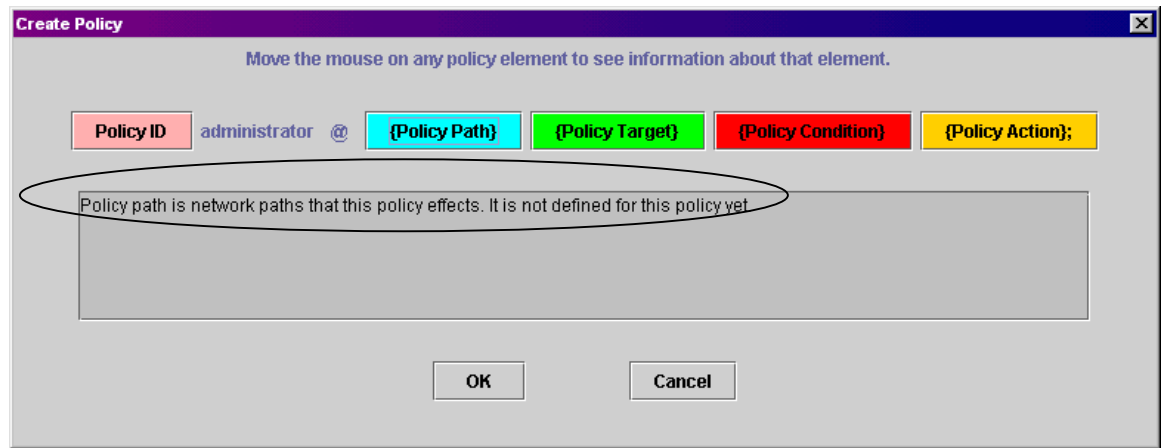


Figure 3.23 Policy Creation Window

There are five buttons in the policy creation window, each representing the fields of a policy that the policy maker has to define. These fields are “Policy ID”, “Policy Path”, “Policy Target”, “Policy Condition” and “Policy Action”. The login ID of the user, “administrator” for figure 3.23, is automatically assigned to the *Policy Creator* element of the policy. When the user moves the mouse over a button, brief information about the element associated with the button is displayed in the text area if that element is not defined yet. The circled region in Figure 3.23 indicates the information displayed in the text area when the user moves the mouse over the “Policy Path” button before that element is defined. If the user has already defined an element, then that definition statement is displayed in the same text area. The statement displayed when the user moves the mouse over the “Policy Path” button after its element is defined is shown in the circled region in Figure 3.24.

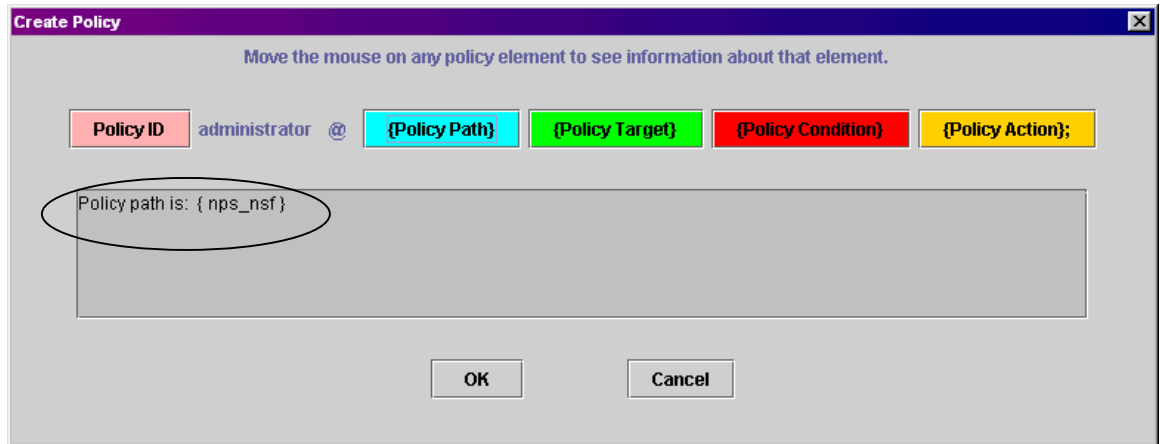


Figure 3.24 Policy Path Definition Display

The user can change contents of a field by clicking on the related button in the “Policy Creation” window at any time, even after it is defined.

After the user defines one of the five policy elements in a new modal window, that new window will disappear and the “Create Policy” window (Figure 3.24) will have the focus again.

If any of the policy elements is left undefined, the GUI provides the policy maker a warning. Incomplete policies are not allowed. Once all five elements of a policy rule are defined, then that policy rule is added to the policy repository by clicking the “OK” button in Figure 3.24.

The policy creator can see the all created policies by choosing the “View Policies” menu item under the “View” menu. The display is updated as the user adds or deletes policies. Figure 3.25 illustrates how policies are displayed to the user when “View Policies” menu item is selected. Policy definition statements are in the same format as in a PPL policy file.

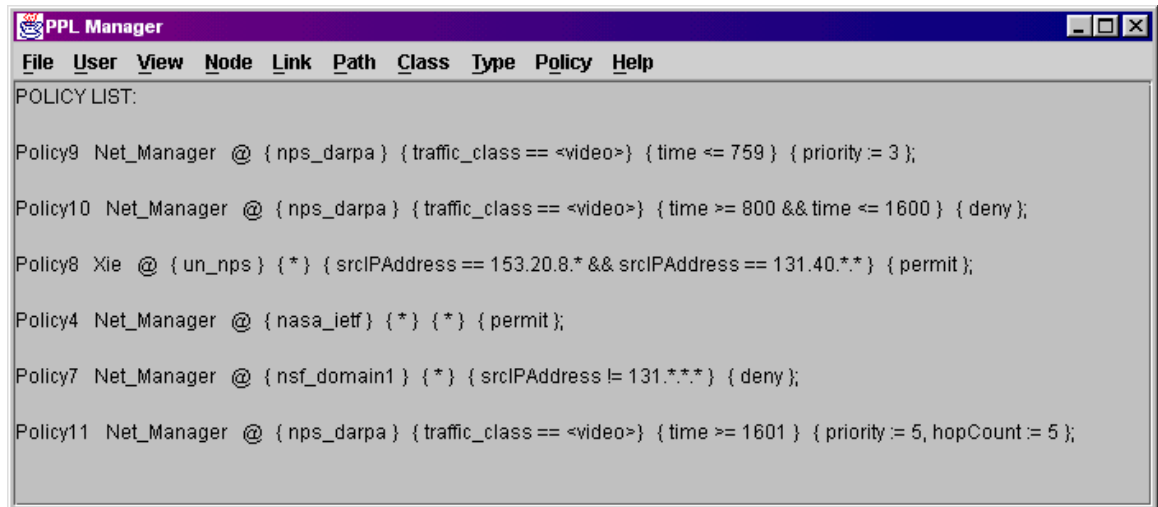


Figure 3.25 Policy Display

Each policy element is explained in detail in the following sections.

a. Policy ID

When the policy maker clicks on the “Policy ID” button in the “Create Policy” window (Figure 3.23) to define the ID of the policy, a modal window, as shown in Figure 3.26, is displayed. The user writes the Policy ID in a text field and clicks on “OK” button.

The policy maker is not allowed to create a policy ID string which does not have a letter as the first character or which has any white space character in it.

Policy ID must be unique and is checked against all existing policy IDs to ensure that there will not be a redefined policy.

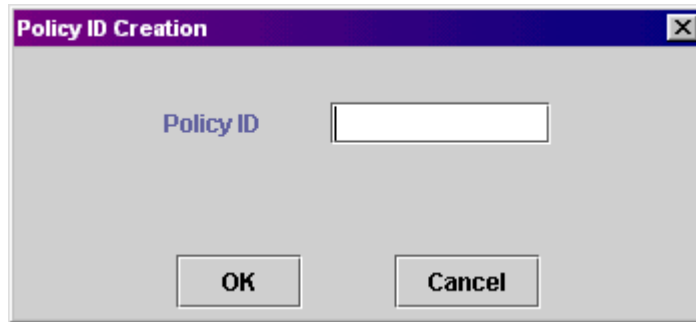


Figure 3.26 Policy ID Creation Window

b. Policy Path

The policy creator defines the path of a policy by clicking on the “Policy Path” button in the “Policy Creation” window (Figure 3.23). Then a modal window, which is depicted in Figure 3.27, is displayed. All the nodes, links, and paths created for the network topology are presented separately in three different combo boxes for the user to select for inclusion in a policy path. A policy path may include any number of nodes, links and paths.

The same node, link, or path cannot be included more than once in a policy path. Otherwise, that policy path would have a loop. As the policy creator adds more nodes, links or paths to a policy path by clicking the “Add selected node,” “Add selected link,” or “Add selected path” buttons, all selected network elements are displayed to the user, in the order they were selected, in the display area of the “Policy Path Creation” window. An example is shown in circled area of Figure 3.27.

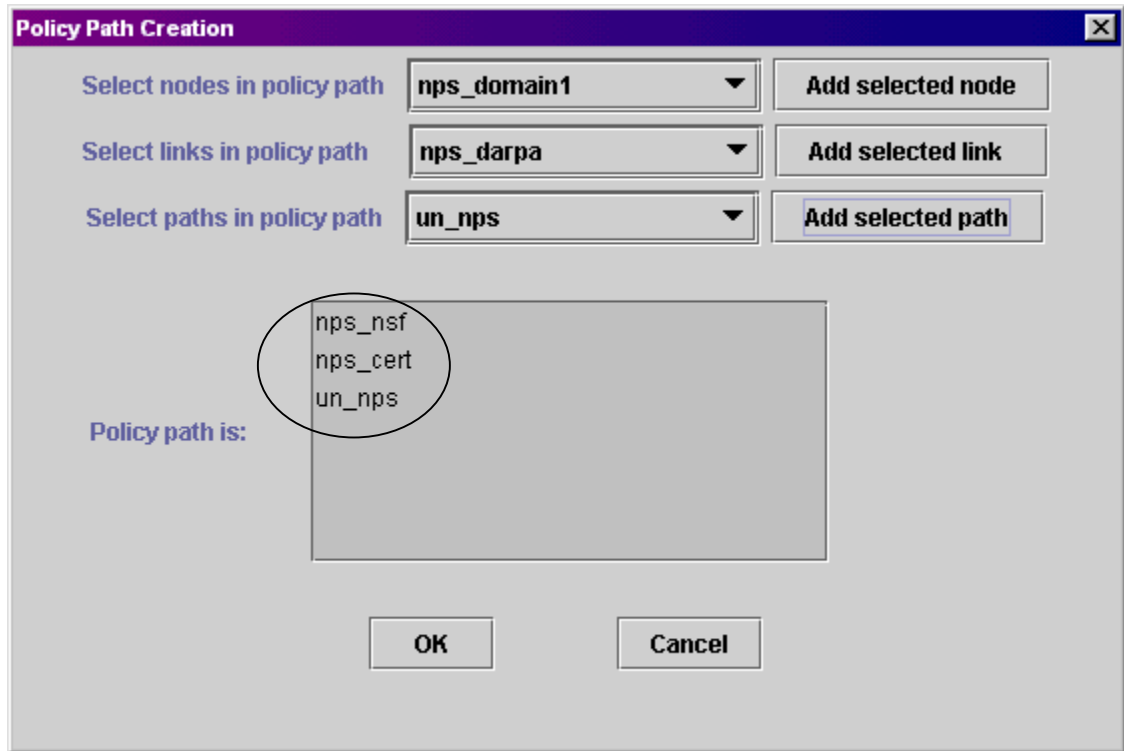


Figure 3.27 Policy Path Creation Window

c. Policy Target

When the policy maker clicks on the “Policy Target” button in the “Create Policy” window (Figure 3.23) to define the *Target Traffic* element of a policy, a new window, shown in Figure 3.28, is displayed.

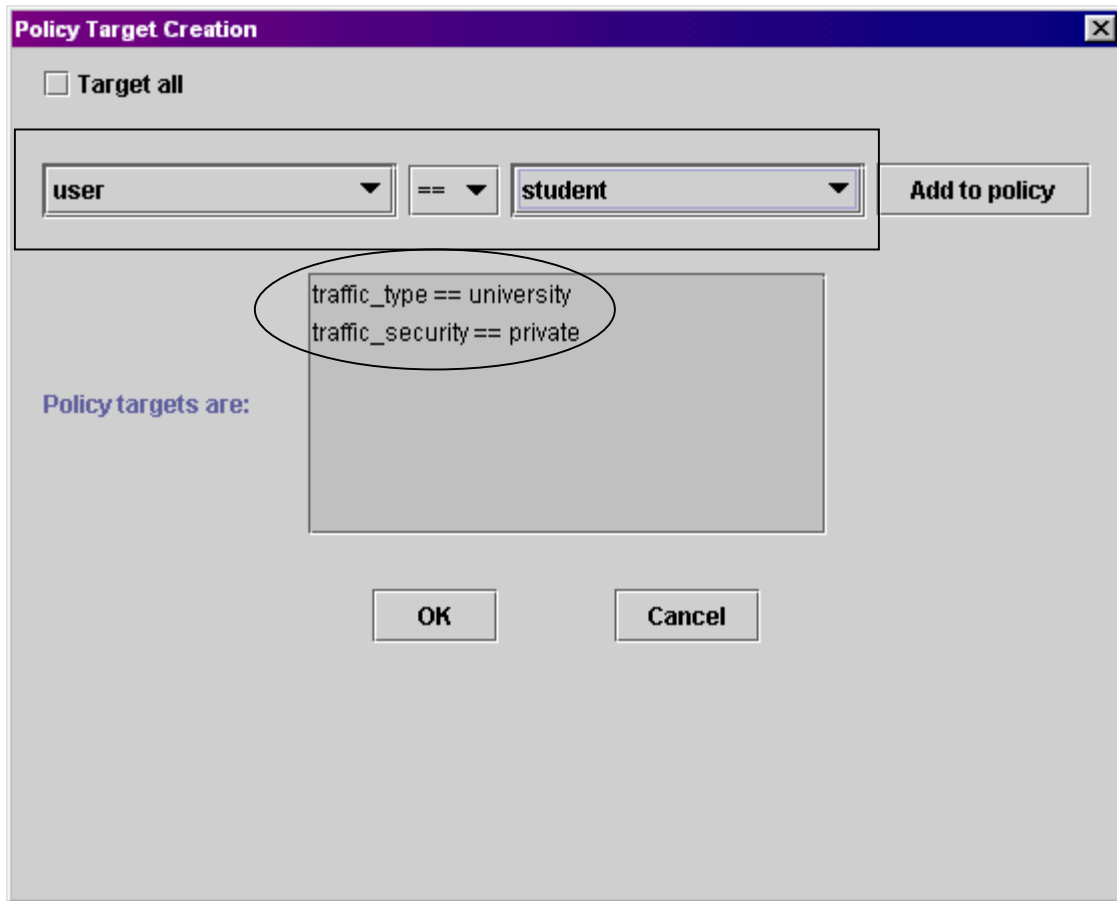


Figure 3.28 Policy Target Creation Window

The *Target Traffic* element of a policy may either be a wild card character or may include one or more classes. There is a “*Target all*” checkbox at the top of the “Policy Target Creation” window. If the user checks it, then a wild card character will appear in the *Target Traffic* element of a policy rule and the “Add to policy” button will be disabled so that the policy maker will not be able to add any class to this part.

The policy creator adds a class to the *Target Traffic* element of a policy by selecting the class name, equation operator, and the class member name from three adjacent combo boxes as shown in rectangle in Figure 3.28. The class member combo box is updated as the user changes classes so that it will only display the members of the class selected in the first combo box. After making the desired selections the “Add to policy” button is clicked to add the class to the target element.

The policy maker is not allowed to add the same class, equation operator, and class member more than one time, since that would create a duplicate target statement.

All policy targets are displayed to the user, as in the region circled in Figure 3.28, as they are added to the target element.

d. Policy Condition

When the policy maker clicks on the “Policy Condition” button in the “Create Policy” window (Figure 3.23) to define the *Path Conditions* element of the policy, a new window, shown in Figure 3.29, is displayed.

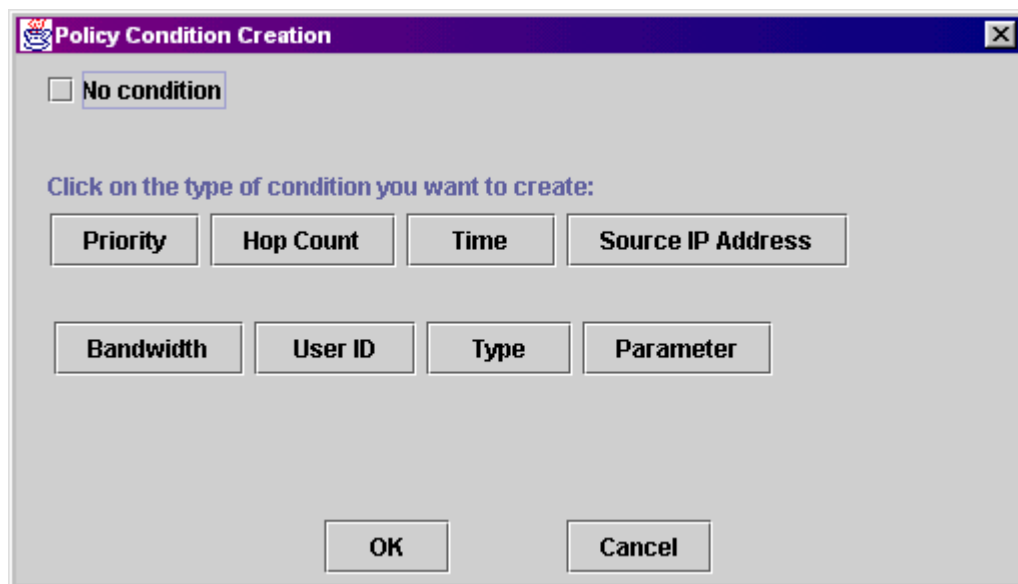


Figure 3.29 Policy Condition Creation Window

A policy condition can either be a wild card character, which means “no condition”, or one or more of the eight conditions. These eight conditions are *Priority*, *Hop Count*, *Time*, *Source IP Address*, *Bandwidth*, *User ID*, *Type* and *Parameter*. There is a *No Condition* check box at the top of the “Policy Condition Creation” window. If the

user checks it, then a wild card character will appear in the *Path Conditions* element of a policy rule and the policy maker will not be able to add any other condition to this part.

The policy creator adds one of eight conditions by clicking on the button upon which the condition name is written. When the user does so, a new modal window is displayed for the definition of the condition. After the user defines the condition and clicks on the “OK” button, that window will disappear and “Policy Condition Creation Window” in Figure 3.29 will have the focus again. The policy maker can add as many conditions as necessary and change a condition at any time by clicking on the related condition button in Figure 3.29. After all the conditions are added, the user will click on the “OK” button to finish the condition generation and to return to the “Policy Creation Window” shown in Figure 3.23.

The details of the condition definitions are explained below.

(1) Priority condition definition

Priority conditions are created by using the GUI in Figure 3.30.

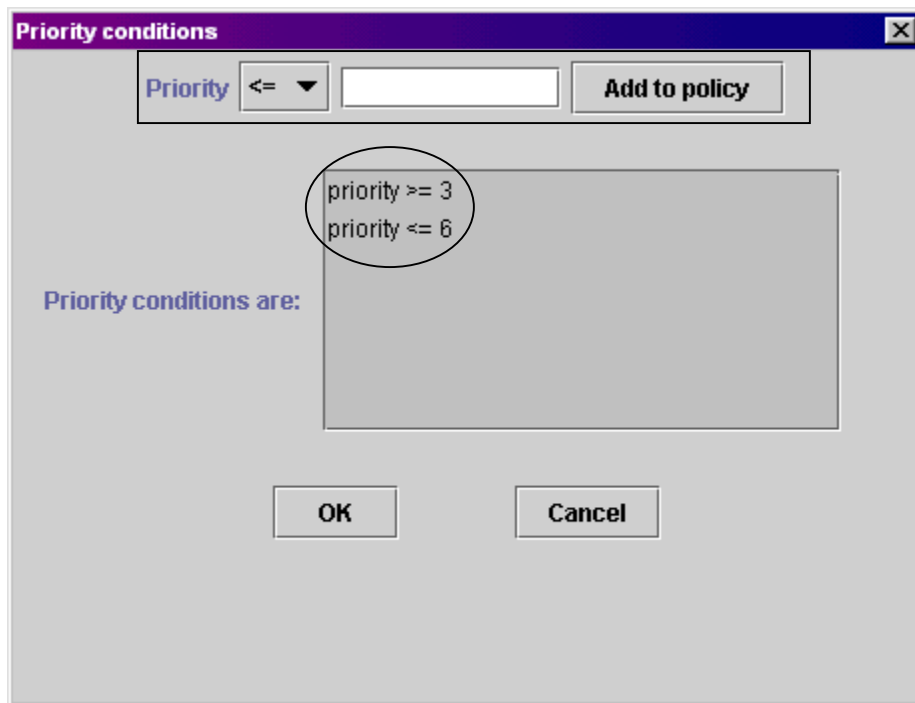


Figure 3.30 Priority Condition Creation Window

The policy maker can add a priority condition by selecting an equation operator from the check box, writing the priority value in the text field and clicking “Add to policy” button, which are delineated in the rectangle in Figure 3.30.

The number that the user enters as a priority value is checked to verify that it is an integer greater than one. A check to prevent duplicate priority conditions is also made.

All priority conditions are displayed to the user, as in the area circled in Figure 3.30, after they are defined.

(2) Hop count condition definition

Hop count conditions are created by using the GUI in Figure 3.31.

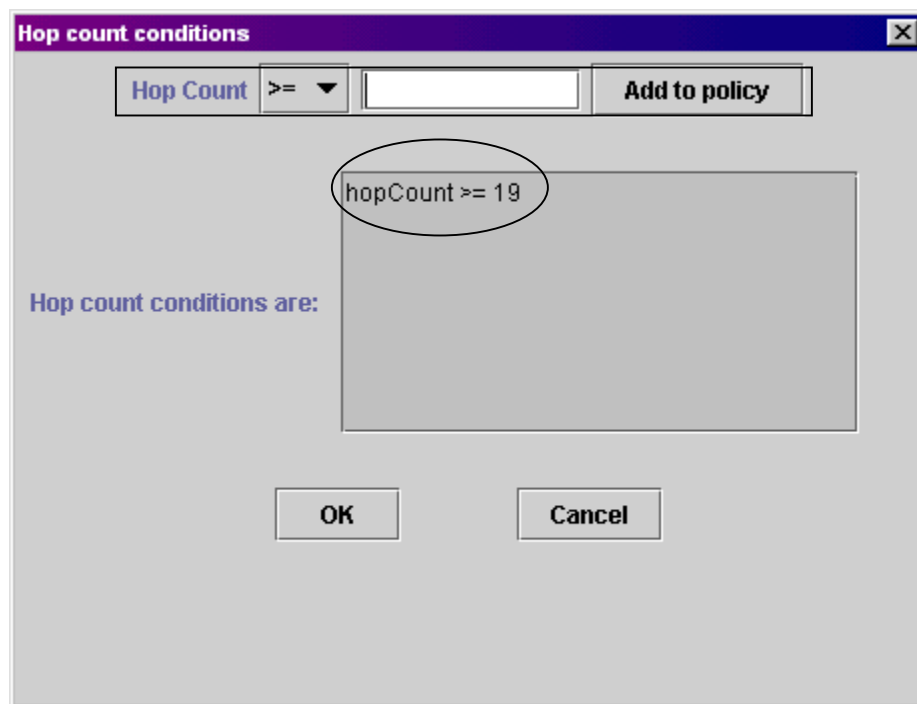


Figure 3.31 Hop Count Condition Creation Window

The policy maker can add a hop count condition by selecting an equation operator from the check box, writing the hop count value in the text field and clicking the “Add to policy” button, which are shown in the rectangle in Figure 3.31.

The number that the user enters as a hop count value is checked to ensure that it is an integer greater than one. A check to prevent duplicate hop count conditions is also made.

All hop count conditions are displayed to the user as in the circled area in Figure 3.31.

(3) Time condition definition

The GUI may be used to create time conditions, as in Figure 3.32.

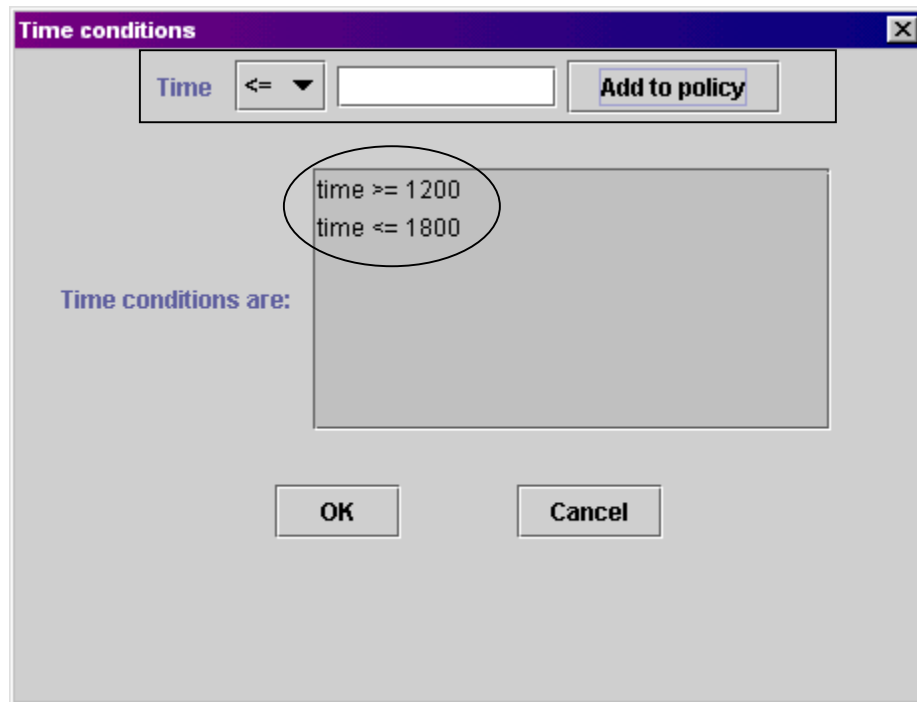


Figure 3.32 Time Condition Creation Window

The policy maker can add a time condition by selecting an equation operator from the list box, writing the time in the text field and clicking the “Add to policy” button, as shown in the rectangle in Figure 3.32.

The time that the user enters is checked so that it will be between 0000-2359 and does not duplicate other time conditions.

All time conditions are displayed to the user in the circled region of Figure 3.32 after they are defined.

(4) Source IP Address Condition Definition

Source IP Address conditions are created by using the GUI in Figure 3.33.

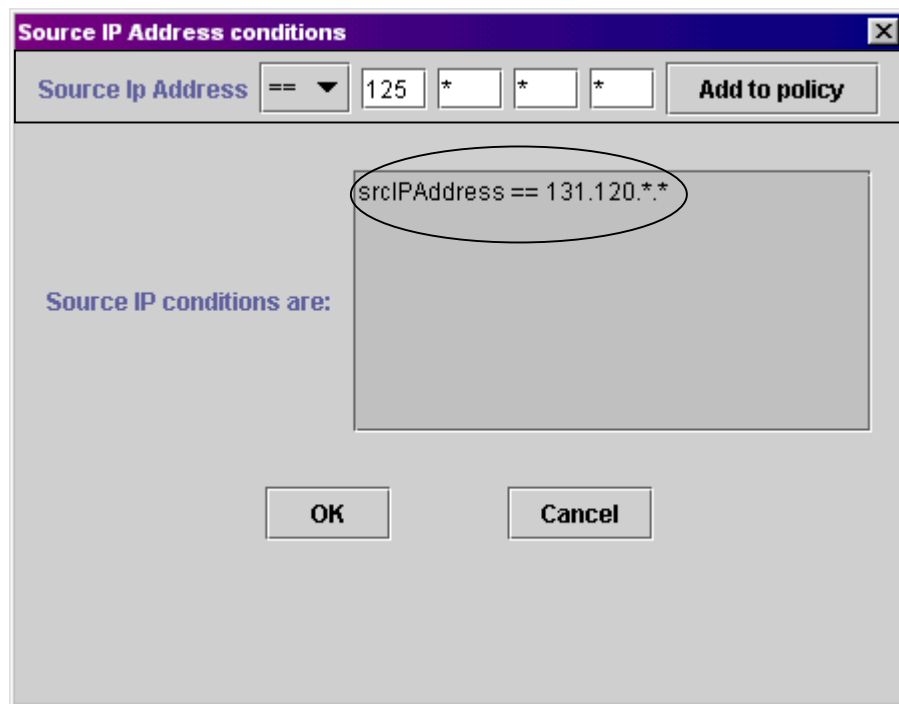


Figure 3.33 Source IP Address Condition Creation Window

The policy creator can add a source IP address condition by selecting the appropriate equation operator from the check box, writing the IP address in the text fields and clicking the “Add to policy” button, as shown in the rectangle in Figure 3.33.

The numbers that the user enters as an IP Address in each one of the four text fields are checked so that they will be either a wild card character or an integer between 0 and 255. Another check to prevent duplicate source IP address conditions is also made.

All source IP address conditions are displayed to the user in the area circled in Figure 3.33 after they are defined.

(5) Bandwidth condition definition

Bandwidth conditions are created by using the GUI shown in Figure 3.34.

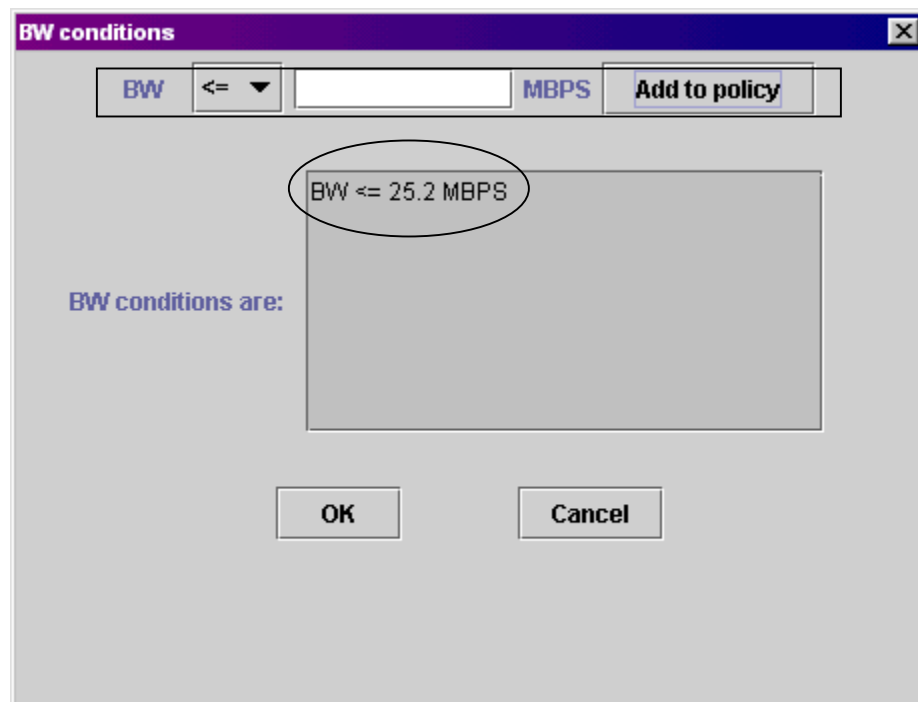


Figure 3.34 Bandwidth Condition Creation Window

The policy creator can add a bandwidth condition by selecting an equation operator from the list box, writing the bandwidth value in the text field and clicking the “Add to policy” button, as shown in the rectangle in Figure 3.34. The number that the user enters as a bandwidth value is checked to make sure it is a positive real number and that no duplicate bandwidth conditions is made.

All bandwidth conditions are displayed to the user in the circled area of Figure 3.34 after they are defined.

(6) User ID condition definition

User ID conditions are created by using the GUI shown in Figure 3.35.

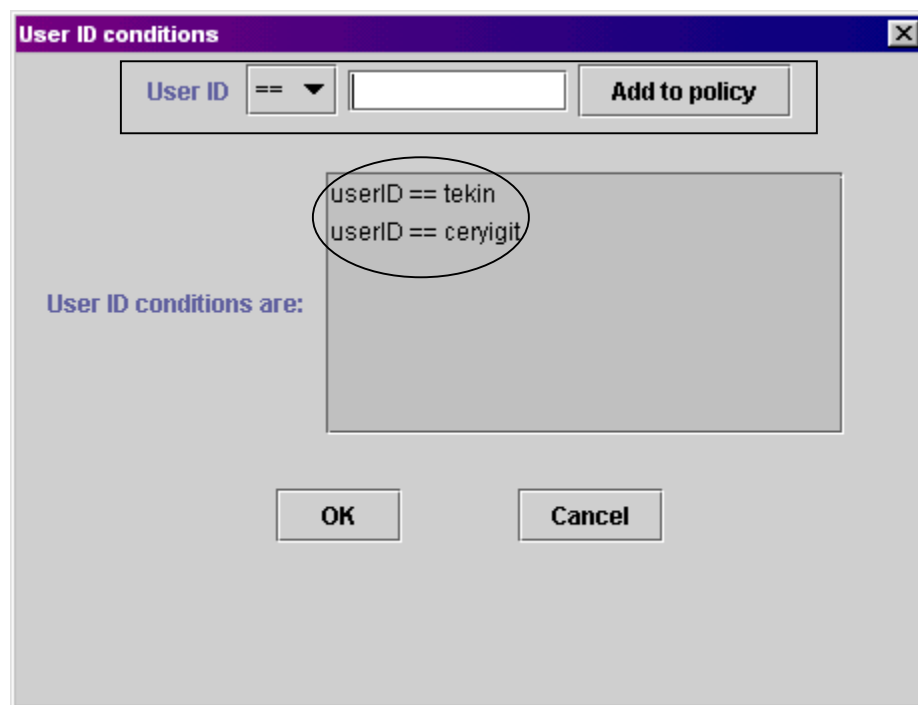


Figure 3.35 User ID Condition Creation Window

The policy creator can add a user ID condition by selecting an equation operator from the list box, writing the ID of the user in the text field and

clicking the “Add to policy” button. These GUI elements are delineated by the rectangle in Figure 3.35.

The user ID string the user enters is checked to verify it contains no white space character nor duplicates any other user ID condition.

All user ID conditions are displayed to the user as shown in the circled area in Figure 3.35.

(7) Type condition definition

Type conditions are created by using the GUI in Figure 3.36.

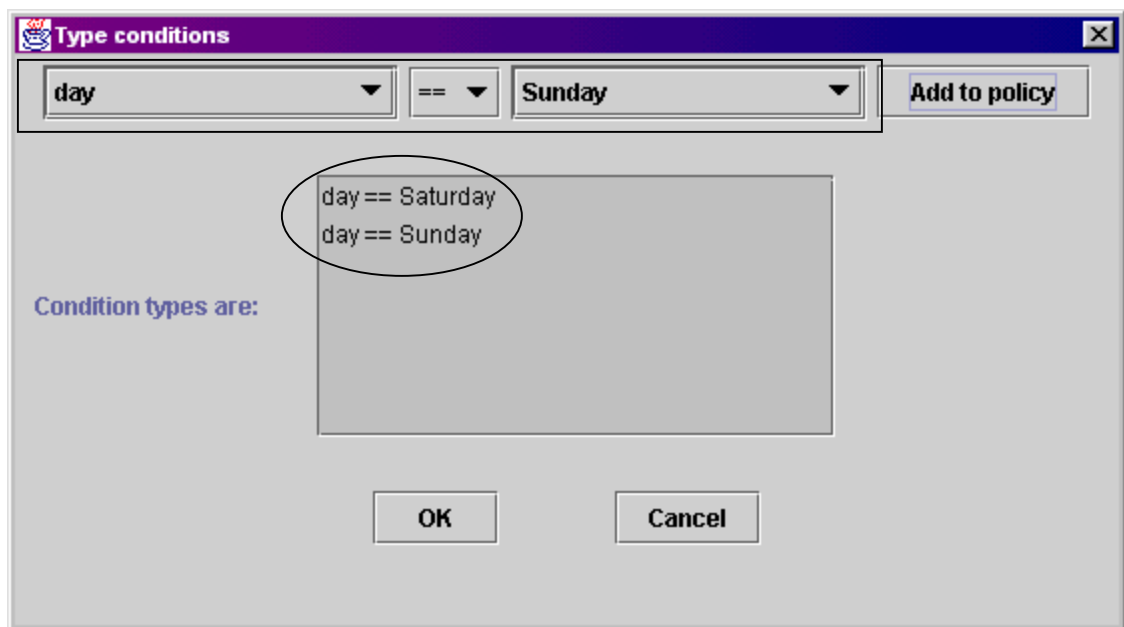


Figure 3.36 Type Condition Creation Window

A type is a user-defined statement specifically designed for use in the Path Conditions element of a PPL policy rule. If at least one type statement is defined before the creation of a policy rule, then the policy creator can add a type condition by selecting the type name, an equation operator, and the type member name from the three adjacent combo boxes shown in the rectangle in Figure 3.36. The type-member combo

box is updated as the user changes types in the type combo box so that it will only display the members of the type selected in the type combo box. After making the desired selections, the user clicks the “Add to policy” button to add the type item to the policy condition. If the user did not define any type statements before creating a policy, then the “Type” button in the “Policy Condition Creation Window” in Figure 3.29 will be disabled so the user will not be able to define a type condition.

The policy maker is not allowed to add the same type, equation operator, and type member more than one time, since it will create a duplicate type condition.

All type conditions are displayed to the user in the area circled of Figure 3.36 as they are added to policy condition.

(8) Parameter condition definition

Parameter conditions are created by using the GUI in Figure 3.37.

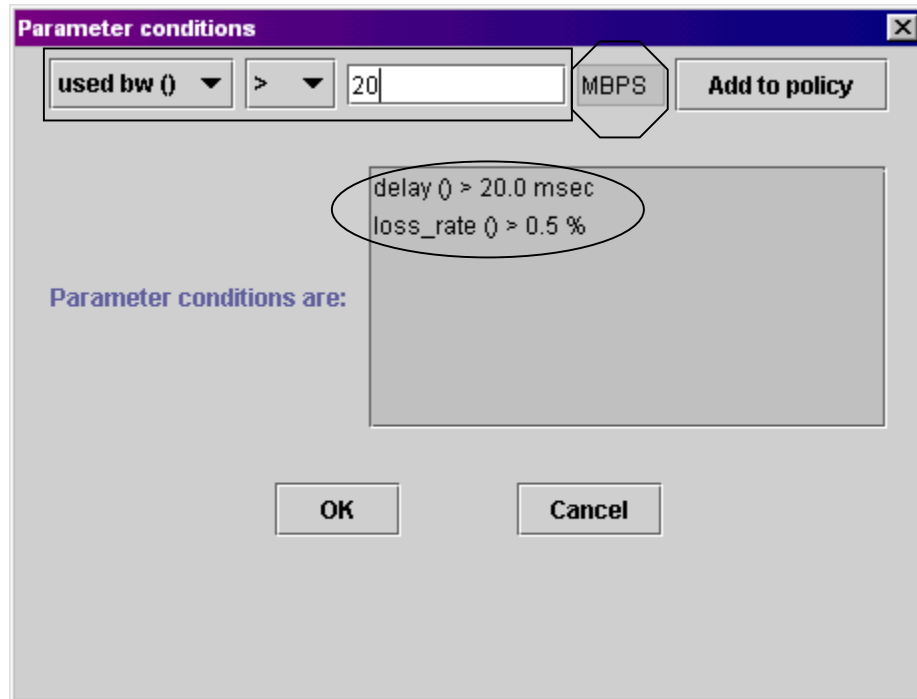


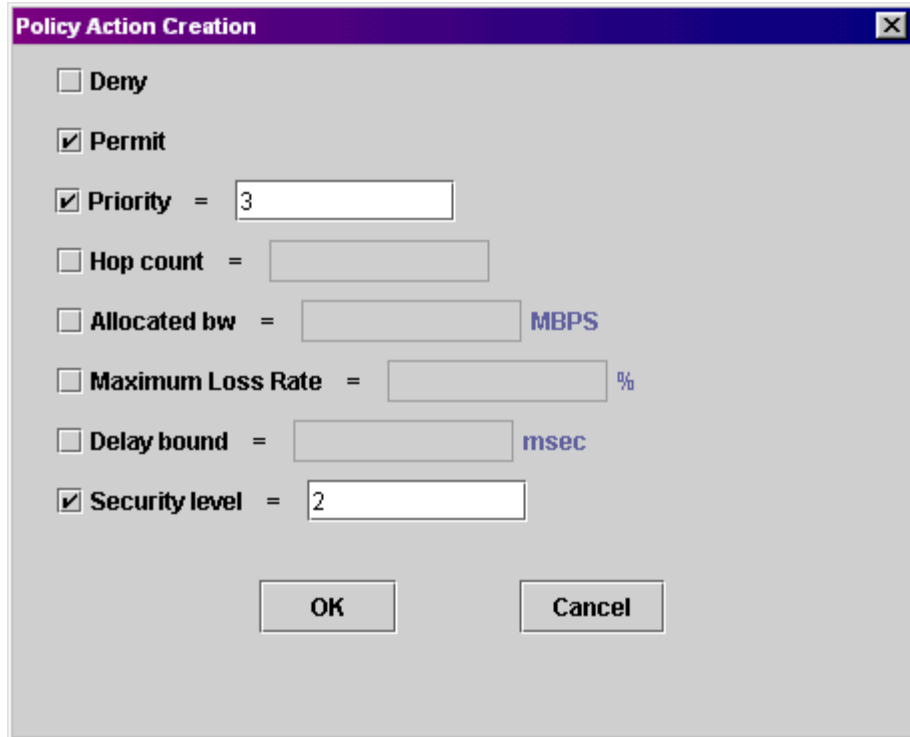
Figure 3.37 Parameter Condition Creation Window

Parameter conditions can only be defined when a node, link or path of a network topology has one or more *delay*, *jitter*, *loss rate*, or *used bandwidth* parameters defined. The policy creator adds a parameter condition by selecting the parameter name and an equation operator from the two adjacent combo boxes and writing the parameter value in the text field, as shown in rectangle in Figure 3.37. The parameter unit label, shown in the octagon in Figure 3.37, is updated as the user changes parameters in the parameter combo box so that it will display the correct unit. The unit for the *delay* and *jitter* parameters is milliseconds (msec). The unit for *loss rate* parameter is percent (%) and the unit for *used bandwidth* parameter is megabits per second (MBPS). After making the desired selections, the “Add to policy” button is clicked to add the parameter condition. If the user did not define any parameters for any node, link or path before creating a policy, then the “Parameter” button in the “Policy Condition Creation Window” in Figure 3.29 will be disabled so that the user will not be able to define a parameter condition.

The policy maker is not allowed to add the same parameter, equation operator, and parameter value more than one time, since doing so would create a duplicate parameter condition. Delay, jitter, and used bandwidth parameter values are checked to verify they are positive numbers. The loss rate parameter value is checked to ensure it contains a number between 0 and 100. All parameter conditions are displayed to the user in the area circled in Figure 3.37 as they are added to policy condition.

e. Policy Action

When the policy maker clicks on the “Policy Action” button in the “Create Policy” window (Figure 3.23) to define the *Action Items* element of the policy, a new modal window, as shown in Figure 3.38, is displayed.



The image shows a 'Policy Action Creation' dialog box with a purple title bar. It contains several options and input fields:

- ☐ Deny
- ☒ Permit
- ☒ Priority =
- ☐ Hop count =
- ☐ Allocated bw = MBPS
- ☐ Maximum Loss Rate = %
- ☐ Delay bound = msec
- ☒ Security level =

At the bottom are 'OK' and 'Cancel' buttons.

Figure 3.38 Policy Action Creation Window

There are eight actions that can be used in the action part of a policy rule. These are *deny*, *permit*, *priority*, *hop count*, *allocated bandwidth*, *maximum loss rate*, *delay bound*, and *security level*. The policy creator can add *deny* or *permit* actions by selecting the “deny” or “permit” check box, as appropriate. If the user selects the *deny* action, then no other action can be added to the policy action part. Thus, if the *deny* check box is selected, then all other check boxes are disabled to prevent the user adding any other action to the policy. If the policy maker selects the *permit* box, one or more of six parameter values may be combined in a single action statement. These parameter values should be interpreted uniformly across different domains in the implementation of the PPL. For example, *securityLevel:=2* action item of a policy should mean that the flow has the second highest security classification level in all the domains of the network. The parameter value text fields are disabled when the “Policy Action Creation Window” is first displayed. After one of the parameter value check boxes is selected, then the corresponding text field for that parameter value is enabled and the user enters the parameter value in the text field. Once the user selects a parameter check box, then the

corresponding text field in which parameter value is written may not be left empty. For Figure 3.38, the policy creator selected permit, and assigned priority value 3 and security level value 2 to the target traffic of the policy.

2. Policy Modification

Policy modification is very similar to policy creation, although there are two differences between these processes. The first difference is that the first step in policy modification is selecting the policy to be modified, as shown in Figure 3.39. As all the elements of the policy to be modified have already been defined, the second difference is that defined statements are displayed to the user when the mouse is moved over one of the policy elements so that the user will be able to see the elements that may be changed, as shown in Figure 3.40.

The policy maker modifies a policy by selecting the “Modify policy” menu item under the “Policy” menu. A new modal window, which is depicted in Figure 3.39, is displayed when the “Modify policy” menu item is selected. The user selects the *Policy ID* of the policy to be modified from the combo box that presents all the *Policy IDs* and then clicks on the “Modify” button.

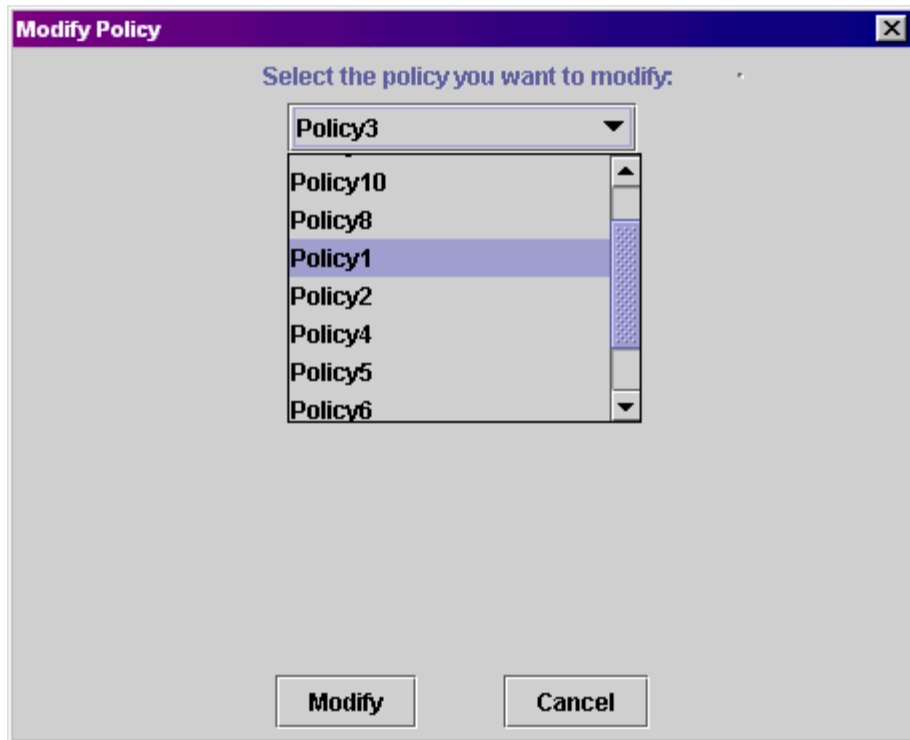


Figure 3.39 Policy Modification Window – 1

A window similar to the “Policy Creation Window” shown in Figure 3.23 is displayed as the second step of the policy modification process. In this window, when the mouse is moved over an element, that element of the policy is displayed to the user. An example display of the *Path Conditions* element of a particular policy is illustrated in Figure 3.40. This snapshot illustrates the result when the policy maker moves the mouse over the “{Policy Condition}” button to view the condition element. After necessary changes are made to the policy, the “OK” button is clicked to apply the changes.

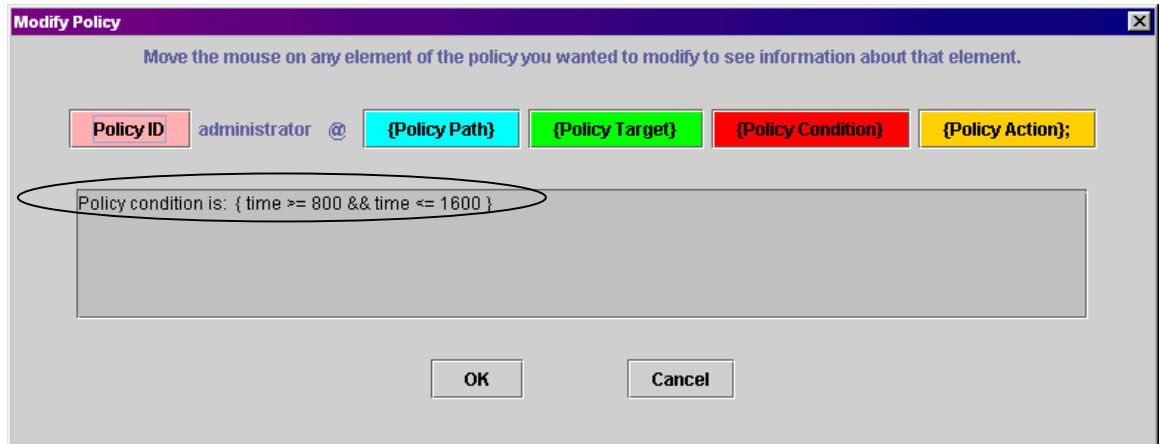


Figure 3.40 Policy Modification Window – 2

3. Policy Deletion

The policy creator can delete a policy by selecting the “Delete policy” menu item under the “Policy” menu. Then the “Delete Policy” modal window, which is shown in Figure 3.41, is displayed. In this window, the *Policy IDs* of all current policies are presented to the user in a combo box. After selecting the *Policy ID* of the policy to be deleted from this combo box, the “Delete” button is clicked to delete that policy.

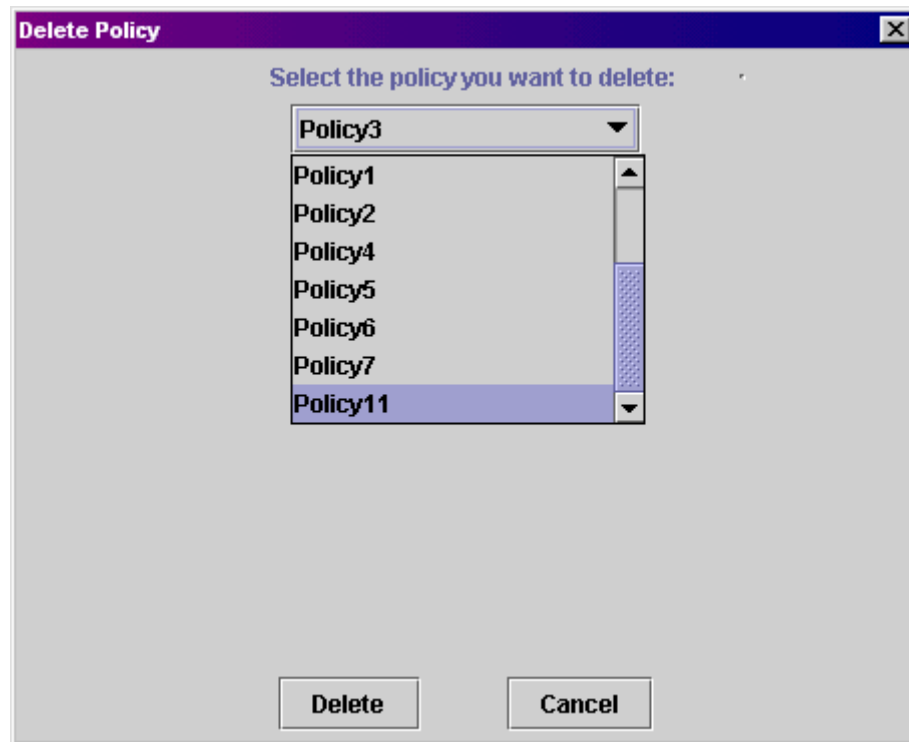


Figure 3.41 Policy Deletion Window

E. FILE MANAGEMENT

File related issues that a policy maker is capable of performing by using the PPL Manager are as follows: creating a new policy file, saving network definition and policies to a file, opening an existing policy file, saving policies, importing policies, setting the compilation mode, and compiling. These are explained in detail in the remainder of this chapter.

1. Creating a New File

Policy makers can create a new policy file when they are using PPL Manager by choosing the “New” menu item under the “File” menu. When this menu item is selected, the window, shown in Figure 3.42, will be displayed.

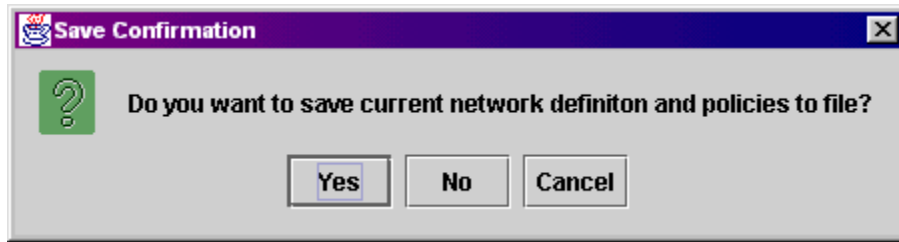


Figure 3.42 Confirmation Window When Creating a New File

The user can save current network definition and policies by clicking the “Yes” button in the “Save Confirmation” window. If the user wants to create a new file without saving the current information to a file, then the “No” button should be clicked. In both cases, the information displayed in the text area of the PPL Manager and the contents of node, link, path, class, type, and policy vectors are all cleared.

2. Saving and Opening a File

a. Saving a File

The policy creator can save nodes, links, paths, classes, types, and policies to a file by selecting “Save File” menu item under “File” menu. A file chooser, which is depicted in Figure 3.43, is displayed after this selection. The user may save current network topology information (i.e., nodes, links, paths, classes and types) and policies or only network topology information without any policy in any directory by entering a file name in the file chooser. The file must be a text file, having the extension “.txt”.

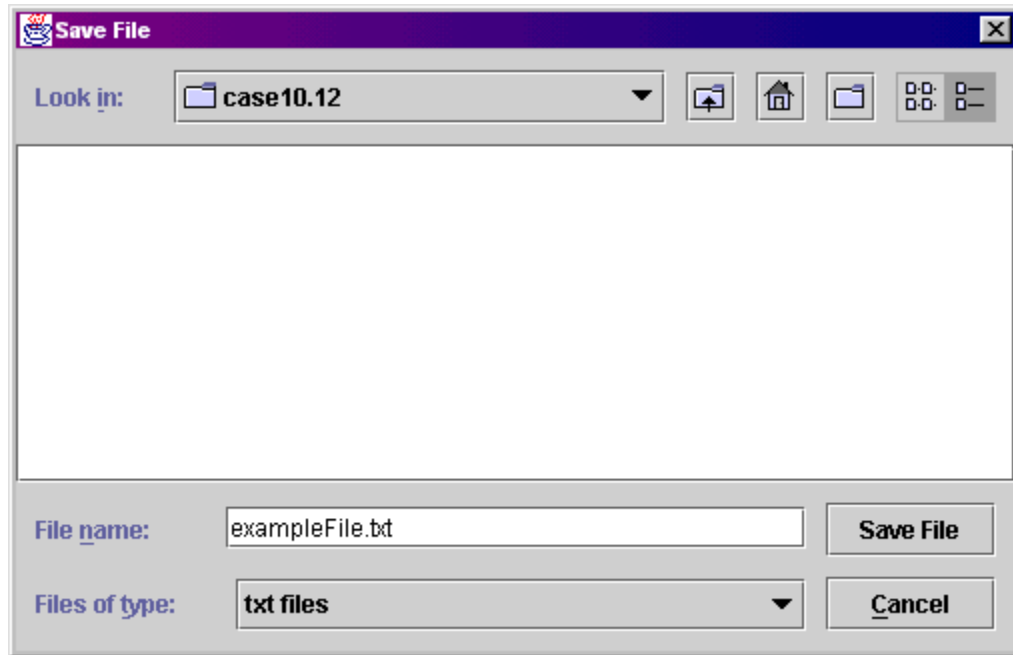


Figure 3.43 File Chooser for Saving Files

After the “Save File” button is clicked in this file chooser, two files are created in the directory the user selected. The first of the files has the exact name that the user gave to the file. It is formed by using the *ObjectOutputStream* class of JAVA and is not in a readable form. This file is used in the “Open File” process. The name of the second file is formed by concatenating the string “PPL” to the front of the file name entered by the user. This file is created automatically in a readable form for reviewing and printing purposes. For example, if the user writes “exampleFile.txt” as file name, the name of the first file would be “exampleFile.txt”; the name of the second file would be “PPLexampleFile.txt”.

b. Opening a File

The policy creator can open a PPL file containing network topology information with policies or only network topology information by selecting the “Open File” menu item under the “File” menu. A file chooser, depicted in Figure 3.44, is displayed after this selection. The user can only see the directories and PPL text files that can be opened in this file chooser. The reason for this filter is to prevent the policy maker

from selecting a non-PPL file. The user can choose to see all files, instead of only PPL files, by selecting the “All Files (*.*)” item from the “Files of type” combo box at the bottom of the file chooser. In this case, if the user selects a file not created with the PPL Manager, a warning message is presented as depicted in Figure 3.45, and the “Open File” process must be repeated if a file is to be opened. After a correct PPL file is selected, the “Open File” button in the file chooser is clicked to open that file.

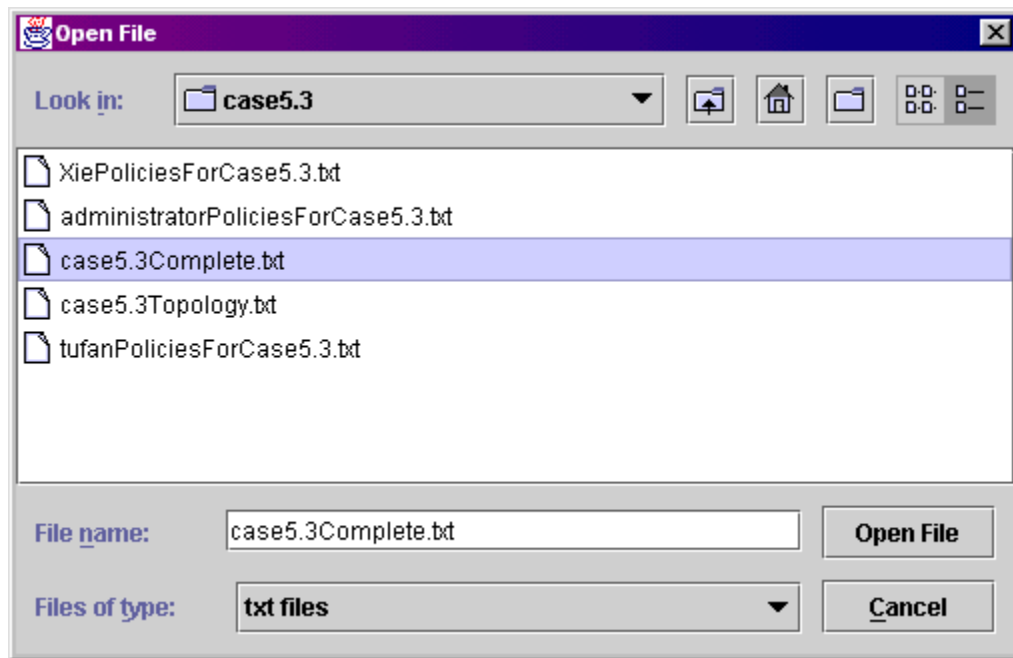


Figure 3.44 File Chooser for Opening Files

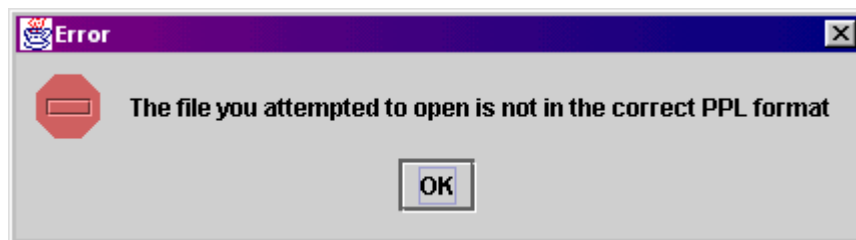


Figure 3.45 Error Message in Opening a File

3. Saving and Importing Policies

Consider the following scenario. At different times, each of three different policy makers, who work for the same organization, opens the file that contains the network topology information for their organization and creates some policies. Each of the users saves their policies in a separate file. Later, the network administrator opens the organization's network topology file and imports the three new policy files. Then he creates his own policies and aggregates all of them into one file. Following the compilation process, the administrator modifies any conflicting policies and re-compiles the file to obtain a conflict-free policy set. This scenario dictates one of the powerful features of PPL Manager: policies of different users for a particular network topology can be aggregated in one file. In order to do that, the policy makers must be able to save policies to a file and import them later on. The method of performing these two tasks is explained in the following two sections.

a. Saving Policies

The policy creator can save policies currently defined in the PPL Manager to a file by selecting the "Save Policy" menu item under the "File" menu. A file chooser is created after this selection as depicted in Figure 3.46. The file must be saved as a text file, with the extension ".txt".

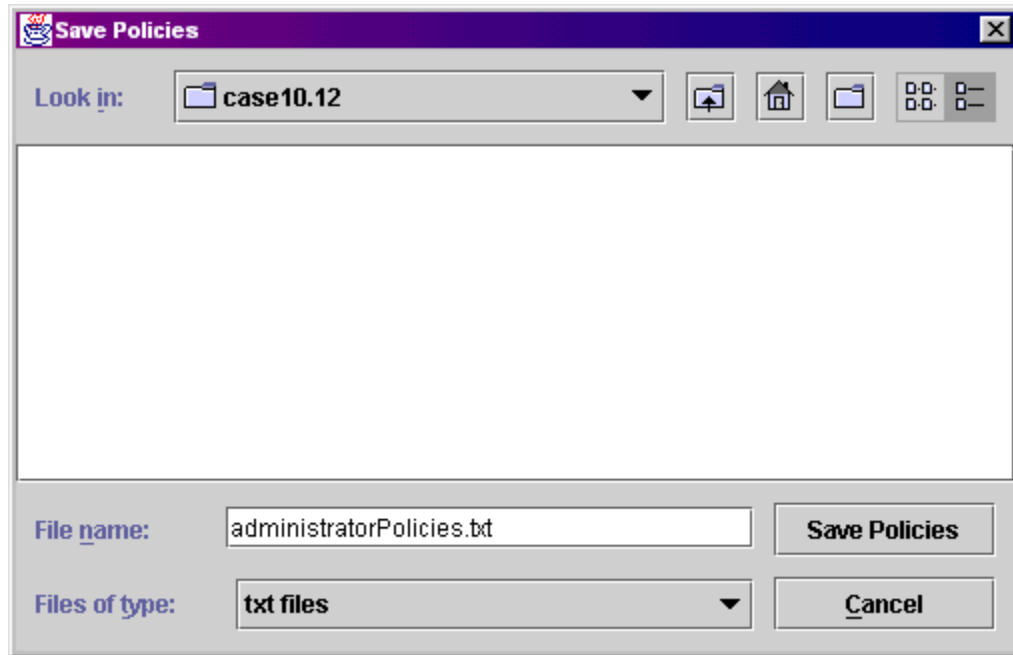


Figure 3.46 File Chooser for Saving Policies

After the “Save Policies” button is clicked in this file chooser, two files are created in the directory the user selected. The first of the files has the exact name that the user gave to the file. It is formed by using the *ObjectOutputStream* class of JAVA and is not in a readable form. This file is used in “Import Policy” process. The name of the second file is formed by concatenating the string “PPL” to the front of the file name entered by the user. This file is created automatically in a readable form for reviewing and printing purposes. For example, if the user writes “administratorPolicies.txt” as file name, the name of the first file would be “administratorPolicies.txt”; the name of the second file would be “PPLadministratorPolicies.txt”.

b. Importing Policies

The policy creator can import PPL policies from a file by selecting the “Import Policy” menu item under the “File” menu. A file chooser, depicted in Figure 3.47, is displayed following this selection. The user can only see the directories and PPL policy files in this file chooser. The reason for this filter is to prevent policy maker from selecting a non-PPL file. The user can also prefer to see all files instead of only PPL files

by selecting “All Files (*.*)” item from the “Files of type” combo box at the bottom of the file chooser. In this case, if the user selects a file not created by PPL Manager to open, a warning message, which is depicted in Figure 3.48, will be presented and “Import Policy” process should be repeated. After a correct PPL file including PPL policies is selected, The “Import Policies” button in the file chooser is clicked to import the policies from the selected file.

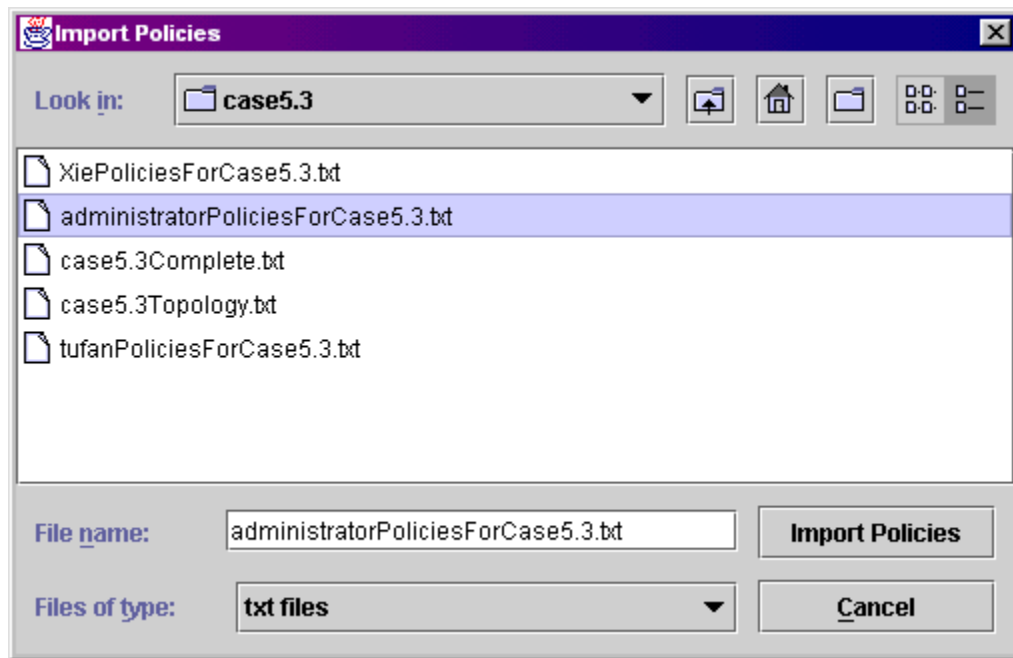


Figure 3.47 File Chooser for Importing Policies

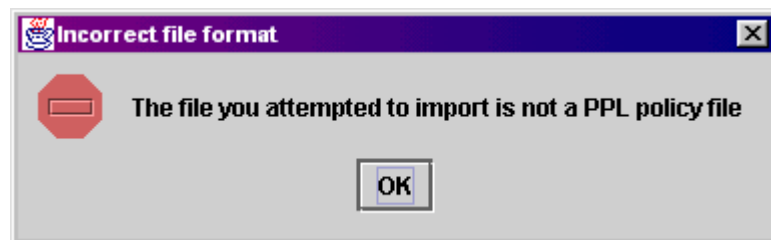


Figure 3.48 Error Message in Importing Policies

4. Compiling

One of the important features of the PPL Manager is that compiling a PPL file is no longer a separate process. Policy makers can compile a PPL file from the “Compile” menu item under the “File” menu. Prior to the development of the PPL Manager, users had to invoke a BASH shell, save network topology and policies to a file, and write the command, “scan filename” to the BASH shell. With the development of the PPL Manager, when the “Compile” menu item is selected, the BASH shell is invoked automatically and the appropriate “scan” command is sent to the BASH shell for the compilation of the current network topology and policies. The user does not have to save current information to a file to be able to compile and see the results.

There are three options that a user may choose when compiling a file. The first option is the “scan” command with no argument, “scan filename”. The second option is the “scan” command with “-no wild” argument, “scan-no_wild filename”. This option does not support the use of wild card characters in paths, as explained in [Ref.2]. The third option is the “scan” command with “-no_implicit_deny” argument, “scan-no_implicit_deny filename”. Stone [Ref.2] states that this option informs the policy tester to ignore conflicts that exist due to implicit denies between policy rules created by the same user. These three options of the “scan” command are presented to the policy maker in a radio button menu item under the “File” menu, as depicted in Figure 3.49. The policy maker selects one of the compilation modes by selecting one of the radio buttons in this menu item.

The results of the compiler output are presented to the user in the text area of the PPL Manager. An example output is shown in Figure 3.50. After compilation, the user will see the conflicts produced by the current set of policies in the “scan_out.txt” file.

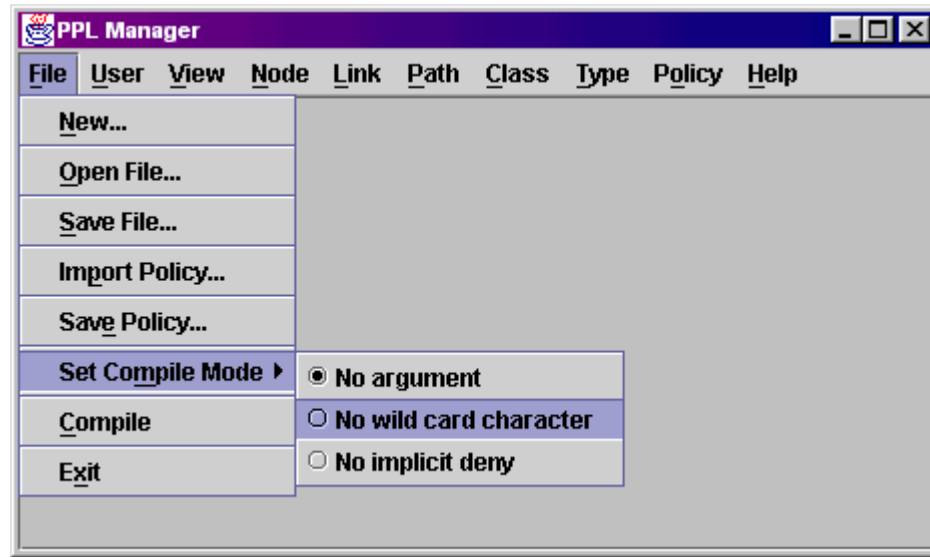


Figure 3.49 Setting Compilation Mode

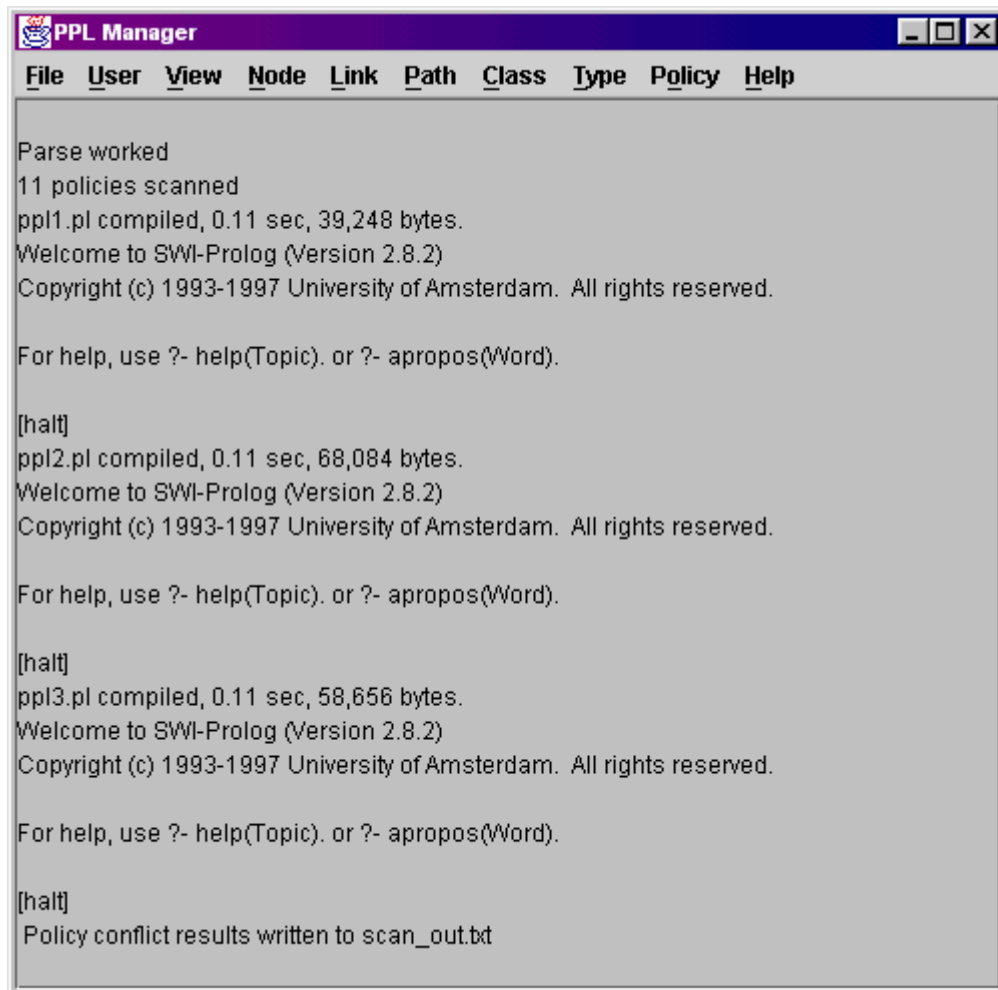


Figure 3.50 Compiler Output

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION

A. SUMMARY

Seven changes were made to the LEXER and YACC code of the PPL to make the language syntax more consistent and to add additional capabilities. A graphical user interface toolkit for creating, validating, archiving, and compiling policies represented in PPL was developed to hide subtle details of the PPL syntax from the user and to combine different processes which formerly required manual intervention in the previous version of PPL. The toolkit also provides security mechanisms for controlling access to the system and maintenance of policy rules.

B. FUTURE WORK

Network policies are created, validated, and stored by means of the PPL Manager. Enforcing these policies in a testbed network is one area of future research. The SAAM [Ref. 8] network is an ideal testbed network for this task because SAAM provides a path-based approach for supporting end-to-end QoS capabilities, paralleling the path-based approach PPL uses to create and enforce network policies. Additional functionality to make the PPL Manager a tool for loading policies into the network policy server might also be added.

More work is required to address security issues. The PPL database needs to be equipped with security mechanisms to protect its confidentiality and integrity.

The expected benefits identified in this thesis need to be substantiated via experiments conducted by potential users of PPL. The first two steps in this research would be to identify metrics and to design the experimental protocol.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. CHANGED VERSION OF PPL SCAN.FLEX FILE

```
/******  
File: scan.flex  
Author: Gary Stone. Changes made by Tufan Ekin are in bold font.  
Course: Thesis (CS 0810)  
Date: August 2001.  
Description: This file is a part of the PPL Compiler code in [Ref.2]  
which defines the tokens (meaningful units) that will be used by PPL  
compiler.  
*****/  
/* DEFINITION SECTION */  
%{  
#include "y.tab.h"  
#include "symbol.h"  
#include <string.h>  
int verbose = 0;  
int lineno = 1;  
%}  
/* RULES SECTION */  
%%  
[ \t]* { /* ignore whitespace */  
  
/*  
* Detect start of comment  
*/  
"/*" { parse_comment();}  
[<>@{}=:,;() ] {return yytext[0]; }  
"*)" {yylval.string = yytext;return WILDCARD;}  
  
/*  
* Reserved Words  
*/  
([dD][eE][nN][yY]) {return DENY;}  
([pP][rR][iI][oO][rR][iI][tT][yY]) {return PRIORITY; }  
([aA][lL][lL][oO][wW]) {return ALLOW;}
```

```
([pP][eE][rR][mM][iI][tT])      {return PERMIT;}
([hH][oO][pP][cC][oO][uU][nN][tT]) {return HOPCOUNT;}
```

/* CHANGE 5-A: hostIP token is changed to srcIPAddress token */

```
([sS][rR][cC][iI][pP][aA][dD][dD][rR][eE][sS][sS])
{return SRCIPADDRESS;}
```

```
([aA][lL][lL][oO][cC][aA][tT][eE][dD][_][bB][wW]){return ALLOCATED_BW;}
```

/* CHANGE 7-A: Allocated_bw, maxLossRate, delayBound and securityLevel tokens are added to be used in the Action Items element of a policy rule*/

```
([mM][aA][xX][lL][oO][sS][sS][rR][aA][tT][eE]) {return MAXLOSSRATE; }
([dD][eE][lL][aA][yY][bB][oO][uU][nN][dD])      { return DELAYBOUND; }
([sS][eE][cC][uU][rR][iI][tT][yY][lL][eE][vV][eE][lL])
{return SECURITYLEVEL; }
([tT][iI][mM][eE])      { return TIME; }
([bB][wW])              { return BW; }
([uU][sS][eE][rR][iI][dD])      { return USERID; }
([dD][eE][fF][iI][nN][eE])      { return DEFINE; }
([cC][lL][aA][sS][sS])          { return CLASS; }
([mM][eE][sS][sS][aA][gG][eE])  { return MESSAGE; }
([tT][yY][pP][eE])            { return TYPE; }
([pP][aA][tT][hH])            { return PATH; }
([nN][oO][dD][eE])            { return NODE; }
([lL][iI][nN][kK])            { return LINK; }
([pP][oO][lL][iI][cC][yY][_][mM][aA][kK][eE][rR]){return POLICY_MAKER; }
([gG][bB][pP][sS])            { return GBPS; }
([mM][bB][yY][tT][eE][sS])      { return MBYTES; }
([bB][yY][tT][eE][sS])          { return BYTES; }
([mM][bB][pP][sS])            { return MBPS; }
([kK][bB][pP][sS])            { return KBPS; }
([bB][iI][tT][sS])            { return BITS; }
([bB][pP][sS])                { return BPS; }
([sS][eE][cC])                { return SEC; }
```

```

([mM][sS][eE][cC])          { return MSEC; }
([uU][sS][eE][cC])          { return USEC; }
([pP][aA][cC][kK][eE][tT][sS]) { return PACKETS; }
([nN][oO][dD][eE][_][pP][aA][rR][aA][mM]) { return NPARAM; }
([lL][iI][nN][kK][_][pP][aA][rR][aA][mM]) { return LPARAM; }
([pP][aA][tT][hH][_][pP][aA][rR][aA][mM]) { return PPARAM; }

/*
* Reserved Symbols
*/

">="      { /* printf ("LEX:Symbol \">>=\"\n"); */ return GTEQ; }
"<="      { /* printf ("LEX:Symbol \"><=\"\n"); */ return LSEQ; }
"!="      { /* printf ("LEX:Symbol \">!=\"\n"); */ return NTEQ; }
":="      { /* printf ("LEX:Symbol \">:=\"\n"); */ return ASSIGN; }
"=="      { return EQUAL; }

/* CHANGE 3-A: The following line of code is added to be used between
the elements of Path Conditions element of a policy rule */

"&&"      { return AND; }

/* CHANGE 4-A: The following line of code is added to be used between
the elements of Target Traffic part of a policy rule */

"||"      { return OR; }

/*
* Variables
*/

[a-zA-Z]+[a-zA-Z_0-9]* {yylval.string = yytext;return LEGALVAR; }

/*
* Numbers
*/

(\\-|\\+)?([0-9]+)      { yyval.string = yytext; return INTEGER; }

```

```

(\-|\+)?([0-9]*\.[0-9]+)
{ yyval.string = yytext; return FLOAT; }
([0-9]{1,3}\.\*\.\*\.\*)
{yyval.string = yytext;return QUAD_DOT_1;}
([0-9]{1,3}\.[0-9]{1,3}\.\*\.\*)
{yyval.string = yytext;return QUAD_DOT_2;}
([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.\*)
{yyval.string = yytext;return QUAD_DOT_3;}
([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})
{yyval.string = yytext;return QUAD_DOT_4;}

[\n]      { lineno++;}
.          {/ *printf("LEX:CATCHALL\"%c\"\\n",yytext[0]);*/ return yytext[0];}

/* USER SUBROUTINES */
%%

/*
 * Called when the lexical analyzer encounters an end of file,
 * this routine will return a 1 indicating a zero
 * token should be returned to the parser to
 * report the end of file.
 */
int yywrap()
{
    return 1;
};

/*
 * A start of comment has been detected, this routine will
 * scan the input until the end of comment is detected
 */
void parse_comment()
{
    char c1,c2;
    while ((c1 = input()) != EOF)

```



```

    {
        if ((c1 == '*' ) && ((c2=input()) == '/'))
        {
            return;
        }
        if (c1 == '\n')
            lineno++;
    }
}

/*
 * Report an error to the user, passing it the string to
 * print out.
 */
void yyerror (char *s)
{
    printf ("%s on line %d at \"%s\"\n", s, lineno, yytext);
    printf ("\nParse failed\n");
    if (verbose){
        dump_tables();
    }
    exit(1);
}

void dump_tables()
{
    int i;
    printf ("Dumping Symbol Table\n");
    for (i = 0; i < MAXSYMBOLS; i++){
        if (symtable[i].name == NULL)
            break;
        printf ("\t%s", symtable[i].name);
        switch (symtable[i].type) {
            case DEFINED_CLASS :    printf ("\tCLASS "); break;
            case DEFINED_MESSAGE:   printf ("\tMESSAGE "); break;
            case DEFINED_TYPE:      printf ("\tTYPE "); break;
        }
    }
}

```

```

        case DEFINED_PATH:      printf ("\tPATH "); break;
        case DEFINED_NODE:      printf ("\tNODE "); break;
        case DEFINED_LINK:      printf ("\tLINK "); break;
        case POLICYID:          printf ("\tPOLICY ID "); break;
        case DEFINED_USER:      printf ("\tDEFINED USER "); break;
        case TARGET_ALL:        printf ("\tALL TARGETS SELECTED"); break;
        default:                 printf ("\tUNKNOWN "); break;
    }
    printf ("\n");
}

printf ("Dumping DEFINED Entry Table\n");
for (i = 0; i < MAXCLASSENTRIES; i++){
    if (class_entries[i].name == NULL)
        break;
    printf ("\t%s \t%s\n", (class_entries[i].class_reference)-
        >name, class_entries[i].name);
}

}

void replace_dots(char s[])
{
    int i;
    for (i=0; s[i] != '\0'; i++){
        if (s[i] == '.') s[i] = ',';
    }
}

```

APPENDIX B. CHANGED VERSION OF PPL PARSE.YACC FILE

```

/*****
File: parser.yacc
Author: Gary Stone. Changes made by Tufan Ekin are in bold font.
Course: Thesis (CS 0810)
Date: August 2001.
Description: This file is a part of the PPL Compiler code in [Ref.2]
which establishes the relationships among tokens (meaningful units).
*****/
/* INCLUDE SECTION */

%{
#include "symbol.h"
#include <stdio.h>
#include <string.h>

/* Function prototype to print out the
 * compiler user options
 */
void print_usage();

/*
 * Text string passed in by flex
 */
extern char *yytext;

/*
 * Pointer to output file created to contain
 * formal logic statements
 */
FILE *logic_out;

/*
 * Variables used record the compiler options
 * entered by the user
 */
int no_wild = 0;
int no_implicit_deny = 0;
extern int verbose;

/*
 * Temporary Pointers to entries in Symbol Table
 */
struct symtab *class_id = 0;
struct symtab *function_id = 0;
struct symtab *type_id = 0;
struct symtab *path_id = 0;
struct symtab *param_id = 0;

```

```

struct symtab *policy_id = 0;
struct symtab *link_name = 0;
struct symtab *link_node1 = 0;
struct symtab *link_node2 = 0;

/* Return value from symbol lookup function */
int class_found;

/* Flag to indicate the first time in the loop */
int first_time;

/*
 * Flag to indicate if defined type is to be
 * represented in formal logic, and output to file.
 */
int output_flag = 0;

/* Number of policies processed */
int policy_count = 0;

/*
 * Flag to indicate messages supported on network links
 * are to output to the file.
 */
int message_flag = 0;

/* Count to uniquely identified defined types internally */
int type_count = 0;

/*
 * Variables to hold clock times converted from strings to
 * integers
 */
int clock_time, clock_time1, clock_time2;

%}

/* DEFINITION SECTION */

/*
 * Union to hold possible symbol types
 */
%union {
    struct symtab *symp;
    char *string;
}

/* Define the following Variables as type string */
%token <string> LEGALVAR
%token <string> INTEGER
%token <string> FLOAT
%token <string> QUAD_DOT_1
%token <string> QUAD_DOT_2
%token <string> QUAD_DOT_3
%token <string> QUAD_DOT_4

```

```

/*
 * Reserved Symbols
 */

/*    CHANGE 3-B: Token AND is added to be used between elements in the
Path Conditions part of a policy. */

/*    CHANGE 4-B: Token OR is added to be used between elements in the
Target Traffic part of a policy. */

%token GTEQ LSEQ NTEQ ASSIGN EQUAL CR WILDCARD AND OR

/*
 * Reserved Words
 */

/* CHANGE 5-B: HOSTIP token is changed to SRCIPADDRESS */

%token DENY PRIORITY ALLOW PERMIT HOPCOUNT SRCIPADDRESS USERID

/* CHANGE 7-B: The following 4 tokens are added to be used in the
Action Items part of a policy */

%token ALLOCATED_BW MAXLOSSRATE DELAYBOUND SECURITYLEVEL

%token TIME DAY DEFINE CLASS MESSAGE TYPE PATH NODE LINK POLICY_MAKER
%token NPARAM LPARAM PPARAM
%token BW

/*
 * Measurement Units
 */
%token MBYTES BYTES GBPS MBPS KBPS BITS BPS SEC MSEC USEC PACKETS

    /* RULES SECTION */
%%

system_policy:    define_list
                |    policy_list
                |    define_list policy_list
                ;

/*
 * Rules to allow user defined classes, messages, types, paths,
 * nodes, links, and users
 */

define_list:      define_type
                |    define_list define_type
                ;

define_type:  DEFINE  class_define_list
            |  DEFINE  type_define_list
            |  DEFINE  path_define_list
            |  DEFINE  node_define_list

```

```

        | DEFINE link_define_list
        | DEFINE user_define_list
        | DEFINE param_define_list
    ;

/*
 * Rules for defining parameters associated with nodes, links, and
 * paths
 */

param_define_list:      NPARAM LEGALVAR
    {if (sym_look ($2,DEFINED_NODE) != FOUND_ENTRY)
        {yyerror ("Unknown NODE ");}
    else {param_id = sym_tab_ref ($2, DEFINED_NODE);
        message_flag = 0;};}
    {' ' param_list'} ' ';
    | LPARAM LEGALVAR
    {if (sym_look ($2,DEFINED_LINK) != FOUND_ENTRY)
        {yyerror ("Unknown LINK ");}
    else{param_id = sym_tab_ref ($2, DEFINED_LINK);
        message_flag = 1;};}
    {' ' param_list '}' ' ';
    | PPARAM LEGALVAR
    {if (sym_look ($2,DEFINED_PATH) != FOUND_ENTRY)
        {yyerror ("Unknown PATH ");}
    else{param_id = sym_tab_ref ($2, DEFINED_PATH);
        message_flag = 0;};}
    {' ' param_list '}' ' ';
    ;

param_list:      param_element
    | param_list ' ' param_element
    ;

/*CHANGE 8: INTEGER WAS CHANGED TO FLOAT FOR A BW VALUE */

param_element:      BW ASSIGN FLOAT /*INTEGER*/
    {fprintf(logic_out,"path_param(\'%s\','\'BW\','\'
        \\'==\','\'%s", (param_id)->name,$3);}
    bw_unit
    {fprintf(logic_out,")\n");}
    | LEGALVAR
    {if (sym_look ($1,DEFINED_MESSAGE) != FOUND_ENTRY){
        sym_add($1,DEFINED_MESSAGE);}
    if(sym_tab_entry_lookup(param_id, $1) == NO_ENTRY){
        sym_tab_entry_add (param_id, $1);};
    if (message_flag)
        fprintf(logic_out,"path_message(\'%s\','\'%s\')\n",
            (param_id)->name,$1);}
    '(' ' ' ')'
    ;

/*
 * Rules for defining users who can make policies

```

```

*/

user_define_list: POLICY_MAKER user_list ';'
;

user_list:      define_user
               | user_list ',' define_user
               ;

define_user:    LEGALVAR
               {if (sym_look ($1,DEFINED_USER) == FOUND_ENTRY)
                {yyerror ("Re-definition of USER ");}
               else {class_id = sym_add($1, DEFINED_USER);};}
               '(' INTEGER
               {sym_tab_entry_add(class_id, $4);}
               ')'
               ;

/*
 * Rules for defining links in the network
 */

link_define_list: LINK link_list ';'
;

link_list:      link
               | link_list ',' link
               ;

link:           LEGALVAR
               {if (sym_look ($1,DEFINED_LINK) == FOUND_ENTRY)
                {yyerror ("Re-definition of Link ");}
               else{link_name = sym_add($1, DEFINED_LINK);};}
               '<' LEGALVAR
               {if ((sym_look ($4,DEFINED_NODE)) != FOUND_ENTRY)
                {yyerror ("Unknown Node in Link ");};
                link_node1 = sym_tab_ref ($4, DEFINED_NODE);}
               ',' LEGALVAR
               {if ((sym_look ($7,DEFINED_NODE)) != FOUND_ENTRY)
                {yyerror ("Unknown Node in Link ");};
                link_node2 = sym_tab_ref ($7, DEFINED_NODE);}
               '>'
               {fprintf(logic_out,"link(\'%s\',\'%s\',\'%s\').\n",
                (link_name)->name,(link_node1)->name,
                (link_node2)->name);
               fprintf(logic_out,"link(\'%s\',\'%s\',\'%s\').\n",
                (link_name)->name,(link_node2)->name,
                (link_node1)->name);
               }
               ;

/*
 * Rules for defining nodes in the network

```

```

*/

node_define_list: NODE node_list';'
;

node_list:      node
              |   node_list ',' node
              ;

node: LEGALVAR
      {if ((class_found = sym_look ($1,DEFINED_NODE)) == FOUND_ENTRY)
        {yyerror ("Re-definition of Node ");}
      else {class_id = sym_add($1, DEFINED_NODE);};}
;

/*
 * Rules for defining paths in the network
 */

path_define_list: path_define_section
                  |   path_define_list path_define_section
                  ;

path_define_section:  PATH LEGALVAR
                     {if ((sym_look ($2,DEFINED_PATH)) == FOUND_ENTRY)
                       {yyerror ("Re-definition of Path ");}
                     else {class_id = sym_add($2, DEFINED_PATH);
                           fprintf (logic_out,"path_links(\'%s\',[", $2);};}
                     {' define_path_list '}' ' ';'} {fprintf (logic_out,")\n");}
                     ;

define_path_list: define_path
                  |define_path_list',' {fprintf (logic_out,",");} define_path
                  ;

define_path:      '<' WILDCARD '>' {fprintf (logic_out,"'*");}
                  |   WILDCARD {fprintf (logic_out,"'*");}
                  |   '<' define_path_element_list '>'
                  ;

define_path_element_list:define_path_element','{fprintf(logic_out,",");}
}
define_path_element
    |define_path_element_list ',' {fprintf(logic_out,",");}
    define_path_element
    ;

define_path_element:  LEGALVAR
                     {if (sym_look ($1, DEFINED_NODE) != FOUND_ENTRY)
                       {yyerror ("Invalid Path (Node not defined) ");}
                     else{fprintf (logic_out,"%s", $1);};}
                     |   WILDCARD {fprintf (logic_out,"'*");}
                     ;

/*
 * Rules for defining traffic classes to be used in the network

```



```

*/

class_define_list:      class_define_section
                        |      class_define_list class_define_section
                        ;

class_define_section:   CLASS LEGALVAR
                        {if ((sym_look ($2,DEFINED_CLASS)) == FOUND_ENTRY)
                          {yyerror ("Re-definition of Class ");}
                        else {class_id = sym_add($2, DEFINED_CLASS);};}
                        {' ' class_element_list '}' ';' ;

class_element_list:     class_element
                        |      class_element_list ',' class_element
                        ;

class_element:          LEGALVAR {sym_tab_entry_add(class_id, $1);};

/*
 * Rules for defining types to be used in the network
 */

type_define_list: type_define_section
                  |      type_define_list class_define_section
                  ;

type_define_section:   TYPE LEGALVAR
                      {type_count = 0;
                      if ((sym_look ($2, DEFINED_TYPE)) == FOUND_ENTRY)
                        {yyerror ("Re-definition of Class ");}
                      else {class_id = sym_add($2, DEFINED_TYPE);};}
                      {' ' type_element_list '}' ';' ;

type_element_list:     type_element
                      |      type_element_list ',' type_element
                      ;

type_element:          LEGALVAR
                      {sym_tab_entry_type_add(class_id,$1,
                      type_count);type_count++;};

/*
 * Syntax for Policy Term
 * policyID <userID> {paths} {target} {conditions} {action_items}
 * - policyID          - unique policy identification token
 * - userID            - user ID of policy creator
 * - paths              - network paths the policy affects
 * - target             - target class of network traffic
 * - conditions         - any global conditions (items are AND'ed)
 * - action_items       - for setting parameters (e.g. policy priority),
 *                        declaring compromises and explicit deny, etc.
 */

/*
 * Rules for defining policies
 */

```

```

policy_list:      policy
                  |
                  policy_list policy
                  ;

/* CHANGE 1: "@" sign is added in front of Target Paths element.*/

policy:      policyID userID '@' '{' path_list '}' '{'
             {fprintf(logic_out,"policy_targets(\'%s\',[" , (policy_id)->name);}
             target_list
             {fprintf (logic_out,")].\n");}
             '}' '{' condition_list '}' '{' action_list '}' ' ';
             ;

policyID:     LEGALVAR
             {if (sym_look ($1, POLICYID) == FOUND_ENTRY)
               {yyerror ("Re-definition of Policy ID ");}
             else {policy_id = sym_add($1, POLICYID);};policy_count++;}
             ;

userID:       LEGALVAR
             {if (sym_look ($1, DEFINED_USER) != FOUND_ENTRY)
               {yyerror ("UNKNOWN user ");}
             else
               {fprintf(logic_out,"policy_owner(\'%s\',
               \'%s\').\n", (policy_id)->name, $1);};}
             ;

/*
 * Syntax for Path Component
 */

path_list:    defined_path
              |
              path_list ',' defined_path
              ;

defined_path:  LEGALVAR
              {if ((sym_look ($1,DEFINED_PATH) != FOUND_ENTRY) &&
                  (sym_look ($1,DEFINED_LINK) != FOUND_ENTRY) &&
                  (sym_look ($1,DEFINED_NODE) != FOUND_ENTRY)){
                  yyerror ("Unknown DEFINED element ");}
              else{fprintf(logic_out,"policy_path(\'%s\',\'%s\').\n",
              (policy_id)->name,$1);};}
              ;

/*
 * Syntax for Target Component

```

```

*/

target_list:      target {}

/* CHANGE 4-C: ',' sign is changed to OR (||), since the elements in
the Target Traffic part of a policy are OR'ed */

|      target_list OR {fprintf(logic_out, ",");} target
|
|      WILDCARD {fprintf(logic_out, "'*', '*', []");
|                  sym_add((policy_id)->name, TARGET_ALL);}
;
target:      LEGALVAR
|      {if (sym_look($1, DEFINED_CLASS) == FOUND_ENTRY)
|                  {class_id = sym_tab_ref($1, DEFINED_CLASS);
|                  fprintf(logic_out, "'%s'", $1);}
|      else yyerror ("Invalid Class");}

/* CHANGE 2-A: '{' sign is changed to '<' to standardize set sign in
PPL */

|      target_symbol '<'
|      {fprintf(logic_out, "[");}
|      target_element_list
|      {fprintf(logic_out, ""]");}

/* CHANGE 2-B: '}' sign is changed to '>' to standardize set sign in
PPL */

|      '>'

;
target_element_list:      target_element {}
|      target_element_list', '{fprintf(logic_out, ",");}
|      target_element
;
target_element:      LEGALVAR
|      {if (sym_tab_entry_look( (class_id)->name, $1,
|      DEFINED_CLASS) == NO_ENTRY)
|      {yyerror ("Invalid Class Entry");}
|      fprintf(logic_out, "'%s'", $1);}
;
target_symbol:      EQUAL {fprintf(logic_out, "=", "");}
|      NTEQ {fprintf(logic_out, "!=", "");}
;

```

```

/*

```

```

* Syntax for Conditions Component
*/

condition_list:  WILDCARD
{fprintf(logic_out,"no_conditions(\'%s\').\n", (policy_id)->name);}

|      {class_id = function_id = type_id = (struct symtab *) -1;}
      condition {}

/* CHANGE 3-C: ', ' sign is changed to AND (&&) since the elements in
the Path Conditions part of a policy are AND'ed */

|      condition_list AND {class_id = function_id = type_id =
      (struct symtab *) -1;}

      condition {}
;
condition:  PRIORITY LSEQ INTEGER

{fprintf(logic_out,"condition_path(\'%s\',priority,<=,%s).\n",
      (policy_id)->name,$3);}
|      PRIORITY GTEQ INTEGER

{fprintf(logic_out,"condition_path(\'%s\',priority,>=,%s).\n",
      (policy_id)->name,$3);}
|      HOPCOUNT GTEQ INTEGER

{fprintf(logic_out,"condition_path(\'%s\',hopcount,>=,%s).\n",
      (policy_id)->name,$3);}
|      HOPCOUNT LSEQ INTEGER

{fprintf(logic_out,"condition_path(\'%s\',hopcount,>=,%s).\n",
      (policy_id)->name,$3);}
|      TIME LSEQ
{fprintf(logic_out,"condition_path(\'%s\',time,<=,",
      (policy_id)->name);} time
|      TIME GTEQ
{fprintf(logic_out,"condition_path(\'%s\',time,>=,",
      (policy_id)->name);} time

/* CHANGE 5-C: HOSTIP is changed to SRCIPADDRESS */

|      SRCIPADDRESS EQUAL

{fprintf(logic_out,"condition_path(\'%s\',\'host_id\',\'!=\',",
      (policy_id)->name);}

      quad_dot
      {fprintf(logic_out,")\n");}

/* CHANGE 5-C: HOSTIP is changed to SRCIPADDRESS */

|      SRCIPADDRESS NTEQ

{fprintf(logic_out,"condition_path(\'%s\',\'host_id\',\'!=\',",
      (policy_id)->name);}

```

```

quad_dot
{fprintf(logic_out,")\n");}
| bandwidth_symbol GTEQ
{fprintf(logic_out,"condition_path(\'%s\',\'BW\',>=,[",
(policy_id)->name);}
number_bw_units
{fprintf(logic_out,")\n");}
| bandwidth_symbol LSEQ
{fprintf(logic_out,"condition_path(\'%s\',\'BW\',<=,[",
(policy_id)->name);}
number_bw_units
{fprintf(logic_out,")\n");}

| USERID EQUAL LEGALVAR
{fprintf(logic_out,"condition_path(\'%s\',\'user_id\',\'==\'
\',\'%s\')\n",
(policy_id)->name,$3);}
| USERID NTEQ LEGALVAR
{fprintf(logic_out,"condition_path(\'%s\',\'user_id\',\'!=\'
\',\'%s\')\n",
(policy_id)->name,$3);}
| LEGALVAR
{if (sym_look ($1, DEFINED_MESSAGE) == FOUND_ENTRY){
function_id = sym_tab_ref ($1, DEFINED_MESSAGE);
fprintf(logic_out,"policy_message(\'%s\',\'%s\')\n",
(policy_id)->name,$1);
};
if (sym_look ($1, DEFINED_TYPE) == FOUND_ENTRY){
type_id = sym_tab_ref ($1, DEFINED_TYPE);
output_flag = 1;
};
if (class_id == (struct symtab *) -1 &&
function_id == (struct symtab *) -1 &&
type_id == (struct symtab *) -1){
yyerror ("Unknown DEFINED element ");
};}
conditional_defined_operations {}
;

time: INTEGER
{clock_time = atoi($1);
if (clock_time >= 0 && clock_time < 2400){
if(clock_time < 10){fprintf(logic_out,"000%d00).\n",clock_time);}
else if (clock_time < 100){fprintf(logic_out,"00%d00).\n",
clock_time);}
else if (clock_time < 1000){fprintf(logic_out,"0%d00).\n",
clock_time);}
else {fprintf(logic_out,"%d00).\n",clock_time);}
}
else {yyerror ("Invalid Time: range 0000-2359 ");}}
| INTEGER ':' INTEGER
{clock_time1 = atoi($1);
clock_time2 = atoi($3);
if (clock_time1 >= 0 && clock_time1 < 2400 ){
if (clock_time1 < 10){fprintf(logic_out,"000%d",clock_time1);}

```

```

        else if (clock_time1 < 100){fprintf(logic_out,"00%d",
        clock_time1);}
        else if (clock_time1 < 1000){fprintf(logic_out,"0%d",
        clock_time1);}
        else {fprintf(logic_out,"%d",clock_time1);}
    }
    else {yyerror ("Invalid Time: range 0000:00-2359:59 ");}
    if (clock_time2 >= 0 && clock_time2 <= 59){
        if (clock_time2 < 10){fprintf(logic_out,"0%d").\n",
        clock_time2);}
        else {fprintf(logic_out,"%d").\n",clock_time2);}

    }
    else {yyerror ("Invalid Time X:Y : X range 0000-2359, Y range 00-
    59");}}
;

```

```

conditional_defined_operations: conditional_operations
    | '(' ' ' )'
        {if (function_id == (struct symtab *) -1) {
            yyerror ("Unknown Message");};}
        conditional_symbol number_units
;

```

```

conditional_operations: {if (output_flag){
    fprintf(logic_out,"condition_path(\'%s\',\'%s\',",
    (policy_id)->name,(type_id)->name);
};}
conditional_symbol LEGALVAR {
if (class_id == (struct symtab *) -1 &&
    type_id == (struct symtab *) -1){
    yyerror ("Invalid CLASS/TYPE");
}
if (class_id != (struct symtab *) -1){
    if(sym_tab_entry_look((class_id)->name,$3,
    DEFINED_CLASS) == NO_ENTRY){
        yyerror ("Invalid CLASS Member");
    }
}
if (type_id != (struct symtab *) -1) {
    if(sym_tab_entry_look((type_id)->name,$3,
    DEFINED_TYPE) == NO_ENTRY){
        yyerror ("Invalid TYPE member");
    }
    else{
        fprintf(logic_out,"\'%s\').\n", $3);
    }
};}
;

```

```

number_bw_units: output_number bw_unit;

```

```

bw_unit:      GBPS {fprintf(logic_out, "\\GBPS\\");}
|            MBPS {fprintf(logic_out, "\\MBPS\\");}
|            KBPS {fprintf(logic_out, "\\KBPS\\");}
|            BPS  {fprintf(logic_out, "\\BPS\\");}
;

number_units:      number
|                  number unit
;

conditional_symbol:
    GTEQ {if (output_flag) {fprintf(logic_out, "\\>=\\");output_flag =
    0;};}
|    LSEQ {if (output_flag) {fprintf(logic_out, "\\<=\\");output_flag =
    0;};}
|    NTEQ {if (output_flag) {fprintf(logic_out, "\\!=\\");output_flag =
    0;};}
|    EQUAL {if (output_flag) {fprintf(logic_out, "\\==\\");output_flag =
    0;};}
|    '>' {if (type_id != (struct symtab *) -1){yyerror ("Invalid
    Type Operator");}
        if (output_flag){fprintf(logic_out, "\\>\\");output_flag =
        0;};}
|    '<' {if (type_id != (struct symtab *) -1){yyerror ("Invalid
    Type Operator");}
        if (output_flag) {fprintf(logic_out, "\\<\\"); output_flag =
        0;};}
;

bandwidth_symbol: BW {}
;

number:      INTEGER      {}
|            FLOAT {}
;

output_number:      INTEGER      {fprintf(logic_out, "%s", $1);}
|                  FLOAT        {fprintf(logic_out, "%s", $1);}
;

quad_dot:QUAD_DOT_1 {replace_dots($1); fprintf(logic_out, "[%s]", $1);}
|    QUAD_DOT_2 {replace_dots($1); fprintf(logic_out, "[%s]", $1);}
|    QUAD_DOT_3 {replace_dots($1); fprintf(logic_out, "[%s]", $1);}
|    QUAD_DOT_4 {replace_dots($1); fprintf(logic_out, "[%s]", $1);}
;

unit: '%'
|    MBYTES
|    BYTES
|    BITS
|    SEC
|    MSEC
|    USEC
|    PACKETS

```

```

| GBPS  /* CHANGE 7-D-1: GBPS is added to be used as the unit of
         allocated_bw in the action part of a policy */
| MBPS  /* CHANGE 7-D-2: MBPS is added to be used as the unit of
         allocated_bw in the action part of a policy */
| KBPS  /* CHANGE 7-D-3: KBPS is added to be used as the unit of
         allocated_bw in the action part of a policy */
| BPS   /* CHANGE 7-D-4: BPS is added to be used as the unit of
         allocated_bw in the action part of a policy */
;
/*
 * Syntax for Action Items Component
 */

action_list:      DENY
{fprintf(logic_out,"policy_action('%s',deny).\n", (policy_id)->name);
 if (sym_look ((policy_id)->name, TARGET_ALL) == FOUND_ENTRY)
 {fprintf(logic_out,"target('%s',deny_all).\n", (policy_id)-
->name);};}
|
|      action
|      action_list ',' action
;
action:           action_reserved_word
|
|      action_reserved_word ASSIGN number_units
;

action_reserved_word:  PRIORITY
{fprintf(logic_out,"policy_action('%s',permit).\n", (policy_id)->name);
 if (sym_look ((policy_id)->name, TARGET_ALL) == FOUND_ENTRY)
 {fprintf(logic_out,"target('%s',permit_all).\n", (policy_id)
->name);};}
|
|      PERMIT
{fprintf(logic_out,"policy_action('%s',permit).\n", (policy_id)->name);
 if (sym_look ((policy_id)->name, TARGET_ALL) == FOUND_ENTRY)
 {fprintf(logic_out,"target('%s',permit_all).\n", (policy_id)
->name);};}
|
|      HOPCOUNT
{fprintf(logic_out,"policy_action('%s',permit).\n", (policy_id)->name);
 if (sym_look ((policy_id)->name, TARGET_ALL) == FOUND_ENTRY)
 {fprintf(logic_out,"target('%s',permit_all).\n", (policy_id)
->name);};}

/* CHANGE 7-C: The following 4 action reserved words (ALLOCATED_BW,
MAXLOSSRATE, DELAYBOUND, SECURITYLEVEL) are added, and they enforce
"implicit permit" just like PRIORITY and HOPCOUNT. */

|
|      ALLOCATED_BW
{fprintf(logic_out,"policy_action('%s',permit).\n", (policy_id)->name);
 if (sym_look ((policy_id)->name, TARGET_ALL) == FOUND_ENTRY)
 {fprintf(logic_out,"target('%s',permit_all).\n", (policy_id)
->name);};}
|
|      MAXLOSSRATE
{fprintf(logic_out,"policy_action('%s',permit).\n", (policy_id)->name);
 if (sym_look ((policy_id)->name, TARGET_ALL) == FOUND_ENTRY)
 {fprintf(logic_out,"target('%s',permit_all).\n", (policy_id)
->name);};}

```



```

|                                DELAYBOUND
{fprintf(logic_out,"policy_action('%s',permit).\n", (policy_id)->name);
  if (sym_look ((policy_id)->name, TARGET_ALL) == FOUND_ENTRY)
    {fprintf(logic_out,"target('%s',permit_all).\n", (policy_id)
      ->name);};}

|                                SECURITYLEVEL
{fprintf(logic_out,"policy_action('%s',permit).\n", (policy_id)->name);
  if (sym_look ((policy_id)->name, TARGET_ALL) == FOUND_ENTRY)
    {fprintf(logic_out,"target('%s',permit_all).\n", (policy_id)
      ->name);};}

;

%%

extern FILE *yyin;
extern FILE *logic_out;

main(int ac, char **av)
{
    FILE *prologrules;          /* Input files of ProLog policy rules */
    int i,j,count,c,first_time; /* Loop counters and flags */
    /*
     * File to contain Prolog Facts from PPL Configuration File
     * and the ProLog policy rules
     */
    char *logic_file_name;

    /*
     * Scan the arguments passed in with the compile statement.
     * Set the appropriate flags according to the user's request
     */
    while (--ac > 0){
        /* User wanted the Symbol Tables Dumped after parse */
        if (strcmp (av[ac],"-v") == 0){
            verbose = 1;
        }
        /* User requests no wild card expansion */
        else if (strcmp (av[ac],"-no_wild") == 0){
            no_wild = 1;
        }
        /*
         * Do not implicit denies in the conflict detection+0
         * phase if the policy's creater are the same.
         */
        else if (strcmp (av[ac],"-no_implicit_deny") == 0){
            no_implicit_deny = 1;
        }
        /* Test the input configuration file */
        else {
            if ((yyin = fopen(av[ac], "r")) == NULL) {
                perror (av[ac]);
                exit(1);
            }
        }
    }
}

```

```

        logic_file_name = (char *)malloc ((size_t)15);
        sprintf (logic_file_name,"ppl1.txt");

        if ((logic_out = fopen(logic_file_name,"w")) == NULL) {
            perror (logic_file_name);
            exit(1);
        }
    }

}

/*
 * If the input file was not specified or invalid,
 * print compiler usage to the user
 */
if (yyin == NULL){
    print_usage();
    exit(1);
}

/* Provide parsing output and number of policies parsed */
if (!yyvsparse()){
    printf ("\nParse worked\n");
    if (policy_count == 0){
        printf("No policies specified\n\n");
        exit(0);
    }
    else
        printf("%d policies scanned\n\n",policy_count);
}
else
    printf ("\nParse failed\n\n");

/*
 * Loop thru the symbol table to create ProLog facts about
 * the nodes used to construct the network.
 */
count = 0;
for (i = 0; i < MAXSYMBOLS; i++){
    if (symtable[i].name == NULL)
        break;
    switch (symtable[i].type) {
        case DEFINED_NODE:
            fprintf (logic_out,"node_label('%s',%i).\n",
                    symtable[i].name,count++);
            break;
    }
}

/*
 * Loop thru the symbol table to create ProLog facts about
 * user defined classes of traffic.
 */
count = 0;
for (i = 0; i < MAXSYMBOLS; i++){
    if (symtable[i].name == NULL)

```

```

        break;
    switch (symtable[i].type) {
        case DEFINED_CLASS:
            first_time = 1;
            fprintf (logic_out, "class('%s', [",
                    symtable[i].name);
            for (j = 0; j < MAXCLASSENTRIES; j++){
                if (class_entries[j].name == NULL)
                    break;
                if (class_entries[j].class_reference ==
                    &(symtable[i])){
                    if (first_time){
                        fprintf (logic_out, "'%s'",
                                class_entries[j].name);
                        first_time = 0;
                    }
                    else
                        fprintf (logic_out, ", '%s'",
                                class_entries[j].name);
                }
            }
            fprintf (logic_out, "]).\n");
            break;
    }
}

/*
 * Loop thru the symbol table to create ProLog facts about
 * user defined types to be used in conditional statements,
 * and users defined who can create network policies.
 */
count = 0;
for (i = 0; i < MAXSYMBOLS; i++){
    if (symtable[i].name == NULL)
        break;
    switch (symtable[i].type) {
        case DEFINED_TYPE:
            for (j = 0; j < MAXCLASSENTRIES; j++){
                if (class_entries[j].name == NULL)
                    break;
                if (class_entries[j].class_reference ==
                    &(symtable[i])){
                    fprintf (logic_out, "type(\'%s\',
                                \'%s\', %d).\n",
                                symtable[i].name, class_entries[j].name, class_entries[j].value);
                }
            }
            break;
        case DEFINED_USER:
            for (j = 0; j < MAXCLASSENTRIES; j++){
                if (class_entries[j].name == NULL)
                    break;
                if (class_entries[j].class_reference ==
                    &(symtable[i])){
                    fprintf (logic_out, "user(\'%s\',
                                \'%s\').\n",

```

```

        symtable[i].name,class_entries[j].name);
    }
}
break;
}
}

/*
 * Output dummy facts to keep ProLog interpreter from choking
when referencing facts that do not exist.
*/
fprintf (logic_out,"no_conditions(\'_null_\').\n");
fprintf (logic_out,"target(\'_null_\',\'_null_\').\n");
fprintf (logic_out,"path_message(\'_null_\',\'_null_\').\n");
fprintf (logic_out,"policy_message(\'_null_\',\'_null_\').\n");
fprintf(logic_out,"condition_path(\'_null_\',\'_null_\',\'_null_\',
    \'_null_\').\n");

/*
 * Set flag in Prolog to allow wild card characters to be
expanded
 * or not, depending on the user's direction
 */
if (no_wild)
    fprintf (logic_out,"wild(\'no\').\n");
else
    fprintf (logic_out,"wild(\'yes\').\n");

/*
 * Set flag in Prolog to allow implicit denies between
 * policies created by the same user to be ignored.
 */
if (no_implicit_deny)
    fprintf (logic_out, "user_implicit_deny(\'no\').\n");
else
    fprintf (logic_out, "user_implicit_deny(\'yes\').\n");

/*
 * Create and run the first stage of the conflict detection
 * process. This involves sorting the facts just created by
 * the parsing process. The sorting is needed only to stop
 * Prolog from issuing warnings about facts not being contiguous
 * in the file. Add to the end of the file, the Prolog rules that
 * are to be applied to the facts entered from the
 * prologrules1.txt file. Then execute the Prolog program by
 * calling the Prolog command line interface with input file.
 */
fclose(logic_out);
system("sort ppl1.txt > ppl1.pl");

if ((logic_out = fopen("ppl1.pl", "a")) == NULL) {
    perror ("ppl1.pl");
    exit(1);
}

```

```

    if ((prologrules = fopen("prologrules1.txt", "r")) == NULL) {
        perror (av[1]);
        exit(1);
    }

while ((c = getc(prologrules)) != EOF)
    putc(c, logic_out);

fclose(prologrules);
fclose(logic_out);

/* If the user asked for verbose mode, dump the symbol tables. */
if (verbose){
    dump_tables();
}
system("pl/bin/plcon.exe -f ppl1.pl -t stage1");

/*
 * Stage two of the conflict detection takes the modified facts
 * generated by stage1, and adding the rules to be applied
 * together in a file named ppl2.pl.
 * The Prolog interpreter then executes this next stage. Stage
 * two like stage one manipulates the facts from stage to stage
 * in a progressive fashion. One stage could have been used, but
 * would be complicated to follow and debug.
 * For efficiency, the combining of stages should be considered.
 */
system("sort ppl2.txt > ppl2.pl");
if ((logic_out = fopen("ppl2.pl", "a")) == NULL) {
    perror ("ppl2.pl");
    exit(1);
}
if ((prologrules = fopen("prologrules2.txt", "r")) == NULL) {
    perror (av[1]);
    exit(1);
}

while ((c = getc(prologrules)) != EOF)
    putc(c, logic_out);
fclose(prologrules);

fclose(logic_out);

system("pl/bin/plcon.exe -f ppl2.pl -t stage2");

/*
 * Stage three is the final stage of the conflict detection and
 * resolution phase of the compiler. The modified facts from
 * stage two are used and applied with the Prolog rules from
 * the file "prologrules3.txt".
 * Final output is created in "scan_out.txt".
 */
system("sort ppl3.txt > ppl3.pl");

```

```

    if ((logic_out = fopen("ppl3.pl", "a")) == NULL) {
        perror ("ppl3.pl");
        exit(1);
    }
    if ((prologrules = fopen("prologrules3.txt", "r")) == NULL) {
        perror (av[1]);
        exit(1);
    }

    while ((c = getc(prologrules)) != EOF)
        putc(c, logic_out);

    fclose(prologrules);
    fclose(logic_out);

    system("pl/bin/plcon.exe -f ppl3.pl -t stage3");

    printf("\n\n Policy Conflict results written to
           \"scan_out.txt\"\n\n");
}

/*
 * Print_usage outputs the options available to the user trying to
 * compile PPL configuration file.
 */
void print_usage()
{
    printf("\n\n Usage: Scan [-v -wild -no_implicit_deny] <filename>\n");
    printf("\t -v                : Verbose mode, dump symbol tables\n");
    printf("\t -no_wild             : Do not support wild cards in paths\n");
    printf("\t -no_implicit_deny : If policy creators are the same
                        between\n");
    printf("\t                a conflict, do not enforce implicit deny\n");
    printf("\t filename            : Filename of PPL configuration file\n");
    printf("\n\n");
}

```

APPENDIX C. PPL MANAGER SOURCE CODE

```

/*****
File: About.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 01/20/2002
Description: Information about PPL and PPL Manager is displayed to the
user by means of this class.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class About extends JDialog {
    private JLabel PPLInfoLabel, PPLManagerInfoLabel, npsLabel,
lineupLabel;
    private JButton okButton;
    private JPanel aboutPanel, buttonPanel;

    /**
     * Method      : About
     * Purpose     : Constructor for About class
     * Parameters  : GuiMenu gui
     */

    public About ( GuiMenu gui ) {
        super ( gui, "About PPL Manager", true );
        // parent Frame is main GUI, and this JDialog is modal
        setLocation (115, 115);
        // A method call for building the GUI for about JDialog
        aboutGuiBuilderMethod ();
        setResizable (false);
        setSize(500,150);
        show();
    } // End of constructor

    /**
     * Method      : aboutGuiBuilderMethod
     * Purpose     : This method places the components of about JDialog.
     * Parameters  : None
     */

    private void aboutGuiBuilderMethod () {

        Container c = getContentPane ();

        aboutPanel = new JPanel ();

        PPLInfoLabel = new JLabel ( "Path-based Policy Language (PPL) is
developed by Geoffrey Xie and Gary Stone.");

```

```

        aboutPanel.add ( PPLInfoLabel );

        PPLManagerInfoLabel = new JLabel ( "PPL Manager is developed by
                                           Geoffrey Xie and Tufan Ekin.");
        aboutPanel.add ( PPLManagerInfoLabel );

        lineupLabel = new JLabel ( "                                ");
        aboutPanel.add ( lineupLabel );

        npsLabel = new JLabel ( "Naval Postgraduate School - 2002");
        aboutPanel.add ( npsLabel );

        c.add (aboutPanel, BorderLayout.CENTER);

        ButtonHandler handlerBut = new ButtonHandler ();

        buttonPanel = new JPanel ();

        okButton = new JButton ( "    OK    ");
        buttonPanel.add(okButton);
        okButton.addActionListener(handlerBut);

        c.add ( buttonPanel, BorderLayout.SOUTH );

    } // End of aboutGuiBuilderMethod

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for event handling of OK button.
 */

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == okButton ) {
            About.this.dispose();
        } // End of if
    } // End of method actionPerformed
} // End of class ButtonHandler

} // End of class About

```



```

/*****
File: AboutFile.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 01/28/2002
Description: Information about "File" menu is displayed to the user by
means of this class.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class AboutFile extends JDialog {

    private JLabel fileInfoLabel, fileInfoLabel2, fileInfoLabel3,
fileInfoLabel4, fileInfoLabel5, fileInfoLabel6;
    private JLabel fileInfoLabel7, fileInfoLabel8;
    private JLabel fileInfoLabel2b, fileInfoLabel3b, fileInfoLabel4b;
    private JButton okButton;
    private JPanel aboutPanel, buttonPanel;

    /**
     * Method      : AboutFile
     * Purpose     : Constructor for AboutFile class
     * Parameters  : GuiMenu gui
     */

    public AboutFile ( GuiMenu gui ) {
        // parent Frame is main GUI, and this JDialog is modal
        super ( gui, "About File Menu", true );
        setLocation (115, 115);
        // A method call for building the GUI for aboutFile JDialog
        aboutFileGuiBuilderMethod ();
        setResizable (false);
        setSize(500,350);
        show();
    } // End of constructor

    /**
     * Method      : aboutFileGuiBuilderMethod
     * Purpose     : This method places the components of aboutFile
                   JDialog which gives information about "File" menu.
     * Parameters  : None
     */

    private void aboutFileGuiBuilderMethod () {

        Container c = getContentPane ();

        aboutPanel = new JPanel ();

        fileInfoLabel = new JLabel ( "INFORMATION ABOUT FILE MENU:");
        aboutPanel.add ( fileInfoLabel );

```

```

        fileInfoLabel2 = new JLabel ("* NEW menu item creates a new PPL
policy file by saving or not saving current network");
        aboutPanel.add ( fileInfoLabel2 );

        fileInfoLabel2b = new JLabel ("topology information (nodes,
links, paths, classes and types) and policies.                ");
        aboutPanel.add ( fileInfoLabel2b );

        fileInfoLabel3 = new JLabel ("* OPEN FILE menu item opens a PPL
policy file which contains only network topology                ");
        aboutPanel.add ( fileInfoLabel3 );

        fileInfoLabel3b = new JLabel ("information or network topology
information together with policies.                            ");
        aboutPanel.add ( fileInfoLabel3b );

        fileInfoLabel4 = new JLabel ("* SAVE FILE menu item saves only
network topology information or network topology                ");
        aboutPanel.add ( fileInfoLabel4 );

        fileInfoLabel4b = new JLabel ("information together with policies
to a file.                                                       ");
        aboutPanel.add ( fileInfoLabel4b );

        fileInfoLabel5 = new JLabel ("* IMPORT POLICY menu item imports
only policies from a file.                                       ");
        aboutPanel.add ( fileInfoLabel5 );

        fileInfoLabel6 = new JLabel ("* SAVE POLICY menu item saves only
policies to a file.                                             ");
        aboutPanel.add ( fileInfoLabel6 );

        fileInfoLabel7 = new JLabel ("* SET COMPILE MODE menu item
selects one of the compilation modes.                            ");
        aboutPanel.add ( fileInfoLabel7 );

        fileInfoLabel8 = new JLabel ("* COMPILE menu item invokes PPL
compiler via BASH shell.                                         ");
        aboutPanel.add ( fileInfoLabel8 );

        c.add (aboutPanel, BorderLayout.CENTER);

        ButtonHandler handlerBut = new ButtonHandler ();

        buttonPanel = new JPanel ();

        okButton = new JButton ("    OK    ");
        buttonPanel.add(okButton);
        okButton.addActionListener(handlerBut);

        c.add ( buttonPanel, BorderLayout.SOUTH );

    } // End of aboutGuiBuilderMethod
/**
 * Class          : ButtonHandler

```

```

    * Purpose      : Inner class for event handling of OK button.
    */

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == okButton ) {
            AboutFile.this.dispose();
        } // End of if
    } // End of method actionPerformed
} // End of class ButtonHandler

} // End of class AboutFile

```

```

/*****
File: Action.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/18/2001
Description: In this file, action element of a policy will be defined.
If the action part of a policy does not have a priority, hop count,
allocated bw, maximum loss rate, delay bound or security level value,
then its appropriate value will be equal to 0. This will also serve as
a flag to understand if action part has a certain value or not.
*****/
import java.util.*;
import java.io.Serializable;

public class Action implements Serializable {
    private boolean hasDenyAction;
    private boolean hasPermitAction;

    private int priorityValue;
    private int hopCountValue;
    private float allocatedBWValue;
    private float maxLossRateValue;
    private float delayBoundValue;
    private int securityLevelValue;

    /**
     * Method      : Action
     * Purpose     : Constructor for Action class
     * Parameters  : None
     */

    public Action () {
        hasDenyAction = false;
        hasPermitAction = false;
        priorityValue = 0;
        hopCountValue = 0;
        allocatedBWValue = 0;
        maxLossRateValue = 0;
        delayBoundValue = 0;
        securityLevelValue = 0;
    } // End of constructor

    /**
     * Method      : set and get methods of Action class
     * Purpose     : To get and set the data members of Action class
     * Parameters  : Each set method has one data member of Action
class. get methods do not have parameters.
     */

    public void setHasDenyAction (boolean b) {
        hasDenyAction = b;
    }

```

```

public boolean getHasDenyAction () {
    return hasDenyAction;
}

public void setHasPermitAction (boolean b) {
    hasPermitAction = b;
}

public boolean getHasPermitAction () {
    return hasPermitAction;
}

public void setPriorityValue (int a) {
    priorityValue = a;
}

public int getPriorityValue () {
    return priorityValue;
}

public void setHopCountValue (int a) {
    hopCountValue = a;
}

public int getHopCountValue () {
    return hopCountValue;
}

public void setAllocatedBWValue (float f) {
    allocatedBWValue = f;
}

public float getAllocatedBWValue () {
    return allocatedBWValue;
}

public void setMaxLossRateValue (float f) {
    maxLossRateValue = f;
}

public float getMaxLossRateValue () {
    return maxLossRateValue;
}

public void setDelayBoundValue (float f) {
    delayBoundValue = f;
}

public float getDelayBoundValue () {
    return delayBoundValue;
}

public void setSecurityLevelValue (int f) {
    securityLevelValue = f;
}

```

```
    public int getSecurityLevelValue () {  
        return securityLevelValue;  
    }  
  
} // End of class Action
```

```

/*****
File: Class.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/30/2001
Description: In this file, user defined classes that can be used in the
target element of a policy rule (items are OR'ed) will be defined.
*****/

```

```

import java.io.Serializable;
import java.util.*;

```

```

public class Class extends Object implements Serializable{
    private String name;
    private Vector classMembersVector;

```

```

    /**
     * Method      : Class
     * Purpose      : Default constructor for class Class
     * Parameters   : None
     */

```

```

    public Class () {

    } // End of default constructor

```

```

    /**
     * Method      : Class
     * Purpose      : Constructor for class Class
     * Parameters   : String n, Vector v
     */

```

```

    public Class (String n, Vector v ) {
        setName ( n );
        setclassMembersVector ( v );
    } // End of Constructor

```

```

    /**
     * Method      : toString
     * Purpose      : to override toString () method of Object class
     * Parameters   : none
     */
    public String toString () {
        return name;
    }

```

```

    /**
     * Method      : set and get methods of Class class

```

```

    * Purpose      : To get and set the data members of Class class
    * Parameters   : Each set method sets one data member of Class
class. get methods do not have parameters.
    */

    public void setName (String n) {
        name = n;
    }

    public String getName () {
        return name;
    }

    public void setclassMembersVector ( Vector v ) {
        classMembersVector = v;
    }

    public Vector getclassMembersVector () {
        return classMembersVector;
    }

} // End of class Class

```



```

/*****
File: Condition.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/07/2001
Description: In this file, condition element of a policy will be
defined.
*****/
import java.util.*;
import java.io.Serializable;

public class Condition implements Serializable {

    private boolean noConditionProperty; // If the policy has no
condition, then this boolean value will be set to true. All data member
vectors in this class will be empty then.

    private Vector priorityConditionVector; // OnePriority objects will
be placed in this vector if there is any
    private Vector hopCountConditionVector;
    private Vector timeConditionVector;
    private Vector srcIPAddressConditionVector;
    private Vector BWConditionVector;
    private Vector userIDConditionVector;
    private Vector typeConditionVector; // OneType class objects will be
placed in this vector if there is any.
    private Vector parameterConditionVector;

    /**
     * Method      : set and get methods of Condition class
     * Purpose      : To set and get the data members of Condition class
     * Parameters   : Each set method has one data member of Condition
                     class. get methods do not have parameters.
     */

    public void setNoConditionProperty (boolean ncp) {
        noConditionProperty = ncp;
    }

    public boolean getNoConditionProperty () {
        return noConditionProperty;
    }

    public void setPriorityConditionVector (Vector v) {
        priorityConditionVector = v;
    }

    public Vector getPriorityConditionVector () {
        return priorityConditionVector;
    }

    public void setHopCountConditionVector (Vector v) {
        hopCountConditionVector = v;
    }

```

```

    }

    public Vector getHopCountConditionVector () {
        return hopCountConditionVector;
    }

    public void setTimeConditionVector (Vector v) {
        timeConditionVector = v;
    }

    public Vector getTimeConditionVector () {
        return timeConditionVector;
    }

    public void setSrcIPAddressConditionVector (Vector v) {
        srcIPAddressConditionVector = v;
    }

    public Vector getSrcIPAddressConditionVector () {
        return srcIPAddressConditionVector;
    }

    public void setBWConditionVector (Vector v) {
        BWConditionVector = v;
    }

    public Vector getBWConditionVector () {
        return BWConditionVector;
    }

    public void setUserIDConditionVector (Vector v) {
        userIDConditionVector = v;
    }

    public Vector getUserIDConditionVector () {
        return userIDConditionVector;
    }

    public void setTypeConditionVector (Vector v) {
        typeConditionVector = v;
    }

    public Vector getTypeConditionVector () {
        return typeConditionVector;
    }

    public void setParameterConditionVector (Vector v) {
        parameterConditionVector = v;
    }

    public Vector getParameterConditionVector () {
        return parameterConditionVector;
    }
} // End of class Condition

```

```

/*****
File: CreateBWCondition.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/14/2001
Description: In this file, I will design the one of the 8 condition
types. By means of this window the user will be able to form policy
conditions by using BW.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreateBWCondition extends JDialog {
    private GuiMenu guiMenuForBWCondition;
    private Condition policyConditionForBW;

    /* I will set the BWCondition vector of condition class object (when
the user clicks on the "Band width" button in the policy condition
window) in this class. And other 7 vectors of the same object of the
condition class will be set in 7 different classes and then when the
user clicks the next button this condition class object will be put in
the policy class object. */

    private Vector tempVectorToSetBWConditionVector = new Vector (1);

    private OneBW oneBWObject = new OneBW ();

    private Vector relationVector = new Vector (1);

    private JLabel BWLabel, mbpsLabel, conditionBWAreLabel;
    private JLabel lineupLabel, lineupLabel2, lineupLabel3, lineupLabel4;
    private JComboBox relationComboBox;
    private JTextField BWValueTextField;
    private JButton addBWConditionToPolicyButton, okButton, cancelButton;
    private JList conditionBWList;

    /**
     * Method      : CreateBWCondition
     * Purpose      : Constructor for CreateBWCondition class
     * Parameters   : GuiMenu g, Condition c
     */

    public CreateBWCondition (GuiMenu g, Condition c ) {

        // Parent Frame is main GUI, and this JDialog is modal
        super ( g, "BW conditions", true );
        setLocation (115, 115);
        this.guiMenuForBWCondition = g;
        this.policyConditionForBW = c;
        createBWConditionGuiBuilderMethod ();
        setResizable (false);
        setSize(460, 350);
        show();
    } // End of constructor

```

```

/**
 * Method      : createBWConditionGuiBuilderMethod
 * Purpose     : This method builds the GUI for the creation of the
 *              BW conditions
 * Parameters  : None
 */
private void createBWConditionGuiBuilderMethod () {
    Container c = getContentPane ();
    c.setLayout( new FlowLayout () );

    BWLabel = new JLabel ("BW      ");
    c.add(BWLabel);

    CheckBoxHandler handler = new CheckBoxHandler ();

    relationVector.addElement(">=");
    relationVector.addElement("<=");

    relationComboBox = new JComboBox (relationVector);
    c.add(relationComboBox);
    relationComboBox.addItemListener(handler);

    oneBWObject.setIsGreaterThanOrEqual(true);
    oneBWObject.setIsSmallerThanOrEqual(false);

    BWValueTextField = new JTextField (10);
    c.add(BWValueTextField);

    mbpsLabel = new JLabel ("MBPS");
    c.add ( mbpsLabel );

    ButtonHandler handlerBut = new ButtonHandler ();
    addBWConditionToPolicyButton = new JButton ("Add to policy");
    addBWConditionToPolicyButton.addActionListener(handlerBut);
    c.add(addBWConditionToPolicyButton);

    lineupLabel = new JLabel ("                ");
    c.add(lineupLabel);

    conditionBWAreLabel = new JLabel ("BW conditions are:  ");
    c.add(conditionBWAreLabel);

    conditionBWList = new JList ();
    conditionBWList.setBackground(Color.lightGray);
    c.add ( new JScrollPane (conditionBWList) );

    lineupLabel2 = new JLabel ("                ");
    c.add(lineupLabel2);

    lineupLabel3 = new JLabel ("                ");
    c.add(lineupLabel3);

    okButton = new JButton (" OK  ");

```

```

        okButton.addActionListener(handlerBut);
        c.add(okButton);

        lineupLabel4 = new JLabel ("");
        c.add(lineupLabel4);

        cancelButton = new JButton ("Cancel");
        cancelButton.addActionListener(handlerBut);
        c.add(cancelButton);

    } // End of createBWConditionGuiBuilderMethod

/**
 * Class      : CheckBoxHandler
 * Purpose    : Inner class for combo box event handling
 */
private class CheckBoxHandler implements ItemListener {

    public void itemStateChanged( ItemEvent e ) {

        if (e.getSource() == relationComboBox) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {

                if ( ( (String) relationComboBox.getSelectedItem()
                    ).equals(">=") ) {
                    oneBWObject.setIsGreaterThanOrEqual(true);
                    oneBWObject.setIsSmallerThanOrEqual(false);
                } // End of if

                if ( ( (String) relationComboBox.getSelectedItem()
                    ).equals("<=") ) {
                    oneBWObject.setIsGreaterThanOrEqual(false);
                    oneBWObject.setIsSmallerThanOrEqual(true);
                } // End of if

            } // End of if ( e.getStateChange() == ItemEvent.SELECTED )

        } // End of relationComboBox if

    } // End of itemStateChanged method

} // End of class CheckBoxHandler

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for Add BW condition to policy, OK and
 * Cancel button event handling. When the user clicks add BW
 * condition to policy button, oneBWObject (which was set according
 * to the combo box and text field) is placed in the
 * tempVectorToSetBWConditionVector vector. When the user clicks ok
 * button, current temporary vector which has the BW conditions in
 * it will set the BW condition vector of the condition class object
 * or add its elements to it. When the user clicks on the cancel

```

```

    * button, time condition creation window will be dismissed.
    */

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == addBWConditionToPolicyButton ) {
            try {
                float temp = Float.parseFloat (
                    BWValueTextField.getText() );

                if ( temp >= 0 ) {
                    oneBWObject.setBWValue ( temp );

/* The reason why I create a new OneTimeObject is that each time I add
a new oneTime to the vector, I do not want the new oneTime to make the
old ones in the vector the same as itself. I use this new OneTime
object to put the copy (clone) of the OneTime object that comes from
the GUI. And then I add this new OneTime object to the vector. So, each
new OneTime object that comes from the GUI does not affect the ones
already in the vector. In other words, they are not referencing to the
same OneTime object anymore. */

                    OneBW oneBWDifferentObject = new OneBW ();
                    oneBWDifferentObject.copy(oneBWObject);

                    if ( !( tempVectorToSetBWConditionVector.contains
(oneBWDifferentObject) ) && !( policyConditionForBW.
getBWConditionVector()).contains(oneBWDifferentObject) ) {

                        tempVectorToSetBWConditionVector.addElement
(oneBWDifferentObject);
                        conditionBWList.setListData(
tempVectorToSetBWConditionVector ); // Display the selected OneBW
objects in the JList.
                        BWValueTextField.setText(""); // Text field is
cleared for the ease of use for the next time
                    }
                    else {
                        JOptionPane.showMessageDialog
(CreateBWCondition.this, "Create a different BW condition", "Different
BW condition needed", JOptionPane.ERROR_MESSAGE);
                    }

                } // End of if ( temp >= 0 )
                else { // If the BW is negative
                    JOptionPane.showMessageDialog (CreateBWCondition.this,
"BW value must be positive.", "Invalid BW value",
JOptionPane.ERROR_MESSAGE);
                } // End of else

            } // End of try

            catch (NumberFormatException nfe) {
                JOptionPane.showMessageDialog
(CreateBWCondition.this, "Please provide a float as a BW value.",
"Invalid number", JOptionPane.ERROR_MESSAGE);

```

```

        } // End of catch

    } // End of if ( e.getSource() ==
addTimeConditionToPolicyButton )

    if ( e.getSource() == okButton ) {
        if ( tempVectorToSetBWConditionVector.isEmpty() ) {
            JOptionPane.showMessageDialog (CreateBWCondition.this,
            "Please create a BW condition", " BW condition needed",
            JOptionPane.ERROR_MESSAGE);
        }
        else {
            if ( ( policyConditionForBW.getBWConditionVector()
).isEmpty() ) {

                policyConditionForBW.setBWConditionVector(
tempVectorToSetBWConditionVector);
            }
            else { // If the BW condition vector of the condition
class object is not empty, of course I do not want to set this vector
(since I will loose data in it). Instead, I will add new elements to
it.

                Enumeration enum =
tempVectorToSetBWConditionVector.elements();
                while ( enum.hasMoreElements() ) {
                    OneBW tempOneBW = new OneBW ();
                    tempOneBW = ( OneBW ) enum.nextElement();
                    ( policyConditionForBW.getBWConditionVector()
).addElement(tempOneBW);
                } // End of while
            } // End of else (when the BW condition vector of
conditon class object is not empty)

            CreateBWCondition.this.dispose();

        } // End of else

    } // End of if ( e.getSource() == okButton )

    if ( e.getSource() == cancelButton ) {
        CreateBWCondition.this.dispose();
    } // End of if ( e.getSource() == cancelButton )

} // End of method actionPerformed

} // End of class ButtonHandler

} // End of class CreateBWCondition

```

```

/*****
File: CreateClass.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/30/2001
Description: In this file, the frame that will be used to create a user
defined class will be formed.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreateClass extends JDialog {
    private DirectorClass boss;
    private Vector tempClassMemberVectorForEachClass = new Vector (1);
    private Vector a; // A Vector reference for Class class constructor
    private Class networkClass = new Class ("", a);

    private JLabel classNameLabel, classMemberLabel, membersAreLabel,
lineupLabel, lineupLabel2, lineupLabel3;
    private JTextField classNameTextField, classMemberTextField;
    private JButton addClassMemberButton, okButton, cancelButton;
    private JList classMembersList;

    private boolean hasWhiteSpace = false;

    /**
     * Method      : CreateClass
     * Purpose      : Constructor for CreateClass class
     * Parameters   : GuiMenu gui, DirectorClass x
     */

    public CreateClass ( GuiMenu gui, DirectorClass x ) {
        super ( gui, "Create Class", true ); // Parent Frame is main GUI,
and this JDialog is modal
        setLocation (115, 115);
        this.boss = x;
        createClassGuiBuilderMethod ( );
        setResizable (false);
        setSize(420, 350);
        show();
    } // End of constructor

    /**
     * Method      : createClassGuiBuilderMethod
     * Purpose      : This method builds the GUI for class creation.
     * Parameters   : None
     */

    private void createClassGuiBuilderMethod ( ) {
        Container c = getContentPane ( );
        c.setLayout( new FlowLayout ( ) );

        classNameLabel = new JLabel ("Class name

```



```

c.add(classNameLabel);

classNameTextField = new JTextField (10);
c.add(classNameTextField);

lineupLabel = new JLabel ("");
c.add(lineupLabel);

classMemberLabel = new JLabel ("Class member");
c.add(classMemberLabel);

classMemberTextField = new JTextField (10);
c.add(classMemberTextField);

ButtonHandler handlerBut = new ButtonHandler ();

addClassMemberButton = new JButton ("Add class member");
addClassMemberButton.addActionListener(handlerBut);
c.add(addClassMemberButton);

membersAreLabel = new JLabel ("Class members are: ");
c.add(membersAreLabel);

classMembersList = new JList ( );
classMembersList.setBackground(Color.lightGray);
c.add ( new JScrollPane (classMembersList) );

lineupLabel2 = new JLabel ("");
c.add(lineupLabel2);

okButton = new JButton (" OK ");
c.add(okButton);
okButton.addActionListener(handlerBut);

lineupLabel3 = new JLabel ("");
c.add(lineupLabel3);

cancelButton = new JButton ("Cancel");
c.add(cancelButton);
cancelButton.addActionListener(handlerBut);
} // End of createClassGuiBuilderMethod

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for Add class member to class, OK and
 * Cancel button event handling. This method checks the validity of
 * the class creation. For example: empty class name, empty class
 * member and white spaces are all checked. And after the checks
 * calls the addRecord method for adding the class and then destroys
 * the creation window.
 */

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == addClassMemberButton ) {

```

```

        if ( ( classMemberTextField.getText() ).equals("") ) {
            JOptionPane.showMessageDialog (CreateClass.this, "Please
provide a member name.", "Member name needed",
JOptionPane.ERROR_MESSAGE);
        }
        else if ( whitespaceControl (
classMemberTextField.getText() ) ) {
            JOptionPane.showMessageDialog (CreateClass.this, "White
spaces are not allowed in class member name", "White space warning",
JOptionPane.ERROR_MESSAGE);
        }
        else if ( tempClassMemberVectorForEachClass.contains(
classMemberTextField.getText() ) ) {
            JOptionPane.showMessageDialog (CreateClass.this, "Please
create different class members.", "Invalid class member",
JOptionPane.ERROR_MESSAGE);
        }
        else {
            tempClassMemberVectorForEachClass.addElement(
classMemberTextField.getText() );
            classMembersList.setListData
(tempClassMemberVectorForEachClass);
            classMemberTextField.setText(""); // clear the text
field to make it easy for user's editing the next class member (if
there are more than 1 members)
        }
    }

    if ( e.getSource() == okButton ) {
        if ( (classNameTextField.getText() ).equals("") ) {
            JOptionPane.showMessageDialog (CreateClass.this, "Please
provide a class name.", "Class name needed",
JOptionPane.ERROR_MESSAGE);
        }
        else if ( tempClassMemberVectorForEachClass.isEmpty() ) {
            JOptionPane.showMessageDialog (CreateClass.this, "Please
provide class member(s).", "Class member needed",
JOptionPane.ERROR_MESSAGE);
        }
        else if ( whitespaceControl ( classNameTextField.getText()
) ) {
            JOptionPane.showMessageDialog (CreateClass.this, "White
spaces are not allowed in class name", "White space warning",
JOptionPane.ERROR_MESSAGE);
        }
        else {
            addRecord();
        }
    } // End of if ( e.getSource() == okButton )

    if ( e.getSource() == cancelButton ) {
        CreateClass.this.dispose();
    } // End of if ( e.getSource() == cancelButton )

} // End of public void actionPerformed ( ActionEvent e)

```

```

/**
 * Method      : whiteSpaceControl
 * Purpose     : This method checks the strings that the user
 * creates to put them in a format that PPL compiler accepts. Class
 * name and member names should not contain white space characters.
 * This method returns true if a string has white space character,
 * false otherwise.
 * Parameters  : String s
 */

private boolean whiteSpaceControl (String s) {
    hasWhiteSpace = false;
    char charArray [] = s.toCharArray();
    for ( int i = 0; i < charArray.length; ++i ) {
        if ( Character.isWhitespace ( charArray[i] ) ) {
            hasWhiteSpace = true;
            break;
        } // End of if
    } // End of for
    if ( hasWhiteSpace == true) {
        return true;
    }
    else {
        return false;
    }
} // End of whiteSpaceControl Method

} // End of private class ButtonHandler implements ActionListener

/**
 * Method      : addRecord
 * Purpose     : This method adds a class to the class vector, if
 * that class is valid. Otherwise warns the policy maker about the
 * redefinition problem.
 * Parameters  : None
 */

private void addRecord () {
    networkClass.setName ( classNameTextField.getText() ); // set the
name of the class
    networkClass.setclassMembersVector
(tempClassMemberVectorForEachClass); // set the class member vector of
class

    // Newly created class is checked against the previous ones to
see if a redefinition occurs or not.
    if ( boss.classRedefinitionCheck (networkClass) ) { // method
classRedefinitionCheck returns true if there is no redefinition
        boss.addClass (networkClass); // add the class to class vector
in the Director class.
        this.dispose(); // Releases resources allocated for a context
(current instance of CreateClass class).
    }
}

```

```
        else {
            JOptionPane.showMessageDialog (this, "A class with the same
name already exists. Please change the name.", "Redefinition error",
JOptionPane.ERROR_MESSAGE);
        }

    } // End of public void addRecord ()

} // End of CreateClass class
```

```

/*****
File: CreateHopCountCondition.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/12/2001
Description: In this file, I will design the one of the 8 condition
types. By means of this window the user will be able to form policy
conditions by using hop count.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreateHopCountCondition extends JDialog {
    private GuiMenu guiMenuForHopCountCondition;
    private Condition policyConditionForHopCount;
    /* I will set the hop count condition vector of condition class
    object (when the user clicks on the "hop count" button in the policy
    condition window) in this class and other 7 vectors of the same object
    of the condition class will be set in 7 different classes. And then
    when the user clicks the next button this condition class object will
    be put in the policy class object. */

    private Vector tempVectorToSetHopCountConditionVector = new Vector
(1);

    private OneHopCount oneHopCountObject = new OneHopCount ();

    private Vector relationVector = new Vector (1);

    private JLabel hopCountLabel, conditionHopCountsAreLabel;
    private JLabel lineupLabel, lineupLabel2, lineupLabel3,
lineupLabel4;
    private JComboBox relationComboBox;
    private JTextField hopCountValueTextField;
    private JButton addHopCountConditionToPolicyButton, okButton,
cancelButton;
    private JList conditionHopCountList;

    /**
     * Method      : CreateHopCountCondition
     * Purpose      : Constructor for CreateHopCountCondition class
     * Parameters   : GuiMenu g, Condition c
     */

    public CreateHopCountCondition (GuiMenu g, Condition c ) {
        super ( g, "Hop count conditions", true ); // Parent Frame is
main GUI, and this JDialog is modal
        setLocation (115, 115);
        this.guiMenuForHopCountCondition = g;
        this.policyConditionForHopCount = c;
        createHopCountConditionGuiBuilderMethod ();
        setResizable (false);
        setSize(460, 350);
        show();
    }
}

```

```

} // End of constructor

/**
 * Method      : createHopCountConditionGuiBuilderMethod
 * Purpose     : This method builds the GUI for the creation of the
 * hop count conditions
 * Parameters  : None
 */

private void createHopCountConditionGuiBuilderMethod () {
    Container c = getContentPane ();
    c.setLayout( new FlowLayout () );

    hopCountLabel = new JLabel ("Hop Count");
    c.add(hopCountLabel);

    CheckBoxHandler handler = new CheckBoxHandler ();

    relationVector.addElement(">=");
    relationVector.addElement("<=");

    relationComboBox = new JComboBox (relationVector);
    c.add(relationComboBox);
    relationComboBox.addItemListener(handler);

    oneHopCountObject.setIsGreaterThanOrEqualTo(true);
    oneHopCountObject.setIsSmallerThanOrEqualTo(false);

    hopCountValueTextField = new JTextField (10);
    c.add(hopCountValueTextField);

    ButtonHandler handlerBut = new ButtonHandler ();
    addHopCountConditionToPolicyButton = new JButton ("Add to
policy");
    addHopCountConditionToPolicyButton.addActionListener(handlerBut);
    c.add(addHopCountConditionToPolicyButton);

    lineupLabel = new JLabel ("");
    c.add(lineupLabel);

    conditionHopCountsAreLabel = new JLabel ("Hop count conditions
are: ");
    c.add(conditionHopCountsAreLabel);

    conditionHopCountList = new JList ();
    conditionHopCountList.setBackground(Color.lightGray);
    c.add ( new JScrollPane (conditionHopCountList) );

    lineupLabel2 = new JLabel ("");
    c.add(lineupLabel2);

    lineupLabel3 = new JLabel ("");
    c.add(lineupLabel3);

```

```

        okButton = new JButton ( "   OK   " );
        okButton.addActionListener(handlerBut);
        c.add(okButton);

        lineupLabel4 = new JLabel ( "                                " );
        c.add(lineupLabel4);

        cancelButton = new JButton ("Cancel");
        cancelButton.addActionListener(handlerBut);
        c.add(cancelButton);

    } // End of createHopCountConditionGuiBuilderMethod

/**
 * Class          : CheckBoxHandler
 * Purpose        : Inner class for combo box event handling
 */

private class CheckBoxHandler implements ItemListener {

    public void itemStateChanged( ItemEvent e ) {

        if ( e.getSource() == relationComboBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {

                if ( ( (String) relationComboBox.getSelectedItem()
).equals(">=") ) {
                    oneHopCountObject.setIsGreaterThanOrEqualTo(true);
                    oneHopCountObject.setIsSmallerThanOrEqualTo(false);
                } // End of if

                if ( ( (String) relationComboBox.getSelectedItem()
).equals("<=") ) {
                    oneHopCountObject.setIsGreaterThanOrEqualTo(false);
                    oneHopCountObject.setIsSmallerThanOrEqualTo(true);
                } // End of if

            } // End of if ( e.getStateChange() == ItemEvent.SELECTED )

        } // End of relationComboBox if

    } // End of itemStateChanged method

} // End of class CheckBoxHandler

/**
 * Class          : ButtonHandler

```

```

* Purpose      : Inner class for Add hop count condition to policy,
* OK and Cancel button event handling. When the user clicks add hop
* count condition to policy button, one hopCount object (which was
* set according to the combo box and text field) is placed in the
* tempVectorToSetHopCountConditionVector vector. When the user
* clicks ok button, current temporary vector which has the hop
* count conditions in it will set the hop count condition vector of
* the condition class object or add its elements to it. When the
* user clicks on the cancel button, hop count condition creation
* window will be dismissed.
*/

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == addHopCountConditionToPolicyButton ) {
            try {
                int temp = Integer.parseInt (
hopCountValueTextField.getText() );
                if ( temp >= 0 ) {
                    oneHopCountObject.setHopCountValue ( temp );

/* The reason why I create a new OneHopCount object is that each time I
add a new oneHopCount to the vector, I do not want the new oneHopCount
to make the old ones in the vector the same as itself. I use this new
OneHopCount object to put the copy (clone) of the OneHopCount object
that comes from the GUI. And then I add this new OneHopCount object to
the vector. So, each new OneHopCount object that comes from the GUI
does not affect the ones already in the vector. In other words, they
are not referencing to the same OneHopCount object anymore. */

                    OneHopCount OneHopCountDifferentObject = new
OneHopCount ();

                    OneHopCountDifferentObject.copy(oneHopCountObject);

                    // If the policy maker did not add the same hop
count condition before.
                    if ( !(
tempVectorToSetHopCountConditionVector.contains(OneHopCountDifferentObj
ect) ) &&
                        !(
(policyConditionForHopCount.getHopCountConditionVector()).contains(OneH
opCountDifferentObject) ) ) {

tempVectorToSetHopCountConditionVector.addElement(OneHopCountDifferentO
bject);

                    conditionHopCountList.setListData(
tempVectorToSetHopCountConditionVector ); // Display the selected
OneHopCount objects in the JList.
                    hopCountValueTextField.setText(""); // Text
field is cleared for the ease of use for the next time
                }
                else {
                    JOptionPane.showMessageDialog
(CreateHopCountCondition.this, "Create a different hop count

```



```

condition", "Different hop count condition needed",
OptionPane.ERROR_MESSAGE);
    }

    } // End of if
    else { // If the hop count value is negative
        JOptionPane.showMessageDialog
(CreateHopCountCondition.this, "Hop count value must be positive.",
"Invalid hop count value", JOptionPane.ERROR_MESSAGE);
    } // End of else

    } // End of try
    catch (NumberFormatException nfe) {
        JOptionPane.showMessageDialog
(CreateHopCountCondition.this, "Please provide an integer as a hop
count value.", "Invalid number", JOptionPane.ERROR_MESSAGE);
    } // End of catch

    } // End of if ( e.getSource() ==
addHopCountConditionToPolicyButton )

    if ( e.getSource() == okButton ) {
        if ( tempVectorToSetHopCountConditionVector.isEmpty() ) {
            JOptionPane.showMessageDialog
(CreateHopCountCondition.this, "Please create a hop count condition", "
Hop count condition needed", JOptionPane.ERROR_MESSAGE);
        }
        else {
            if ( (
policyConditionForHopCount.getHopCountConditionVector() ).isEmpty() ) {

policyConditionForHopCount.setHopCountConditionVector(tempVectorToSetHo
pCountConditionVector);
            }
            else { // If the hop count condition vector of the
condition class object is not empty, of course I do not want to set
this vector (since I will loose data in it). Instead, I will add new
elements to it.
                Enumeration enum =
tempVectorToSetHopCountConditionVector.elements();
                while ( enum.hasMoreElements() ) {
                    OneHopCount tempOneHopCount = new OneHopCount ();
                    tempOneHopCount = ( OneHopCount )
enum.nextElement();
                    (
policyConditionForHopCount.getHopCountConditionVector()
).addElement(tempOneHopCount);
                } // End of while
            } // End of else (when the hop count condition vector of
conditon class object is not empty)

            CreateHopCountCondition.this.dispose();

        } // End of else
    }

```

```

    } // End of if ( e.getSource() == okButton )

    if ( e.getSource() == cancelButton ) {
        CreateHopCountCondition.this.dispose();
    } // End of if ( e.getSource() == cancelButton )

    } // End of method actionPerformed

    } // End of class BittonHandler

} // End of class CreateHopCountCondition

```

```

/*****
File: CreateLink.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/12/2001
Description: In this file, the frame that will be used to create a link
will be formed.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreateLink extends JDialog {
    private DirectorClass boss;
    private Node a; // A node reference for Link class constructor
    private Node b; // A node reference for Link class constructor
    private Link networkLink = new Link ("", a, b, 0, false, false,
false, false, false);

    private JLabel linkNameLabel, node1Label, node2Label,
linkParametersLabel, lineupLabel, lineupLabel2, bwUnitLabel;
    private JLabel lineupLabel3, lineupLabel4, lineupLabel5,
lineupLabel6, lineupLabel7, lineupLabel8;
    private JTextField linkNameTextField, bwValueTextField;
    private JComboBox node1ComboBox, node2ComboBox;
    private JCheckBox bwCheckBox, delayCheckBox, lossRateCheckBox,
jitterCheckBox, usedBwCheckBox;
    private JButton okButton, cancelButton;

    private boolean hasWhiteSpace = false;

    /**
     * Method      : CreateLink
     * Purpose      : Constructor for  class CreateLink
     * Parameters   : GuiMenu gui, DirectorClass x
     */

    public CreateLink ( GuiMenu gui, DirectorClass x ) {
        super ( gui, "Create Link", true ); // Parent Frame is main GUI,
and this JDialog is modal
        setLocation (115, 115);
        this.boss = x;
        createLinkGuiBuilderMethod ( );
        setResizable (false);
        setSize(455,400);
        show();
    } // End of constructor

    /**
     * Method      : createLinkGuiBuilderMethod

```

```

* Purpose      : This method builds the GUI for link creation.
* Parameters   : None
*/

private void createLinkGuiBuilderMethod ( ) {
    Container c = getContentPane ( );
    c.setLayout( new FlowLayout ( ) );

    linkNameLabel = new JLabel ( "          Link name          ");
    c.add(linkNameLabel);

    linkNameTextField = new JTextField (10);
    c.add(linkNameTextField);

    lineupLabel7 = new JLabel ( "          ");
    c.add(lineupLabel7);
    node1Label = new JLabel ("Node 1 of link");
    c.add(node1Label);

    lineupLabel8 = new JLabel ( "          ");
    c.add(lineupLabel8);

    node2Label = new JLabel ("Node 2 of link");
    c.add(node2Label);

    node1ComboBox = new JComboBox ( boss.getNodeVector() );
    node1ComboBox.setMaximumRowCount(7);
    c.add ( new JScrollPane (node1ComboBox) ); // Provide a scroll
bar if there are more than 7 nodes in the combo box.
    Dimension d = new Dimension (175, 25); // dimension object
    (width, height) to be used as the dimension of node combo box
    node1ComboBox.setPreferredSize(d); // set the size of the node
combo box so that it will not be too big or too small
    c.add ( node1ComboBox );

    lineupLabel = new JLabel ( "          ");
    c.add(lineupLabel);

    node2ComboBox = new JComboBox ( boss.getNodeVector() );
    node2ComboBox.setMaximumRowCount(7);
    c.add ( new JScrollPane (node2ComboBox) ); // Provide a scroll
bar if there are more than 7 nodes in the combo box.
    node2ComboBox.setPreferredSize(d); // set the size of the node
combo box so that it will not be too big or too small
    c.add ( node2ComboBox );

    linkParametersLabel = new JLabel ("Link parameters          ");
    c.add(linkParametersLabel);

    CheckBoxHandler handler = new CheckBoxHandler ( );

    bwCheckBox = new JCheckBox ("BW          ");
    c.add (bwCheckBox);
    bwCheckBox.addItemListener(handler);

    bwValueTextField = new JTextField (10);

```

```

bwValueTextField.setEditable(false);
c.add(bwValueTextField);

bwUnitLabel = new JLabel ("MBPS");
c.add(bwUnitLabel);

lineupLabel3 = new JLabel ("");
c.add(lineupLabel3);

delayCheckBox = new JCheckBox ("Delay ( )");
c.add (delayCheckBox);
delayCheckBox.addItemListener(handler);

lineupLabel4 = new JLabel ("");
c.add(lineupLabel4);

lossRateCheckBox = new JCheckBox ("Loss Rate ( )");
c.add (lossRateCheckBox);
lossRateCheckBox.addItemListener(handler);

jitterCheckBox = new JCheckBox ("Jitter ( )");
c.add (jitterCheckBox);
jitterCheckBox.addItemListener(handler);

usedBwCheckBox = new JCheckBox ("Used BW ( )");
c.add (usedBwCheckBox);
usedBwCheckBox.addItemListener(handler);

lineupLabel5 = new JLabel ("");
c.add(lineupLabel5);

ButtonHandler handlerBut = new ButtonHandler ();

okButton = new JButton (" OK ");
c.add(okButton);
okButton.addActionListener(handlerBut);

lineupLabel6 = new JLabel ("");
c.add(lineupLabel6);

cancelButton = new JButton ("Cancel");
c.add(cancelButton);
cancelButton.addActionListener(handlerBut);
} // End of createLinkGuiBuilderMethod

```

```

/**
 * Class : CheckBoxHandler

```

```

* Purpose      : Inner class for combo box event handling
*/

private class CheckBoxHandler implements ItemListener {
    public void itemStateChanged( ItemEvent e )
    {
        if ( e.getSource() == bwCheckBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                bwValueTextField.setEditable(true);
                networkLink.setHasBwParameter(true);
            }

            else {
                bwValueTextField.setEditable(false);
                /* If the user deselects the BW parameter check box,
then the value in the bw value text field is cleared. */
                bwValueTextField.setText("");
                networkLink.setHasBwParameter(false);
                networkLink.setBwValue(0);
            }
        }

        if ( e.getSource() == delayCheckBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                networkLink.setHasDelayParameter(true);
            }

            else {
                networkLink.setHasDelayParameter(false);
            }
        }

        if ( e.getSource() == lossRateCheckBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                networkLink.setHasLossRateParameter(true);
            }

            else {
                networkLink.setHasLossRateParameter(false);
            }
        }

        if ( e.getSource() == jitterCheckBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                networkLink.setHasJitterParameter(true);
            }

            else {
                networkLink.setHasJitterParameter(false);
            }
        }

        if ( e.getSource() == usedBwCheckBox ) {

```

```

        if ( e.getStateChange() == ItemEvent.SELECTED ) {
            networkLink.setHasUsedBwParameter(true);
        }

        else {
            networkLink.setHasUsedBwParameter(false);
        }
    }

    } //end of method public void itemStateChanged (ItemEvent e)

} //end of inner class CheckBoxHandler

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for OK and Cancel button event
 * handling.  This method checks the validity of the link creation.
 * For example, empty link name, empty BW value, invalid BW value.
 * And after the checks calls the addRecord method for adding the
 * link and then destroys the creation window.
 */

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == okButton ) {
            if ( (linkNameTextField.getText() ).equals("") ) {
                JOptionPane.showMessageDialog (CreateLink.this, "Please
provide a link name.", "Link name needed", JOptionPane.ERROR_MESSAGE);
            }
            else if ( whiteSpaceControl ( linkNameTextField.getText() )
) {
                JOptionPane.showMessageDialog (CreateLink.this, "White
spaces are not allowed in link name", "White space warning",
JOptionPane.ERROR_MESSAGE);
            }
            else if ( ( (Node) node1ComboBox.getSelectedItem() == null
) && ( (Node) node2ComboBox.getSelectedItem() == null) ) {
                JOptionPane.showMessageDialog (CreateLink.this, "Please
create nodes.", "Node creation required", JOptionPane.ERROR_MESSAGE);
            }
            else if ( (Node) node1ComboBox.getSelectedItem() == (Node)
node2ComboBox.getSelectedItem() ) {
                JOptionPane.showMessageDialog (CreateLink.this, "Nodes
cannot be the same.", "Different nodes needed",
JOptionPane.ERROR_MESSAGE);
            }
            else if ( ( networkLink.getHasBwParameter () ) && (
(bwValueTextField.getText() ).equals("") ) ) {
                JOptionPane.showMessageDialog (CreateLink.this, "Please
provide a BW value.", "BW value needed", JOptionPane.ERROR_MESSAGE);
            }

            else if ( ( networkLink.getHasBwParameter () ) ) {
                try {

```

```

        float temp = Float.parseFloat (
bwValueTextField.getText() );
        if ( temp >= 0 ) {
            addRecord ();
        } // End of if
        else { // If the BW value is negative
            JOptionPane.showMessageDialog (CreateLink.this,
"BW value must be positive.", "Invalid BW value",
JOptionPane.ERROR_MESSAGE);
        } // End of else
    } // End of try
    catch (NumberFormatException nfe) {
        JOptionPane.showMessageDialog (CreateLink.this,
"Please provide a number as a Bandwidth value.", "Invalid number",
JOptionPane.ERROR_MESSAGE);
    } // End of catch

    } // End of else if

    else {
        addRecord();
    }

} // End of if ( e.getSource() == okButton )

if ( e.getSource() == cancelButton ) {
    CreateLink.this.dispose();
} // End of if ( e.getSource() == cancelButton )

} // End of public void actionPerformed ( ActionEvent e)

/**
 * Method      : whiteSpaceControl
 * Purpose     : This method checks the strings that the user
 * creates to put them in a format that PPL compiler accepts. Link
 * name should not contain white space characters. This method
 * returns true if a string has white space character, false
 * otherwise.
 * Parameters  : String s
 */

private boolean whiteSpaceControl (String s) {
    hasWhiteSpace = false;
    char charArray [] = s.toCharArray();
    for ( int i = 0; i < charArray.length; ++i ) {
        if ( Character.isWhitespace ( charArray[i] ) ) {
            hasWhiteSpace = true;
            break;
        } // End of if
    } // End of for
    if ( hasWhiteSpace == true) {
        return true;
    }
    else {
        return false;
    }
}

```



```

        }
    } // End of whiteSpaceControl Method

} // End of private class ButtonHandler implements ActionListener

/**
 * Method      : addRecord
 * Purpose     : This method adds a link to the link vector, if that
 * link is valid. Otherwise warns the policy maker about the
 * redefinition problem.
 * Parameters  : None
 */

private void addRecord () {
    networkLink.setName ( linkNameTextField.getText() );
    networkLink.setNode1 ( (Node) node1ComboBox.getSelectedItem() );
    networkLink.setNode2 ( (Node) node2ComboBox.getSelectedItem() );

    if ( networkLink.getHasBwParameter () ) {
        networkLink.setBwValue( Float.parseFloat (
bwValueTextField.getText() ) );
    }
    else {
        networkLink.setBwValue (0);
    }

    // Newly created link is checked against the previous ones to see
    if a redefinition occurs or not.
    if ( boss.linkRedefinitionCheck (networkLink) ) { // method
linkRedefinitionCheck returns true if there is no redefinition
        boss.addLink(networkLink);
        this.dispose(); // Releases resources allocated for a context
(current instance of CreateLink class).
    }
    else {
        JOptionPane.showMessageDialog (this, "A link with the same
name already exists. Please change the name.", "Redefinition error",
JOptionPane.ERROR_MESSAGE);
    }

} // End of public void addRecord ()

} // End class CreateLink

```

```

/*****
File: CreateNode.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/12/2001
Description: In this file, the window that will be used to create a
node will be formed.
*****/

```

```

import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

```

```

public class CreateNode extends JDialog {
    private DirectorClass boss;
    private Node networkNode = new Node ();

    private JLabel nodeDomainLabel, nodeNameLabel, nodeParametersLabel,
lineupLabel, lineupLabel2, bwUnitLabel;
    private JLabel lineupLabel3, lineupLabel4, lineupLabel5,
lineupLabel6, lineupLabel7, lineupLabel8;
    private JTextField nodeDomainTextField, nodeNameTextField,
bwValueTextField;
    private JCheckBox bwCheckBox, delayCheckBox, lossRateCheckBox,
jitterCheckBox, usedBwCheckBox;
    private JButton okButton, cancelButton;

    private boolean hasWhiteSpace = false;

    /**
     * Method      : CreateNode
     * Purpose      : Constructor for class CreateNode
     * Parameters   : GuiMenu gui, DirectorClass x
     */

    public CreateNode ( GuiMenu gui, DirectorClass x ) {
        super ( gui, "Create Node", true ); // Parent Frame is main GUI,
and this JDialog is modal
        /* Following line of code Moves this component to a new location.
The top-left corner of the new location is specified by the x and y
parameters in the coordinate space of this component's parent. */
        setLocation (115, 115);
        this.boss = x;
        createNodeGuiBuilderMethod ( );
        setResizable (false);
        setSize(455,370);
        show();
    } // End of constructor

```

```

/**
 * Method      : createNodeGuiBuilderMethod

```

```

* Purpose      : This method builds the GUI for node creation.
* Parameters   : None
*/

private void createNodeGuiBuilderMethod ( ) {
    Container c = getContentPane ( );
    c.setLayout( new FlowLayout ( ) );

    nodeNameLabel = new JLabel ( "          Node name          ");
    c.add(nodeNameLabel);

    nodeNameTextField = new JTextField (10);
    c.add(nodeNameTextField);

    lineupLabel7 = new JLabel ( "          ");
    c.add(lineupLabel7);

    lineupLabel8 = new JLabel ( "          ");
    c.add(lineupLabel8);

    nodeDomainLabel = new JLabel ( "          Node domain          ");
    nodeDomainLabel.setToolTipText("Enter the name of the domain that
the node belongs to." );
    c.add(nodeDomainLabel);

    nodeDomainTextField = new JTextField (10);
    c.add(nodeDomainTextField);

    lineupLabel = new JLabel ( "          ");
    c.add(lineupLabel);

    lineupLabel2 = new JLabel ( "          ");
    c.add(lineupLabel2);

    nodeParametersLabel = new JLabel ("Node parameters          ");
    c.add(nodeParametersLabel);

    CheckBoxHandler handler = new CheckBoxHandler ( );

    bwCheckBox = new JCheckBox ("BW          ");
    c.add (bwCheckBox);
    bwCheckBox.addItemListener(handler);

    bwValueTextField = new JTextField (10);
    bwValueTextField.setEditable(false);
    c.add(bwValueTextField);

    bwUnitLabel = new JLabel ("MBPS          ");
    c.add(bwUnitLabel);

    lineupLabel3 = new JLabel ( "          ");
    c.add(lineupLabel3);

    delayCheckBox = new JCheckBox ("Delay ( )          ");
    c.add (delayCheckBox);
    delayCheckBox.addItemListener(handler);

```

```

        lineupLabel4 = new JLabel ( "                                ");
        c.add(lineupLabel4);

        lossRateCheckBox = new JCheckBox ("Loss Rate ( )                                ");
        c.add (lossRateCheckBox);
        lossRateCheckBox.addItemListener(handler);

        jitterCheckBox = new JCheckBox ("Jitter ( )                                ");
        c.add (jitterCheckBox);
        jitterCheckBox.addItemListener(handler);

        usedBwCheckBox = new JCheckBox ("Used BW ( )                                ");
        c.add (usedBwCheckBox);
        usedBwCheckBox.addItemListener(handler);

        lineupLabel5 = new JLabel ( "                                ");
        c.add(lineupLabel5);

        ButtonHandler handlerBut = new ButtonHandler ();

        okButton = new JButton ( "    OK    ");
        c.add(okButton);
        okButton.addActionListener(handlerBut);

        lineupLabel6 = new JLabel ( "                                ");
        c.add(lineupLabel6);

        cancelButton = new JButton ("Cancel");
        c.add(cancelButton);
        cancelButton.addActionListener(handlerBut);
    } // End of createNodeGuiBuilderMethod

/**
 * Class          : CheckBoxHandler
 * Purpose        : Inner class for combo box event handling
 */

private class CheckBoxHandler implements ItemListener {
    public void itemStateChanged( ItemEvent e )
    {
        if ( e.getSource() == bwCheckBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                bwValueTextField.setEditable(true);
                networkNode.setHasBwParameter(true);
            }

            else {
                bwValueTextField.setEditable(false);
                // If the user deselects the BW parameter check box,
then the value in the bw value text field
                // is cleared.
                bwValueTextField.setText("");
                networkNode.setHasBwParameter(false);
                networkNode.setBwValue(0);
            }
        }
    }
}

```

```

    }
}

if ( e.getSource() == delayCheckBox ) {
    if ( e.getStateChange() == ItemEvent.SELECTED ) {
        networkNode.setHasDelayParameter(true);
    }

    else {
        networkNode.setHasDelayParameter(false);
    }
}

if ( e.getSource() == lossRateCheckBox ) {
    if ( e.getStateChange() == ItemEvent.SELECTED ) {
        networkNode.setHasLossRateParameter(true);
    }

    else {
        networkNode.setHasLossRateParameter(false);
    }
}

if ( e.getSource() == jitterCheckBox ) {
    if ( e.getStateChange() == ItemEvent.SELECTED ) {
        networkNode.setHasJitterParameter(true);
    }

    else {
        networkNode.setHasJitterParameter(false);
    }
}

if ( e.getSource() == usedBwCheckBox ) {
    if ( e.getStateChange() == ItemEvent.SELECTED ) {
        networkNode.setHasUsedBwParameter(true);
    }

    else {
        networkNode.setHasUsedBwParameter(false);
    }
}

} //end of method public void itemStateChanged (ItemEvent e)
} //end of inner class CheckBoxHandler

```

```

/**
 * Class      : ButtonHandler

```

```

* Purpose      : Inner class for OK and Cancel button event
* handling. This method checks the validity of the node creation.
* For example, empty domain name, empty node name, empty BW value,
* invalid BW value. And after the checks calls the addRecord method
* for adding the node and then destroys the creation window.
*/

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == okButton ) {
            if ( (nodeNameTextField.getText() ).equals("") ) {
                JOptionPane.showMessageDialog (CreateNode.this, "Please
provide a node name.", "Node name needed", JOptionPane.ERROR_MESSAGE);
            }
            else if ( whiteSpaceControl ( nodeNameTextField.getText() )
) {
                JOptionPane.showMessageDialog (CreateNode.this, "White
spaces are not allowed in node name", "White space warning",
JOptionPane.ERROR_MESSAGE);
            }
            else if ( (nodeDomainTextField.getText() ).equals("") ) {
                JOptionPane.showMessageDialog (CreateNode.this, "Please
provide a domain name.", "Domain name needed",
JOptionPane.ERROR_MESSAGE);
            }
            else if ( whiteSpaceControl ( nodeDomainTextField.getText()
) ) {
                JOptionPane.showMessageDialog (CreateNode.this, "White
spaces are not allowed in domain name", "White space warning",
JOptionPane.ERROR_MESSAGE);
            }
            else if ( ( networkNode.getHasBwParameter () ) && (
bwValueTextField.getText() ).equals("") ) ) {
                JOptionPane.showMessageDialog (CreateNode.this, "Please
provide a BW value.", "BW value needed", JOptionPane.ERROR_MESSAGE);
            }
            else if ( ( networkNode.getHasBwParameter () ) ) {
                try {
                    float temp = Float.parseFloat (
bwValueTextField.getText() );
                    if ( temp >= 0 ) {
                        addRecord ();
                    } // End of if
                    else { // If the BW value is negative
                        JOptionPane.showMessageDialog (CreateNode.this,
"BW value must be positive.", "Invalid BW value",
JOptionPane.ERROR_MESSAGE);
                    } // End of else
                } // End of try
                catch (NumberFormatException nfe) {
                    JOptionPane.showMessageDialog (CreateNode.this,
"Please provide a number as a Bandwidth value.", "Invalid number",
JOptionPane.ERROR_MESSAGE);
                } // End of catch

            } // End of else if
        }
    }
}

```

```

        else {
            addRecord();
        }

    } // End of if ( e.getSource() == okButton )

    if ( e.getSource() == cancelButton ) {
        CreateNode.this.dispose();
    } // End of if ( e.getSource() == cancelButton )

} // End of public void actionPerformed ( ActionEvent e)

/**
 * Method      : whiteSpaceControl
 * Purpose     : This method checks the strings that the user
 * creates to put them in a format that PPL compiler accepts. Node
 * name and node domain name should not contain white space
 * characters. This method returns true if a string has white space
 * character, false otherwise.
 */

private boolean whiteSpaceControl (String s) {
    hasWhiteSpace = false;
    char charArray [] = s.toCharArray();
    for ( int i = 0; i < charArray.length; ++i ) {
        if ( Character.isWhitespace ( charArray[i] ) ) {
            hasWhiteSpace = true;
            break;
        } // End of if
    } // End of for
    if ( hasWhiteSpace == true) {
        return true;
    }
    else {
        return false;
    }
} // End of whiteSpaceControl Method

} // End of private class ButtonHandler implements ActionListener

/**
 * Method      : addRecord
 * Purpose     : This method adds a node to the node vector, if that
 * node is valid. Otherwise warns the policy maker about the
 * redefinition problem
 * Parameters  : None
 */

private void addRecord () {

    networkNode.setDomain ( nodeDomainTextField.getText() );
    networkNode.setName ( nodeNameTextField.getText() );

```

```

        if ( networkNode.getHasBwParameter () ) {
            networkNode.setBwValue( Float.parseFloat (
bwValueTextField.getText() ) );
        }
        else {
            networkNode.setBwValue (0);
        }

        // Newly created node is checked against the previous ones to see
        if a redefinition occurs or not.
        if ( boss.nodeRedefinitionCheck (networkNode) ) { // method
nodeRedefinitionCheck returns true if there is no redefinition
            boss.addNode(networkNode);
            this.dispose(); // Releases resources allocated for a context
(current instance of CreateNode class).
        }
        else {
            JOptionPane.showMessageDialog (this, "A node with the same
domain and name already exists. Please change one of them.",
"Redefinition error", JOptionPane.ERROR_MESSAGE);
        }

    } // End of public void addRecord ()

} // end of class CreateNode

```



```

/*****
File: CreateParameterCondition.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/14/2001
Description: In this file, I will design the one of the 8 condition
types. By means of this window
the user will be able to form policy conditions by using a network
parameter.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreateParameterCondition extends JDialog {

    private GuiMenu guiMenuForParameterCondition;
    private Condition policyConditionForParameter;

    private boolean networkHasDelayParameterInGui;
    private boolean networkHasLossRateParameterInGui;
    private boolean networkHasJitterParameterInGui;
    private boolean networkHasUsedBWParameterInGui;

    /* I will set the parameterCondition vector of condition class
    object (when the user clicks on the "parameter" button in the policy
    condition window) in this class. And other 7 vectors of the same object
    of the condition class will be set in 7 different classes and then when
    the user clicks the next button this condition class object will be put
    in the policy class object. */

    private Vector tempVectorToSetParameterConditionVector = new Vector
(1);

    private OneParameter oneParameterObject = new OneParameter ();

    private Vector relationVector = new Vector (1);

    private Vector parameterVector = new Vector (1);

    private JLabel parameterLabel, conditionParametersAreLabel;
    private JLabel lineupLabel, lineupLabel2, lineupLabel3;
    private JComboBox parameterComboBox, relationComboBox;
    private JTextField parameterValueTextField, unitTextField;
    private JButton addParameterConditionToPolicyButton, okButton,
cancelButton;
    private JList conditionParameterList;

    /**
     * Method      : CreateParameterCondition

```

```

* Purpose      : Constructor for CreateParameterCondition class
* Parameters   : GuiMenu g, Condition c, boolean t, boolean u,
*               boolean f, boolean a
*/

public CreateParameterCondition (GuiMenu g, Condition c, boolean t,
boolean u, boolean f, boolean a ) {
    super ( g, "Parameter conditions", true ); // Parent Frame is
main GUI, and this JDialog is modal
    setLocation (115, 115);
    this.guiMenuForParameterCondition = g;
    this.policyConditionForParameter = c;
    this.networkHasDelayParameterInGui = t;
    this.networkHasLossRateParameterInGui = u;
    this.networkHasJitterParameterInGui = f;
    this.networkHasUsedBWParameterInGui = a;

    createParameterConditionGuiBuilderMethod ();
    setResizable (false);
    setSize(460, 350);
    show();
} // End of constructor

/**
* Method      : createParameterConditionGuiBuilderMethod
* Purpose     : This method builds the GUI for the creation of the
*               parameter conditions
* Parameters   : None
*/

private void createParameterConditionGuiBuilderMethod () {
    Container c = getContentPane ();
    c.setLayout( new FlowLayout () );

    CheckBoxHandler handler = new CheckBoxHandler ();

    if ( networkHasDelayParameterInGui ) {
        parameterVector.addElement("delay ()");
    }
    if ( networkHasLossRateParameterInGui ) {
        parameterVector.addElement("loss rate ()");
    }
    if ( networkHasJitterParameterInGui ) {
        parameterVector.addElement("jitter ()");
    }
    if ( networkHasUsedBWParameterInGui ) {
        parameterVector.addElement("used bw ()");
    }

    parameterComboBox = new JComboBox (parameterVector);
    c.add (parameterComboBox);
    oneParameterObject.setParameterName ( (String) (
parameterComboBox.getSelectedItem() ) );
    parameterComboBox.addItemListener(handler);
}

```

```

relationVector.addElement("==");
relationVector.addElement("!=");
relationVector.addElement(">=");
relationVector.addElement("<=");
relationVector.addElement(">");
relationVector.addElement("<");

relationComboBox = new JComboBox (relationVector);
c.add(relationComboBox);
relationComboBox.addItemListener(handler);

oneParameterObject.setIsEqual(true);
oneParameterObject.setIsNotEqual(false);
oneParameterObject.setIsGreaterThanOrEqual(false);
oneParameterObject.setIsSmallerThanOrEqual(false);
oneParameterObject.setIsGreater(false);
oneParameterObject.setIsSmaller(false);

parameterValueTextField = new JTextField (10);
c.add(parameterValueTextField);

unitTextField = new JTextField ("msec", 4);
unitTextField.setEditable(false);
unitTextField.setBackground(Color.lightGray);
c.add(unitTextField);

ButtonHandler handlerBut = new ButtonHandler ();
addParameterConditionToPolicyButton = new JButton ("Add to
policy");

addParameterConditionToPolicyButton.addActionListener(handlerBut);
c.add(addParameterConditionToPolicyButton);

lineupLabel = new JLabel ("");
c.add(lineupLabel);

conditionParametersAreLabel = new JLabel ("Parameter conditions
are: ");
c.add(conditionParametersAreLabel);

conditionParameterList = new JList ();
conditionParameterList.setBackground(Color.lightGray);

c.add ( new JScrollPane (conditionParameterList) );

lineupLabel2 = new JLabel ("");
c.add(lineupLabel2);

okButton = new JButton (" OK ");
okButton.addActionListener(handlerBut);
c.add(okButton);

lineupLabel3 = new JLabel ("");
c.add(lineupLabel3);

```

```

        cancelButton = new JButton ("Cancel");
        cancelButton.addActionListener(handlerBut);
        c.add(cancelButton);

    } // End of createBWConditionGuiBuilderMethod

    /**
     * Class      : CheckBoxHandler
     * Purpose    : Inner class for combo box event handling
     */

    private class CheckBoxHandler implements ItemListener {

        public void itemStateChanged( ItemEvent e ) {

            if (e.getSource() == parameterComboBox) {

                if ( e.getStateChange() == ItemEvent.SELECTED ) {

                    if ( ( (String) parameterComboBox.getSelectedItem()
).equals("delay ()") ) {
                        oneParameterObject.setParameterName("delay ()");
                        unitTextField.setText("msec");
                    }
                    if ( ( (String) parameterComboBox.getSelectedItem()
).equals("loss rate ()") ) {
                        oneParameterObject.setParameterName("loss rate ()");
                        unitTextField.setText("%");
                    }
                    if ( ( (String) parameterComboBox.getSelectedItem()
).equals("jitter ()") ) {
                        oneParameterObject.setParameterName("jitter ()");
                        unitTextField.setText("msec");
                    }
                    if ( ( (String) parameterComboBox.getSelectedItem()
).equals("used bw ()") ) {
                        oneParameterObject.setParameterName("used bw ()");
                        unitTextField.setText("MBPS");
                    }
                }

            } // End of if ( e.getStateChange() == ItemEvent.SELECTED )

        } // End of if (e.getSource() == parameterComboBox)

        if (e.getSource() == relationComboBox) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {

                if ( ( (String) relationComboBox.getSelectedItem()
).equals("==") ) {
                    oneParameterObject.setIsEqual(true);
                    oneParameterObject.setIsNotEqual(false);
                    oneParameterObject.setIsGreaterThanOrEqual(false);
                    oneParameterObject.setIsSmallerThanOrEqual(false);

```

```

        oneParameterObject.setIsGreater(false);
        oneParameterObject.setIsSmaller(false);
    } // End of if

    if ( ( (String) relationComboBox.getSelectedItemAt()
).equals("!=") ) {
        oneParameterObject.setIsEqual(false);
        oneParameterObject.setIsNotEqual(true);
        oneParameterObject.setIsGreaterThanOrEqual(false);
        oneParameterObject.setIsSmallerThanOrEqual(false);
        oneParameterObject.setIsGreater(false);
        oneParameterObject.setIsSmaller(false);
    } // End of if

    if ( ( (String) relationComboBox.getSelectedItemAt()
).equals(">=") ) {
        oneParameterObject.setIsEqual(false);
        oneParameterObject.setIsNotEqual(false);
        oneParameterObject.setIsGreaterThanOrEqual(true);
        oneParameterObject.setIsSmallerThanOrEqual(false);
        oneParameterObject.setIsGreater(false);
        oneParameterObject.setIsSmaller(false);
    } // End of if

    if ( ( (String) relationComboBox.getSelectedItemAt()
).equals("<=") ) {
        oneParameterObject.setIsEqual(false);
        oneParameterObject.setIsNotEqual(false);
        oneParameterObject.setIsGreaterThanOrEqual(false);
        oneParameterObject.setIsSmallerThanOrEqual(true);
        oneParameterObject.setIsGreater(false);
        oneParameterObject.setIsSmaller(false);
    } // End of if

    if ( ( (String) relationComboBox.getSelectedItemAt()
).equals(">") ) {
        oneParameterObject.setIsEqual(false);
        oneParameterObject.setIsNotEqual(false);
        oneParameterObject.setIsGreaterThanOrEqual(false);
        oneParameterObject.setIsSmallerThanOrEqual(false);
        oneParameterObject.setIsGreater(true);
        oneParameterObject.setIsSmaller(false);
    } // End of if

    if ( ( (String) relationComboBox.getSelectedItemAt()
).equals("<") ) {
        oneParameterObject.setIsEqual(false);
        oneParameterObject.setIsNotEqual(false);
        oneParameterObject.setIsGreaterThanOrEqual(false);
        oneParameterObject.setIsSmallerThanOrEqual(false);
        oneParameterObject.setIsGreater(false);
        oneParameterObject.setIsSmaller(true);
    } // End of if

} // End of if ( e.getStateChange() == ItemEvent.SELECTED )

```

```

        } // End of relationComboBox if

    } // End of itemStateChanged method

} // End of class CheckBoxHandler

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for Add parameter condition to policy,
 * OK and Cancel button event handling. When the user clicks add
 * parameter condition to policy button, oneParameterObject (which
 * was set according to the combo boxes and text field) is
 * placed in the tempVectorToSetParameterConditionVector vector.
 * When the user clicks ok button, current temporary vector which
 * has the parameter conditions in it will set the parameter
 * condition vector of the condition class object or add its
 * elements to it. When the user clicks on the cancel button, time
 * condition creation window will be dismissed.
 */

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == addParameterConditionToPolicyButton ) {
            try {
                float temp = Float.parseFloat (
parameterValueTextField.getText() );

                if ( !( (String) (
parameterComboBox.getSelectedItemAt() ) ).equals("loss rate ()") ) &&
temp < 0 ) {
                    JOptionPane.showMessageDialog
(CreateParameterCondition.this, "Parameter value must be positive.",
"Invalid parameter value", JOptionPane.ERROR_MESSAGE);
                }
                else if ( (String) (
parameterComboBox.getSelectedItemAt() ) ).equals("loss rate ()") && temp
< 0 ) {
                    JOptionPane.showMessageDialog
(CreateParameterCondition.this, "Loss rate () parameter value must be
between 0-100.", "Invalid loss rate () parameter value",
JOptionPane.ERROR_MESSAGE);
                }
                else if ( (String) (
parameterComboBox.getSelectedItemAt() ) ).equals("loss rate ()") && temp
> 100 ) {
                    JOptionPane.showMessageDialog
(CreateParameterCondition.this, "Loss rate () parameter value must be
between 0-100.", "Invalid loss rate () parameter value",
JOptionPane.ERROR_MESSAGE);
                }
                else {
                    oneParameterObject.setParameterValue(temp);

```

```

/* The reason why I create a new OneParameterObject is that each time I
add a new oneParameter to the vector, I do not want the new
oneParameter to make the old ones in the vector the same as itself. I
use this new OneParameter object to put the copy (clone) of the
OneParameter object that comes from the GUI. And then I add this new
OneParameter object to the vector. So, each new OneParameter object
that comes from the GUI does not affect the ones already in the vector.
In other words, they are not referencing to the same OneTime object
anymore. */

```

```

        OneParameter oneParameterDifferentObject = new
OneParameter ();
        oneParameterDifferentObject.copy(oneParameterObject);

        if ( !(
tempVectorToSetParameterConditionVector.contains(oneParameterDifferentO
bject) ) &&
                !(
(policyConditionForParameter.getParameterConditionVector()).contains(on
eParameterDifferentObject) ) ) {

tempVectorToSetParameterConditionVector.addElement(oneParameterDifferen
tObject);
                conditionParameterList.setListData(
tempVectorToSetParameterConditionVector ); // Display the selected
OneParameter objects in the JList.
                parameterValueTextField.setText(""); // Text field
is cleared for the ease of use for the next time
        }
        else {
                JOptionPane.showMessageDialog
(CreateParameterCondition.this, "Create a different parameter
condition", "Different parameter condition needed",
JOptionPane.ERROR_MESSAGE);
        }

        } // End of else

    } // End of try

    catch (NumberFormatException nfe) {
        JOptionPane.showMessageDialog
(CreateParameterCondition.this, "Please provide a float as a parameter
value.", "Invalid number", JOptionPane.ERROR_MESSAGE);
    } // End of catch

    } // End of if ( e.getSource() ==
addParameterConditionToPolicyButton )

    if ( e.getSource() == okButton ) {
        if ( tempVectorToSetParameterConditionVector.isEmpty() ) {
            JOptionPane.showMessageDialog
(CreateParameterCondition.this, "Please create a parameter condition",
" Parameter condition needed", JOptionPane.ERROR_MESSAGE);
        }
    }

```

```

        else {
            if ( (
policyConditionForParameter.getParameterConditionVector() ).isEmpty() )
{

policyConditionForParameter.setParameterConditionVector(tempVectorToSet
ParameterConditionVector);
            }
            else { // If the parameter condition vector of the
condition class object is not empty, of course I do not want to set
this vector (since I will loose data in it). Instead, I will add new
elements to it.
                Enumeration enum =
tempVectorToSetParameterConditionVector.elements();
                while ( enum.hasMoreElements() ) {
                    OneParameter tempOneParameter = new OneParameter
();
                    tempOneParameter = ( OneParameter )
enum.nextElement();
                    (
policyConditionForParameter.getParameterConditionVector()
).addElement(tempOneParameter);
                } // End of while
            } // End of else (when the parameter condition vector of
conditon class object is not empty)

                CreateParameterCondition.this.dispose();

            } // End of else

        } // End of if ( e.getSource() == okButton )

        if ( e.getSource() == cancelButton ) {
            CreateParameterCondition.this.dispose();
        } // End of if ( e.getSource() == cancelButton )

    } // End of method actionPerformed

} // End of class ButtonHandler

} // End of class CreateParameterCondition

```



```

/*****
File: CreatePath.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/12/2001
Description: In this file, the frame that will be used to create a path
will be formed.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreatePath extends JDialog {
    private DirectorClass boss;
    private Vector a; // A Vector reference for Path class constructor
    private Path networkPath = new Path ("", a, 0, false, false, false,
false, false);
    private Vector tempNodeVectorForEachPath = new Vector (1); // This
vector will be used to set the path nodes vector of the path class to
make each path know the nodes that it contains

    private Node wildNode = new Node ("", "*", 0, false, false, false,
false, false); // wild character is created as a node so that a user
will be able to choose it when he needs to
    private Vector vectorIncludingWildNode; // This vector will include
all of the current nodes created by the user and additionally the wild
node which is a symbolic node representing all possible paths from node
A to node B

    private JLabel pathNameLabel, selectPathNodesLabel, pathNodesLabel,
pathParametersLabel, lineupLabel, lineupLabel2, bwUnitLabel;
    private JLabel lineupLabel3, lineupLabel4, lineupLabel5,
lineupLabel6, lineupLabel7, lineupLabel8, lineupLabel9;
    private JTextField pathNameTextField, bwValueTextField;
    private JComboBox nodesComboBox;
    private JList pathNodesList;
    private JCheckBox bwCheckBox, delayCheckBox, lossRateCheckBox,
jitterCheckBox, usedBwCheckBox;
    private JButton addNodeToPathButton, okButton, cancelButton;

    private boolean hasWhiteSpace = false;

    /**
     * Method      : CreatePath
     * Purpose     : Constructor for class CreatePath
     * Parameters  : GuiMenu gui, DirectorClass x
     */

    public CreatePath ( GuiMenu gui, DirectorClass x ) {
        super ( gui, "Create Path", true ); // Parent Frame is main GUI,
and this JDialog is modal
        setLocation (115, 115);
        this.boss = x;
        createPathGuiBuilderMethod ( );
        setResizable (false);

```

```

        setSize(560, 480);
        show();
    } // End of constructor

/**
 * Method      : createPathGuiBuilderMethod
 * Purpose     : This method builds the GUI for path creation.
 * Parameters  : None
 */

private void createPathGuiBuilderMethod ( ) {
    Container c = getContentPane ();
    c.setLayout( new FlowLayout () );

    pathNameLabel = new JLabel ("Path name          ");
    c.add(pathNameLabel);

    pathNameTextField = new JTextField (10);
    c.add(pathNameTextField);

    lineupLabel7 = new JLabel ("                ");
    c.add(lineupLabel7);

    selectPathNodesLabel = new JLabel ("Select nodes of the path ");
    c.add(selectPathNodesLabel);

    vectorIncludingWildNode = new Vector (1);
    vectorIncludingWildNode = boss.getNodeVector(); // a reference
to node vector of Director class to add a wild node
    vectorIncludingWildNode.addElement(wildNode);

    nodesComboBox = new JComboBox ( vectorIncludingWildNode );
    nodesComboBox.setMaximumRowCount(7);
    c.add ( new JScrollPane (nodesComboBox) ); // Provide a scroll
bar if there are more than 7 nodes in the combo box.
    Dimension d = new Dimension (175, 25); // dimension object
(width, height) to be used as the dimension of node combo box
    nodesComboBox.setPreferredSize(d); // set the size of the node
combo box so that it will not be too big or too small

    ButtonHandler handlerBut = new ButtonHandler ();

    addNodeToPathButton = new JButton ("Add selected node to path");
    addNodeToPathButton.addActionListener(handlerBut);
    c.add(addNodeToPathButton);

    pathNodesLabel = new JLabel ("Nodes of the path are:          ");
    c.add(pathNodesLabel);

    pathNodesList = new JList ( );
    pathNodesList.setBackground(Color.lightGray);
    c.add ( new JScrollPane (pathNodesList) );

    lineupLabel8 = new JLabel ("                ");
    c.add(lineupLabel8);

```

```

pathParametersLabel = new JLabel ("Path parameters:");
c.add(pathParametersLabel);

CheckBoxHandler handler = new CheckBoxHandler ();

lineupLabel9 = new JLabel ("");
c.add(lineupLabel9);

bwCheckBox = new JCheckBox ("BW");
c.add (bwCheckBox);
bwCheckBox.addItemListener(handler);

bwValueTextField = new JTextField (10);
bwValueTextField.setEditable(false);
c.add(bwValueTextField);

bwUnitLabel = new JLabel ("MBPS");
c.add(bwUnitLabel);

lineupLabel13 = new JLabel ("");
c.add(lineupLabel13);

delayCheckBox = new JCheckBox ("Delay ( )");
c.add (delayCheckBox);
delayCheckBox.addItemListener(handler);

lineupLabel14 = new JLabel ("");
c.add(lineupLabel14);

lossRateCheckBox = new JCheckBox ("Loss Rate ( )");
c.add (lossRateCheckBox);
lossRateCheckBox.addItemListener(handler);

jitterCheckBox = new JCheckBox ("Jitter ( )");
c.add (jitterCheckBox);
jitterCheckBox.addItemListener(handler);

usedBwCheckBox = new JCheckBox ("Used BW ( )");
c.add (usedBwCheckBox);
usedBwCheckBox.addItemListener(handler);

lineupLabel15 = new JLabel ("");
c.add(lineupLabel15);

okButton = new JButton (" OK ");
c.add(okButton);
okButton.addActionListener(handlerBut);

lineupLabel16 = new JLabel ("");
c.add(lineupLabel16);

cancelButton = new JButton ("Cancel");
c.add(cancelButton);
cancelButton.addActionListener(handlerBut);
} // End of createPathGuiBuilderMethod

```

```

/**
 * Class      : CheckBoxHandler
 * Purpose    : Inner class for combo box event handling
 */

private class CheckBoxHandler implements ItemListener {
    public void itemStateChanged( ItemEvent e ) {
        if ( e.getSource() == bwCheckBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                bwValueTextField.setEditable(true);
                networkPath.setHasBwParameter(true);
            }
            else {
                bwValueTextField.setEditable(false);
                // If the user deselects the BW parameter check box,
then the value in the bw value text field
                // is cleared.
                bwValueTextField.setText("");
                networkPath.setHasBwParameter(false);
                networkPath.setBwValue(0);
            }
        }
        if ( e.getSource() == delayCheckBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                networkPath.setHasDelayParameter(true);
            }
            else {
                networkPath.setHasDelayParameter(false);
            }
        }
        if ( e.getSource() == lossRateCheckBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                networkPath.setHasLossRateParameter(true);
            }
            else {
                networkPath.setHasLossRateParameter(false);
            }
        }
        if ( e.getSource() == jitterCheckBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                networkPath.setHasJitterParameter(true);
            }
            else {
                networkPath.setHasJitterParameter(false);
            }
        }
        if ( e.getSource() == usedBwCheckBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                networkPath.setHasUsedBwParameter(true);
            }
            else {
                networkPath.setHasUsedBwParameter(false);
            }
        }
    }
}

```

```

        } //end of method public void itemStateChanged (ItemEvent e)

    } //end of inner class CheckBoxHandler

    /**
     * Class      : ButtonHandler
     * Purpose    : Inner class for add target to policy, OK and Cancel
     * button event handling. This method checks the validity of the
     * path creation. For example, loops, empty path name, empty BW
     * value, invalid BW value are all checked. And after the checks
     * calls the addRecord method for adding the path and then destroys
     * the creation window.
     */

    private class ButtonHandler implements ActionListener {
        public void actionPerformed ( ActionEvent e) {
            if ( e.getSource() == addNodeToPathButton ) {
                // Following if statement checks if the newly selected node
                will cause a loop for the path, if it is so appropriate warning message
                will be given to the user.
                if ( ( tempNodeVectorForEachPath.contains(
nodesComboBox.getSelectedItemAt() ) ) && !( ( (Node)
nodesComboBox.getSelectedItemAt() ).getName() ).equalsIgnoreCase("*") )
                ) {
                    JOptionPane.showMessageDialog (CreatePath.this, "Please
create a valid path.", "Loop warning", JOptionPane.ERROR_MESSAGE);
                }
                else {
                    tempNodeVectorForEachPath.addElement (
nodesComboBox.getSelectedItemAt() );
                    pathNodesList.setListData (tempNodeVectorForEachPath);
                }
            }

            if ( e.getSource() == okButton ) {
                if ( (pathNameTextField.getText() ).equals("") ) {
                    JOptionPane.showMessageDialog (CreatePath.this, "Please
provide a path name.", "Path name needed", JOptionPane.ERROR_MESSAGE);
                }
                else if ( whiteSpaceControl ( pathNameTextField.getText() )
                ) {
                    JOptionPane.showMessageDialog (CreatePath.this, "White
spaces are not allowed in path name", "White space warning",
JOptionPane.ERROR_MESSAGE);
                }
                else if ( tempNodeVectorForEachPath.isEmpty() ) {
                    JOptionPane.showMessageDialog (CreatePath.this, "Please
select node(s).", "Node selection needed", JOptionPane.ERROR_MESSAGE);
                }
                // If only one node is defined as the path and that node is
                not wild card character, then give error message.
                // Because, there must be at least two nodes to create a
                path.
            }
        }
    }

```

```

        else if ( ( tempNodeVectorForEachPath.size() == 1 ) && !( (
( ( Node ) tempNodeVectorForEachPath.firstElement() ).getName()
).equalsIgnoreCase("*") ) ) {
            JOptionPane.showMessageDialog (CreatePath.this, "There
must be at least 2 nodes to create a path.", "Additional node selection
needed", JOptionPane.ERROR_MESSAGE);
        }
        else if ( ( networkPath.getHasBwParameter () ) && (
(bwValueTextField.getText() ).equals("") ) ) {
            JOptionPane.showMessageDialog (CreatePath.this, "Please
provide a BW value.", "BW value needed", JOptionPane.ERROR_MESSAGE);
        }
        else if ( ( networkPath.getHasBwParameter () ) ) {
            try {
                float temp = Float.parseFloat (
bwValueTextField.getText() );
                if ( temp >= 0 ) {
                    addRecord ();
                } // End of if
                else { // If the BW value is negative
                    JOptionPane.showMessageDialog (CreatePath.this,
"BW value must be positive.", "Invalid BW value",
JOptionPane.ERROR_MESSAGE);
                } // End of else
            } // End of try
            catch (NumberFormatException nfe) {
                JOptionPane.showMessageDialog (CreatePath.this,
"Please provide a number as a Bandwidth value.", "Invalid number",
JOptionPane.ERROR_MESSAGE);
            } // End of catch
        } // End of else if ( ( networkLink.getHasBwParameter () ) )
    )

    else {
        addRecord();
    }
} // End of if ( e.getSource() == okButton )

if ( e.getSource() == cancelButton ) {
    // remove the wild node from the vector including the wild
node since this vector
    // is a reference to node vector of Director class
    vectorIncludingWildNode.removeElement (wildNode);
    CreatePath.this.dispose();
} // End of if ( e.getSource() == cancelButton )

} // End of public void actionPerformed ( ActionEvent e)

```

```

/**
 * Method      : whiteSpaceControl

```

```

* Purpose      : This method checks the strings that the user
* creates to put them in a format that PPL compiler accepts. Path
* name should not contain white space characters. This method
* returns true if a string has white space character, false
* otherwise.
* Parameters   : String s
*/

private boolean whiteSpaceControl (String s) {
    hasWhiteSpace = false;
    char charArray [] = s.toCharArray();
    for ( int i = 0; i < charArray.length; ++i ) {
        if ( Character.isWhitespace ( charArray[i] ) ) {
            hasWhiteSpace = true;
            break;
        } // End of if
    } // End of for
    if ( hasWhiteSpace == true) {
        return true;
    }
    else {
        return false;
    }
} // End of whiteSpaceControl Method

} // End of private class ButtonHandler implements ActionListener

/**
* Method      : addRecord
* Purpose     : This method adds a path to the path vector, if that
* path is valid. Otherwise warns the policy maker about the
* redefinition problem.
* Parameters  : None
*/

private void addRecord () {
    networkPath.setName ( pathNameTextField.getText() ); // set the
name of the path
    networkPath.setPathNodesVector(tempNodeVectorForEachPath); // set
the node vector of path
    if ( networkPath.getHasBwParameter () ) {
        networkPath.setBwValue( Float.parseFloat (
bwValueTextField.getText() ) ); // set the bw value of the path if it
has one
    }
    else {
        networkPath.setBwValue (0); // set the bw value of the path to
0, if it does not have a bw value
    }

    // Newly created path is checked against the previous ones to see
if a redefinition occurs or not.

```

```

        if ( boss.pathRedefinitionCheck (networkPath) ) { // method
pathRedefinitionCheck returns true if there is no redefinition
            boss.addPath(networkPath); // add the path to path vector in
the Director class.
            vectorIncludingWildNode.removeElement (wildNode); // remove
the wild node from the vector including the wild node since this vector
is a reference to node vector of Director class and wild node is not a
node created by a user but a symbolic one.
            this.dispose(); // Releases resources allocated for a context
(current instance of CreatePath class).
        }
        else {
            JOptionPane.showMessageDialog (this, "A path with the same
name already exists. Please change the name.", "Redefinition error",
JOptionPane.ERROR_MESSAGE);
        }

    } // End of public void addRecord ()

} // End of class CreatePath

```



```

/*****
File: CreatePolicy.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/22/2001
Description: In this file, I will design the main GUI which will be
used to create a policy. There will be 5 JButtons for Policy ID, Policy
Path, Policy Target, Policy Condition, Policy Action. Policy maker will
be a JLabel since it will automatically be filled after the login
process.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreatePolicy extends JDialog {

    private DirectorClass boss;
    private GuiMenu guiMenuForPolicyCreation;
    private Policy networkPolicy = new Policy ();

    private boolean policyIDIsSet = false;
    private boolean policyPathIsSet = false;
    private boolean policyTargetIsSet = false;
    private boolean policyConditionIsSet = false;
    private boolean policyActionIsSet = false;

    private JButton policyIDButton, policyPathButton,
policyTargetButton, policyConditionButton;
    private JButton policyActionButton, okButton, cancelButton;
    private JLabel policyCreatorLabel, atLabel, lineupLabel,
lineupLabel2, lineupLabel3, lineupLabel4, infoLabel;
    private JTextArea viewJTextArea;

    /**
     * Method      : CreatePolicy
     * Purpose      : Constructor for class CreatePolicy
     * Parameters   : GuiMenu gui, DirectorClass x
     */

    public CreatePolicy ( GuiMenu gui, DirectorClass x ) {
        super ( gui, "Create Policy", true );
        setLocation (115, 115);
        this.guiMenuForPolicyCreation = gui;
        this.boss = x;
        createPolicyGuiBuilderMethod ( );
        setResizable (false);
        setSize(780, 300);
        show();

        /* The login ID of the policy maker is automatically assigned to
the policy maker ID of the policy. Therefore, policies will only be

```

able to be associated with the user currently logged on. This will prevent an ordinary user from assigning the "a network manager" identifier to a policy in hopes of guaranteeing its implementation. */

```
        networkPolicy.setPolicyMakerID( boss.getPolicyMakerID() ); // set
the policy maker of the policy.
```

```
    } // End of constructor
```

```
/**
 * Method      : seven set methods
 * Purpose     : Following seven set methods are used to set the
 * Policy ID, Policy Path, Target Traffic, Path Conditions and
 * Action Items elements of a policy.
 * Parameters  : Appropriate parameter for setting each of the
elements.
 */
```

```
public void setNetworkPolicyID ( String s ) {
    networkPolicy.setPolicyID ( s );
    policyIDIsSet = true;
}
```

```
public void setNetworkPolicyNodeVector ( Vector v ) {
    networkPolicy.setNodesInPolicyPathVector ( v );
    policyPathIsSet = true;
}
```

```
public void setNetworkPolicyLinkVector ( Vector v ) {
    networkPolicy.setLinksInPolicyPathVector ( v );
}
```

```
public void setNetworkPolicyPathVector ( Vector v ) {
    networkPolicy.setPathsInPolicyPathVector ( v );
}
```

```
public void setNetworkPolicyTarget ( Target t ) {
    networkPolicy.setPolicyTarget ( t );
    policyTargetIsSet = true;
}
```

```
public void setNetworkPolicyCondition ( Condition c ) {
    networkPolicy.setPolicyCondition ( c );
    policyConditionIsSet = true;
}
```

```
public void setNetworkPolicyAction ( Action a ) {
    networkPolicy.setPolicyAction ( a );
    policyActionIsSet = true;
}
```

```
/**
 * Method      : createPolicyGuiBuilderMethod
```

```

* Purpose      : This method builds the GUI for policy creation
* process .
* Parameters   : None
*/

private void createPolicyGuiBuilderMethod () {
    Container c = getContentPane ();
    c.setLayout( new FlowLayout () );

    infoLabel = new JLabel ("Move the mouse on any policy element to
see information about that element.      ");
    c.add(infoLabel);

    lineupLabel = new JLabel ("      ");
    c.add(lineupLabel);

    ButtonHandler handlerBut = new ButtonHandler ();

    MouseHandler handlerMouse = new MouseHandler ();

    policyIDButton = new JButton ("Policy ID");
    policyIDButton.addActionListener(handlerBut);
    policyIDButton.addMouseListener(handlerMouse);
    policyIDButton.setBackground(Color.pink);
    policyIDButton.setBorderPainted(true);
    c.add ( policyIDButton );

    policyCreatorLabel = new JLabel ( boss.getPolicyMakerID() );
    policyCreatorLabel.addMouseListener(handlerMouse);
    c.add ( policyCreatorLabel );

    atLabel = new JLabel ( " @ " );
    c.add(atLabel);

    policyPathButton = new JButton ("{Policy Path}");
    policyPathButton.addActionListener(handlerBut);
    policyPathButton.addMouseListener(handlerMouse);
    policyPathButton.setBackground(Color.cyan);
    policyPathButton.setBorderPainted(true);
    c.add ( policyPathButton );

    policyTargetButton = new JButton ("{Policy Target}");
    policyTargetButton.addActionListener(handlerBut);
    policyTargetButton.addMouseListener(handlerMouse);
    policyTargetButton.setBackground(Color.green);
    policyTargetButton.setBorderPainted(true);
    c.add ( policyTargetButton );

    policyConditionButton = new JButton ("{Policy Condition}");
    policyConditionButton.addActionListener(handlerBut);
    policyConditionButton.addMouseListener(handlerMouse);
    policyConditionButton.setBackground(Color.red);
    policyConditionButton.setBorderPainted(true);
    c.add ( policyConditionButton );

    policyActionButton = new JButton ("{Policy Action}");

```

```

policyActionButton.addActionListener(handlerBut);
policyActionButton.addMouseListener(handlerMouse);
policyActionButton.setBackground(Color.orange);
policyActionButton.setBorderPainted(true);
c.add ( policyActionButton );

lineupLabel2 = new JLabel ( "                                ");
c.add(lineupLabel2);

viewJTextArea = new JTextArea (5, 62);
viewJTextArea.setBackground(Color.lightGray);
viewJTextArea.setEditable(false);
viewJTextArea.setLineWrap(true);
c.add ( new JScrollPane (viewJTextArea) );

lineupLabel3 = new JLabel ( "                                ");
c.add(lineupLabel3);

okButton = new JButton ( "   OK   ");
okButton.addActionListener(handlerBut);
c.add ( okButton );

lineupLabel4 = new JLabel ( "                                ");
c.add(lineupLabel4);

cancelButton = new JButton ("Cancel");
cancelButton.addActionListener(handlerBut);
c.add ( cancelButton );

} // End of method createPolicyGuiBuilderMethod

/**
 * Class           : MouseHandler
 * Purpose         : When the mouse is moved on a policy element,
 * general information about that element will be displayed to the
 * user if that element has not been defined yet. If an element has
 * already been defined, then information about that element will be
 * displayed to the user by means of this class.
 */
private class MouseHandler extends MouseAdapter {

    public void mouseEntered ( MouseEvent e ) {
        if (e.getSource() == policyIDButton ) {
            if ( policyIDIsSet ) {
                viewJTextArea.setText ( "Policy ID is:  " +
networkPolicy.getPolicyID() );
            }
            else {
                viewJTextArea.setText ( "Policy ID is a unique policy
identification token. It is not defined for this policy yet." );
            }
        } // End of if (e.getSource() == policyIDButton )

        if ( e.getSource() == policyCreatorLabel ) {

```

```

        viewJTextArea.setText ( "This is login name of policy
creator. It is automatically assigned after login process." );
    }

    if ( e.getSource() == policyPathButton ) {
        if ( policyPathIsSet ) {
            pathDisplayMethod ();
        }
        else {
            viewJTextArea.setText ( "Policy path is network paths
that this policy affects. It is not defined for this policy yet." );
        }
    }

    if ( e.getSource() == policyTargetButton ) {
        if ( policyTargetIsSet ) {
            targetDisplayMethod ();
        }
        else {
            viewJTextArea.setText ( "Policy target is user packets
that this policy affects. Items are OR'ed. It is not defined for this
policy yet." );
        }
    }

    if ( e.getSource() == policyConditionButton ) {
        if ( policyConditionIsSet ) {
            conditionDisplayMethod ();
        }
        else {
            viewJTextArea.setText ( "Policy condition is any
conditions associated with policy path. Items are AND'ed. It is not
defined for this policy yet." );
        }
    }

    if ( e.getSource() == policyActionButton ) {
        if ( policyActionIsSet ) {
            actionDisplayMethod ();
        }
        else {
            viewJTextArea.setText ( "Policy action is used for
explicit accept, deny and for setting parameters. It is not defined for
this policy yet." );
        }
    }

} // End of method mouseEntered

```

```

public void mouseExited ( MouseEvent e ) {

```

```

        viewJTextArea.setText("");
    } // End of method mouseExited

    private void pathDisplayMethod () {
        viewJTextArea.append ( "Policy path is: " + "{ " );
        Vector nodeLinkPathNames = new Vector (1);

        Enumeration enumForPolicyNodeVector = (
networkPolicy.getNodesInPolicyPathVector() ).elements();
        while ( enumForPolicyNodeVector.hasMoreElements() ) {
            Node tempNode = new Node ();
            tempNode = ( Node ) enumForPolicyNodeVector.nextElement ();
            nodeLinkPathNames.addElement( tempNode.toString() );
        }

        Enumeration enumForPolicyLinkVector = (
networkPolicy.getLinksInPolicyPathVector() ).elements();
        while ( enumForPolicyLinkVector.hasMoreElements() ) {
            Link tempLink = new Link ();
            tempLink = ( Link ) enumForPolicyLinkVector.nextElement ();
            nodeLinkPathNames.addElement( tempLink.toString() );
        }

        Enumeration enumForPolicyPathVector = (
networkPolicy.getPathsInPolicyPathVector() ).elements();
        while ( enumForPolicyPathVector.hasMoreElements() ) {
            Path tempPath = new Path ();
            tempPath = ( Path ) enumForPolicyPathVector.nextElement ();
            nodeLinkPathNames.addElement( tempPath.toString() );
        }

        Enumeration enumForNodeLinkPathNamesVector =
nodeLinkPathNames.elements();
        for ( int i = 0;
enumForNodeLinkPathNamesVector.hasMoreElements (); i++) {
            String tempString;
            tempString = ( String )
enumForNodeLinkPathNamesVector.nextElement ();
            if ( i >= 1 ) {
                viewJTextArea.append ( ", " );
            }

            viewJTextArea.append ( tempString );
        } // End of for
        viewJTextArea.append ( " }" );
    } // End of method pathDisplayMethod

    private void targetDisplayMethod () {

```

```

        viewJTextArea.append ( "Policy target is: " + "{ " );
        // If this policy has a wild card character in its target
        element.
        if ( ( networkPolicy.getPolicyTarget()
        ).getTargetAllProperty() ) {
            viewJTextArea.append ( "*" );
        } // End of if
        else {
            Enumeration enumForTargetVector = ( (
            networkPolicy.getPolicyTarget() ).getPolicyTargets() ).elements();
            for ( int i = 0; enumForTargetVector.hasMoreElements () ;
            i++) {
                String tempString;
                tempString = ( ( OneTarget )
            enumForTargetVector.nextElement () ).outputMethod();
                if ( i >=1 ) {
                    viewJTextArea.append ( " || " );
                } // End of if

                viewJTextArea.append ( tempString );

            } // End of for

            viewJTextArea.append ( " }" );

        } // End of else

    } // End of method targetDisplayMethod

    private void conditionDisplayMethod () {
        Vector conditionElementVector = new Vector (1); // Condition
        elements will be placed in it as Strings
        viewJTextArea.append ( "Policy condition is: " + "{ " );
        // If this policy has a wild card character in its condition
        element.
        if ( ( networkPolicy.getPolicyCondition()
        ).getNoConditionProperty() ) {
            viewJTextArea.append ( "*" );
        } // End of if

        else{
            Enumeration enumForPriorityConditionVector = ( (
            networkPolicy.getPolicyCondition() ).getPriorityConditionVector()
            ).elements();
            while ( enumForPriorityConditionVector.hasMoreElements() )
            {
                OnePriority tempOnePriority = new OnePriority ();
                tempOnePriority = ( OnePriority )
            enumForPriorityConditionVector.nextElement ();
                conditionElementVector.addElement(
            tempOnePriority.toString() );
            }
        }
    }

```

```

        Enumeration enumForHopCountConditionVector = ( (
networkPolicy.getPolicyCondition() ).getHopCountConditionVector()
).elements();
        while ( enumForHopCountConditionVector.hasMoreElements() )
        {
            OneHopCount tempOneHopCount = new OneHopCount ();
            tempOneHopCount = ( OneHopCount )
enumForHopCountConditionVector.nextElement ();
            conditionElementVector.addElement(
tempOneHopCount.toString() );
        }

        Enumeration enumForTimeConditionVector = ( (
networkPolicy.getPolicyCondition() ).getTimeConditionVector()
).elements();
        while ( enumForTimeConditionVector.hasMoreElements() ) {
            OneTime tempOneTime = new OneTime ();
            tempOneTime = ( OneTime )
enumForTimeConditionVector.nextElement ();
            conditionElementVector.addElement(
tempOneTime.toString() );
        }

        Enumeration enumForSrcIPAddressConditionVector = ( (
networkPolicy.getPolicyCondition() ).getSrcIPAddressConditionVector()
).elements();
        while (
enumForSrcIPAddressConditionVector.hasMoreElements() ) {
            OneSrcIPAddress tempOneSrcIPAddress = new
OneSrcIPAddress ();
            tempOneSrcIPAddress = ( OneSrcIPAddress )
enumForSrcIPAddressConditionVector.nextElement ();
            conditionElementVector.addElement(
tempOneSrcIPAddress.toString() );
        }

        Enumeration enumForBWConditionVector = ( (
networkPolicy.getPolicyCondition() ).getBWConditionVector()
).elements();

        while ( enumForBWConditionVector.hasMoreElements() ) {
            OneBW tempOneBW = new OneBW ();
            tempOneBW = ( OneBW )
enumForBWConditionVector.nextElement ();
            conditionElementVector.addElement( tempOneBW.toString()
);
        }

        Enumeration enumForUserIDConditionVector = ( (
networkPolicy.getPolicyCondition() ).getUserIDConditionVector()
).elements();

        while ( enumForUserIDConditionVector.hasMoreElements() ) {
            OneUserID tempOneUserID = new OneUserID ();
            tempOneUserID = ( OneUserID )
enumForUserIDConditionVector.nextElement ();

```



```

        String temp = tempOneUserID.toString();
        conditionElementVector.addElement(
tempOneUserID.toString() );
    }

    Enumeration enumForTypeConditionVector = ( (
networkPolicy.getPolicyCondition() ).getTypeConditionVector()
).elements();
    while ( enumForTypeConditionVector.hasMoreElements() ) {
        OneType tempOneType = new OneType ();
        tempOneType = ( OneType )
enumForTypeConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneType.toString() );
    }

    Enumeration enumForParameterConditionVector = ( (
networkPolicy.getPolicyCondition() ).getParameterConditionVector()
).elements();
    while ( enumForParameterConditionVector.hasMoreElements() )
    {
        OneParameter tempOneParameter = new OneParameter ();
        tempOneParameter = ( OneParameter )
enumForParameterConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneParameter.toString() );
    }

    /* All 8 types of conditions are placed in the
conditionElementVector and they are displayed in the JTextArea by the
following Enumeration */
    Enumeration enumForConditionElementVector =
conditionElementVector.elements();
    for ( int i = 0;
enumForConditionElementVector.hasMoreElements (); i++) {
        String tempString;
        tempString = ( String )
enumForConditionElementVector.nextElement ();
        if ( i >= 1 ) {
            viewJTextArea.append ( " && " );
        }
        viewJTextArea.append ( tempString );
    } // End of for

    viewJTextArea.append ( " }" );

} // End of else (condition part does not have a wild card
character)

} // End of method conditionDisplayMethod

```

```

private void actionDisplayMethod () {
    viewJTextArea.append ( "Policy action is:  " + "{ " );

    if ( ( networkPolicy.getPolicyAction() ).getHasDenyAction() )
{
    viewJTextArea.append("deny "; " + "\n");
    }
    else {
        Vector actionElementVector = new Vector (1);
        if ( ( networkPolicy.getPolicyAction()
).getHasPermitAction() ) {
            actionElementVector.addElement("permit");
        }
        if ( ( networkPolicy.getPolicyAction() ).getPriorityValue()
!= 0 ) {
            actionElementVector.addElement("priority" + " := " + (
networkPolicy.getPolicyAction() ).getPriorityValue() );
        }
        if ( ( networkPolicy.getPolicyAction() ).getHopCountValue()
!= 0 ) {
            actionElementVector.addElement("hopCount" + " := " + (
networkPolicy.getPolicyAction() ).getHopCountValue() );
        }
        if ( ( networkPolicy.getPolicyAction()
).getAllocatedBWValue() != 0 ) {
            actionElementVector.addElement("allocated_bw" + " := " +
( networkPolicy.getPolicyAction() ).getAllocatedBWValue() + " MBPS");
        }
        if ( ( networkPolicy.getPolicyAction()
).getMaxLossRateValue() != 0 ) {
            actionElementVector.addElement("maxLossRate" + " := " +
( networkPolicy.getPolicyAction() ).getMaxLossRateValue() + " %");
        }
        if ( ( networkPolicy.getPolicyAction()
).getDelayBoundValue() != 0 ) {
            actionElementVector.addElement("delayBound" + " := " + (
networkPolicy.getPolicyAction() ).getDelayBoundValue() + " msec");
        }
        if ( ( networkPolicy.getPolicyAction()
).getSecurityLevelValue() != 0 ) {
            actionElementVector.addElement("securityLevel" + " := "
+ ( networkPolicy.getPolicyAction() ).getSecurityLevelValue() );
        }
    }

    Enumeration enumForActionElementVector =
actionElementVector.elements();
    for ( int i = 0; enumForActionElementVector.hasMoreElements
()); i++) {
        String tempString;
        tempString = ( String )
enumForActionElementVector.nextElement ();
        if ( i >= 1 ) {
            viewJTextArea.append ( ", " );
        }
        viewJTextArea.append ( tempString );
    }
}

```

```

        } // End of for
        viewJTextArea.append ( " };" );
    } // End of else

    } // End of method actionPerformed

} // End of class MouseHandler

/**
 * Class      : ButtonHandler
 * Purpose    : This class creates different windows for the
 * creation of each of the element of a policy when the button, on
 * which the name of the element is written, is clicked. Policy ID,
 * Policy Path, Target Traffic, Path Conditions and Action Items are
 * 5 elements of a policy in PPL.
 */
private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e ) {
        if ( e.getSource() == policyIDButton ) {
            CreatePolicyID createPolicyIDObject = new CreatePolicyID (
boss, CreatePolicy.this);
        } // End of if ( e.getSource() == policyIDButton )
        if ( e.getSource() == policyPathButton ) {
            CreatePolicyFirst createPolicyFirstObject = new
CreatePolicyFirst ( boss, CreatePolicy.this);
        }
        if ( e.getSource() == policyTargetButton ) {
            CreatePolicySecond createPolicySecondObject = new
CreatePolicySecond ( boss, CreatePolicy.this);
        }
        if ( e.getSource() == policyConditionButton ) {
            CreatePolicyThird createPolicyThirdObject = new
CreatePolicyThird ( guiMenuForPolicyCreation, boss, CreatePolicy.this);
        }
        if ( e.getSource() == policyActionButton ) {
            CreatePolicyFourth createPolicyFourthObject = new
CreatePolicyFourth ( boss, CreatePolicy.this);
        }
        if ( e.getSource() == okButton ) {
            if ( ! policyIDIsSet ) {
                JOptionPane.showMessageDialog (CreatePolicy.this,
"Policy ID was not defined", "Policy ID definition needed ",
JOptionPane.ERROR_MESSAGE);
            }
            else if ( ! policyPathIsSet ) {
                JOptionPane.showMessageDialog (CreatePolicy.this,
"Policy path was not defined", "Policy path definition needed ",
JOptionPane.ERROR_MESSAGE);
            }
            else if ( ! policyTargetIsSet ) {
                JOptionPane.showMessageDialog (CreatePolicy.this,
"Policy target was not defined", "Policy target definition needed ",
JOptionPane.ERROR_MESSAGE);
            }
            else if ( ! policyConditionIsSet ) {

```

```

        JOptionPane.showMessageDialog (CreatePolicy.this,
"Policy condition was not defined", "Policy condition definition needed
", JOptionPane.ERROR_MESSAGE);
    }
    else if ( ! policyActionIsSet ) {
        JOptionPane.showMessageDialog (CreatePolicy.this,
"Policy action was not defined", "Policy action definition needed ",
JOptionPane.ERROR_MESSAGE);
    }
    else {
        boss.addPolicy (networkPolicy);
        CreatePolicy.this.dispose();
    }
} // End of if ( e.getSource() == okButton )
if ( e.getSource() == cancelButton ) {
    CreatePolicy.this.dispose();
}

} // End of method actionPerformed

} // End of class ButtonHandler

} // End of class CreatePolicy

```

```

/*****
File: CreatePolicyFirst.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/03/2001
Description: By this file, the policy maker will create the path that
the policy will affect.
*****/

```

```

import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

```

```

public class CreatePolicyFirst extends JDialog {
    private DirectorClass boss;

    private CreatePolicy createPolicyForPath;

    private ModifyPolicySecond modifyPolicySecondForPath;

    private boolean createPolicyIsCalled = false;

    private Vector tempNodeVectorForEachPolicyPath = new Vector (1); //
    This vector will be used to set the nodes in policy path vector of the
    policy class to make each policy know the nodes that it contains in the
    policy path part of itself (policy)
    private Vector tempLinkVectorForEachPolicyPath = new Vector (1); //
    This vector will be used to set the links in policy path vector of the
    policy class to make each policy know the links that it contains in the
    policy path part of itself (policy)
    private Vector tempPathVectorForEachPolicyPath = new Vector (1); //
    This vector will be used to set the paths in policy path vector of the
    policy class to make each policy know the paths that it contains in the
    policy path part of itself (policy)

    private Vector displayNodesLinksPathsVector = new Vector (1); //
    This vector will collect the nodes, links and paths in a policy path as
    objects and will be used to display policy path in the policy path
    JList for the user to be able to see the policy path he is creating for
    the time being.

    private JLabel selectNodesLabel, selectLinksLabel, selectPathsLabel;
    private JLabel policyPathLabel, lineupLabel, lineupLabel2,
lineupLabel3, lineupLabel4;
    private JComboBox nodesComboBox, linksComboBox, pathsComboBox;
    private JList policyPathList;
    private JButton addNodeButton, addLinkButton, addPathButton,
okButton, cancelButton;

```

```

/**
 * Method      : CreatePolicyFirst

```

```

* Purpose      : Constructor for policy path creation
* Parameters   : DirectorClass x, CreatePolicy c
*/

public CreatePolicyFirst ( DirectorClass x, CreatePolicy c ) {
    super ( c, "Policy Path Creation", true ); // Parent Frame is
CreatePolicy window (c), and this JDialog is modal
    setLocation (115, 115);
    this.boss = x;
    this.createPolicyForPath = c;
    createPolicyIsCalled = true;
    createPolicyFirstGuiBuilderMethod ( );
    setResizable (false);
    setSize(560, 375);
    show();
} // End of constructor for policy path creation

/**
* Method       : CreatePolicyFirst
* Purpose      : Constructor for policy path modification
* Parameters   : DirectorClass x, ModifyPolicySecond c
*/

public CreatePolicyFirst ( DirectorClass x, ModifyPolicySecond c ) {
    super ( c, "Policy Path Modification", true );
    setLocation (115, 115);
    this.boss = x;
    this.modifyPolicySecondForPath = c;
    createPolicyFirstGuiBuilderMethod ( );
    setResizable (false);
    setSize(560, 450);
    show();
} // End of constructor for policy path modification

/**
* Method       : createPolicyFirstGuiBuilderMethod
* Purpose      : This method builds the GUI for the creation of the
* first screen of policy creation process.
* Parameters   : None
*/

private void createPolicyFirstGuiBuilderMethod () {
    Container c = getContentPane ( );
    c.setLayout( new FlowLayout ( ) );

    selectNodesLabel = new JLabel ("Select nodes in policy path  ");
    c.add(selectNodesLabel);

    nodesComboBox = new JComboBox ( boss.getNodeVector() );
    nodesComboBox.setMaximumRowCount(7);
    c.add ( new JScrollPane (nodesComboBox) ); // Provide a scroll
bar if there are more than 7 nodes in the combo box.

```

```

        Dimension d = new Dimension (175, 25); // dimension object
        (width, height) to be used as the dimension of node combo box
        nodesComboBox.setPreferredSize(d); // set the size of the node
        combo box so that it will not be too big or too small

        ButtonHandler handlerBut = new ButtonHandler ();

        addNodeButton = new JButton ("Add selected node");
        addNodeButton.addActionListener(handlerBut);
        c.add(addNodeButton);
        if ( ( boss.getNodeVector() ).isEmpty() ) {
            addNodeButton.setEnabled(false); // If the user did not create
            a node yet, then he would not be able to add that non-existent node to
            policy path.
            addNodeButton.setToolTipText("No node is created");
        }

        selectLinksLabel = new JLabel ("Select links in policy path  ");
        c.add(selectLinksLabel);

        linksComboBox = new JComboBox ( boss.getLinkVector() );
        linksComboBox.setMaximumRowCount(7);
        c.add ( new JScrollPane (linksComboBox) ); // Provide a scroll
        bar if there are more than 7 links in the combo box.
        linksComboBox.setPreferredSize(d); // set the size of the link
        combo box so that it will not be too big or too small

        addLinkButton = new JButton ("Add selected link  ");
        addLinkButton.addActionListener(handlerBut);
        c.add(addLinkButton);
        if ( ( boss.getLinkVector() ).isEmpty() ) {
            addLinkButton.setEnabled(false); // If the user did not create
            a link yet, then he would not be able to add that non-existent link to
            policy path.
            addLinkButton.setToolTipText("No link is created");
        }

        selectPathsLabel = new JLabel ("Select paths in policy path  ");
        c.add(selectPathsLabel);

        pathsComboBox = new JComboBox ( boss.getPathVector() );
        pathsComboBox.setMaximumRowCount(7);
        c.add ( new JScrollPane (pathsComboBox) ); // Provide a scroll
        bar if there are more than 7 paths in the combo box.
        pathsComboBox.setPreferredSize(d); // set the size of the path
        combo box so that it will not be too big or too small

        addPathButton = new JButton ("Add selected path");
        addPathButton.addActionListener(handlerBut);
        c.add(addPathButton);
        if ( ( boss.getPathVector() ).isEmpty() ) {
            addPathButton.setEnabled(false); // If the user did not create
            a path yet, then he would not be able to add that non-existent path to
            policy path.

```

```

        addPathButton.setToolTipText("No path is created");
    }

    lineupLabel = new JLabel ("");
    c.add(lineupLabel);

    policyPathLabel = new JLabel ("Policy path is:");
    c.add(policyPathLabel);

    policyPathList = new JList ();
    policyPathList.setBackground(Color.lightGray);
    c.add ( new JScrollPane (policyPathList) );

    lineupLabel2 = new JLabel ("");
    c.add(lineupLabel2);

    lineupLabel3 = new JLabel ("");
    c.add(lineupLabel3);

    okButton = new JButton (" OK ");
    c.add(okButton);
    okButton.addActionListener(handlerBut);
    if ( ( ( boss.getNodeVector() ).isEmpty() ) && ( (
boss.getLinkVector() ).isEmpty() ) && ( ( boss.getPathVector()
).isEmpty() ) ) ) {
        okButton.setEnabled(false);
        okButton.setToolTipText("No node, link or path is created");
    }

    lineupLabel4 = new JLabel ("");
    c.add(lineupLabel4);

    cancelButton = new JButton ("Cancel");
    c.add(cancelButton);
    cancelButton.addActionListener(handlerBut);
} // End of createPolicyFirstGuiBuilderMethod

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for add selected node, add selected
 * link, add selected path, next and Cancel button event handling.
 * This method checks the validity of the first part of policy
 * creation (policy ID, user ID, policy path. For example, loops,
 * empty policy ID are checked. And after the checks the first 3
 * parts of policy is created.
 */

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == addNodeButton ) {
            // Following if statement checks if the newly selected node
will cause a loop, if it is so appropriate warning message will be
given to the user.
            if ( ( tempNodeVectorForEachPolicyPath.contains(
nodesComboBox.getSelectedItemAt() ) ) ) ) {

```



```

        JOptionPane.showMessageDialog (CreatePolicyFirst.this,
        "Please create a valid policy path.", "Loop warning",
        JOptionPane.ERROR_MESSAGE);
    }
    else {
        tempNodeVectorForEachPolicyPath.addElement(
        nodesComboBox.getSelectedItem() );
        displayNodesLinksPathsVector.addElement(
        nodesComboBox.getSelectedItem() ); // This vector will be used to
        display nodes, links and paths of a policy path in a JList.
        policyPathList.setListData
        (displayNodesLinksPathsVector); // display selected network elements
        (nodes, links and paths) in policy path
    }
}

    if ( e.getSource() == addLinkButton ) {
        // Following if statement checks if the newly selected link
        will cause a loop, if it is so appropriate warning message will be
        given to the user.
        if ( ( tempLinkVectorForEachPolicyPath.contains(
        linksComboBox.getSelectedItem() ) ) ) {
            JOptionPane.showMessageDialog (CreatePolicyFirst.this,
            "Please create a valid policy path.", "Loop warning",
            JOptionPane.ERROR_MESSAGE);
        }
        else {
            tempLinkVectorForEachPolicyPath.addElement(
            linksComboBox.getSelectedItem() );
            displayNodesLinksPathsVector.addElement(
            linksComboBox.getSelectedItem() ); // This vector will be used to
            display nodes, links and paths of a policy path in a JList.
            policyPathList.setListData
            (displayNodesLinksPathsVector); // display selected network elements
            (nodes, links and paths) in policy path
        }
    }

    if ( e.getSource() == addPathButton ) {
        // Following if statement checks if the newly selected path
        will cause a loop, if it is so appropriate warning message will be
        given to the user.
        if ( ( tempPathVectorForEachPolicyPath.contains(
        pathsComboBox.getSelectedItem() ) ) ) {
            JOptionPane.showMessageDialog (CreatePolicyFirst.this,
            "Please create a valid policy path.", "Loop warning",
            JOptionPane.ERROR_MESSAGE);
        }
        else {
            tempPathVectorForEachPolicyPath.addElement(
            pathsComboBox.getSelectedItem() );
            displayNodesLinksPathsVector.addElement(
            pathsComboBox.getSelectedItem() ); // This vector will be used to
            display nodes, links and paths of a policy path in a JList.

```

```

        policyPathList.setListData
(displayNodesLinksPathsVector); // display selected network elements
(nodes, links and paths) in policy path
    }
}

    if ( e.getSource() == okButton ) {
        if ( tempNodeVectorForEachPolicyPath.isEmpty() &&
tempLinkVectorForEachPolicyPath.isEmpty() &&
tempPathVectorForEachPolicyPath.isEmpty() ) {
            JOptionPane.showMessageDialog (CreatePolicyFirst.this,
"Please create a policy path.", "Policy path needed",
JOptionPane.ERROR_MESSAGE);
        }
        else {
            if ( createPolicyIsCalled ) {
                createPolicyForPath.setNetworkPolicyNodeVector
(tempNodeVectorForEachPolicyPath); // set the node vector of a policy
path in policy creation process
                createPolicyForPath.setNetworkPolicyLinkVector
(tempLinkVectorForEachPolicyPath); // set the link vector of a policy
path in policy creation process
                createPolicyForPath.setNetworkPolicyPathVector
(tempPathVectorForEachPolicyPath); // set the path vector of a policy
path in policy creation process
            }
            else {
                modifyPolicySecondForPath.setNetworkPolicyNodeVector
(tempNodeVectorForEachPolicyPath); // set the node vector of a policy
path in policy modification process
                modifyPolicySecondForPath.setNetworkPolicyLinkVector
(tempLinkVectorForEachPolicyPath); // set the link vector of a policy
path in policy modification process
                modifyPolicySecondForPath.setNetworkPolicyPathVector
(tempPathVectorForEachPolicyPath); // set the path vector of a policy
path in policy modification process
            }
            CreatePolicyFirst.this.dispose();
        }
    } // End of if ( e.getSource() == okButton )

    if ( e.getSource() == cancelButton ) {
        CreatePolicyFirst.this.dispose();
    } // End of if ( e.getSource() == cancelButton )

} // End of public void actionPerformed ( ActionEvent e)

} // End of private class ButtonHandler implements ActionListener

} // End of class CreatePolicyFirst

```

```

/*****
File: CreatePolicyFourth.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/18/2001
Description: In this file, I will design the fourth part of the GUI
which will be used to create a policy. The user will create the action
part of a policy by using this window.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreatePolicyFourth extends JDialog {

    private DirectorClass bossForAction;

    private CreatePolicy createPolicyForAction;

    private ModifyPolicySecond modifyPolicySecondForAction;

    private boolean createPolicyIsCalled = false;

    private Action policyAction = new Action ();

    private boolean inputError = false;

    private JCheckBox denyCheckBox, permitCheckBox, priorityCheckBox,
hopCountCheckBox, allocatedBWCheckBox;
    private JCheckBox maxLossRateCheckBox, delayBoundCheckBox,
securityLevelCheckBox;
    private JTextField priorityValueTextField, hopCountValueTextField,
allocatedBWValueTextField;
    private JTextField maxLossRateValueTextField,
delayBoundValueTextField, securityLevelValueTextField;
    private JButton okButton, cancelButton;
    private JLabel lineupLabel, lineupLabel2, lineupLabel3,
lineupLabel4, lineupLabel5, lineupLabel6, lineupLabel7, lineupLabel8;
    private JLabel mbpsLabel, percentLabel, msecLabel, equalLabel;

    /**
     * Method      : CreatePolicyFourth
     * Purpose     : Constructor for policy action creation
     * Parameters  : DirectorClass d, CreatePolicy c
     */

    public CreatePolicyFourth ( DirectorClass d, CreatePolicy c ) {
        super ( c, "Policy Action Creation", true );
        setLocation (115, 115);
        this.bossForAction = d;
        this.createPolicyForAction = c;
        createPolicyIsCalled = true;
        createPolicyFourthGuiBuilderMethod ();
        setResizable (false);
    }

```

```

        setSize(455, 370);
        show();
    } // End of constructor for policy action creation

    /**
     * Method      : CreatePolicyFourth
     * Purpose     : Constructor for policy action modification
     * Parameters  : DirectorClass d, ModifyPolicySecond c
     */

    public CreatePolicyFourth ( DirectorClass d, ModifyPolicySecond c )
    {
        super ( c, "Policy Action Modification", true );
        setLocation (115, 115);
        this.bossForAction = d;
        this.modifyPolicySecondForAction = c;
        createPolicyFourthGuiBuilderMethod ();
        setResizable (false);
        setSize(455, 370);
        show();
    } // End of constructor for policy action modification

    /**
     * Method      : createPolicyFourthGuiBuilderMethod
     * Purpose     : This method builds the GUI for node creation.
     * Parameters  : None
     */

    private void createPolicyFourthGuiBuilderMethod ( ) {
        Container c = getContentPane ();
        c.setLayout( new FlowLayout () );

        CheckBoxHandler handler = new CheckBoxHandler ();

        denyCheckBox = new JCheckBox ("Deny          ");
        c.add (denyCheckBox);
        denyCheckBox.addItemListener(handler);

        permitCheckBox = new JCheckBox ("Permit          ");
        c.add (permitCheckBox);
        permitCheckBox.addItemListener(handler);

        priorityCheckBox = new JCheckBox ("Priority      =");
        c.add (priorityCheckBox);
        priorityCheckBox.addItemListener(handler);

        priorityValueTextField = new JTextField (10);
        priorityValueTextField.setEditable(false);
        c.add(priorityValueTextField);

        lineupLabel = new JLabel ("          ");
        c.add(lineupLabel);

        hopCountCheckBox = new JCheckBox ("Hop count    =");

```

```

c.add (hopCountCheckBox);
hopCountCheckBox.addItemListener(handler);

hopCountValueTextField = new JTextField (10);
hopCountValueTextField.setEditable(false);
c.add(hopCountValueTextField);

lineupLabel2= new JLabel ("");
c.add(lineupLabel2);

allocatedBWCheckBox = new JCheckBox ("Allocated bw =");
c.add (allocatedBWCheckBox);
allocatedBWCheckBox.addItemListener(handler);

allocatedBWValueTextField = new JTextField (10);
allocatedBWValueTextField.setEditable(false);
c.add(allocatedBWValueTextField);

mbpsLabel = new JLabel ("MBPS");
c.add(mbpsLabel);

lineupLabel3= new JLabel ("");
c.add(lineupLabel3);

maxLossRateCheckBox = new JCheckBox ("Maximum Loss Rate =");
c.add (maxLossRateCheckBox);
maxLossRateCheckBox.addItemListener(handler);

maxLossRateValueTextField = new JTextField (10);
maxLossRateValueTextField.setEditable(false);
c.add(maxLossRateValueTextField);

percentLabel = new JLabel ("%");
c.add(percentLabel);

lineupLabel4= new JLabel ("");
c.add(lineupLabel4);

delayBoundCheckBox = new JCheckBox ("Delay bound =");
c.add (delayBoundCheckBox);
delayBoundCheckBox.addItemListener(handler);

delayBoundValueTextField = new JTextField (10);
delayBoundValueTextField.setEditable(false);
c.add(delayBoundValueTextField);

msecLabel = new JLabel ("msec");
c.add(msecLabel);

lineupLabel5= new JLabel ("");
c.add(lineupLabel5);

securityLevelCheckBox = new JCheckBox ("Security level =");
c.add (securityLevelCheckBox);
securityLevelCheckBox.addItemListener(handler);

```

```

securityLevelValueTextField = new JTextField (10);
securityLevelValueTextField.setEditable(false);
c.add(securityLevelValueTextField);

lineupLabel6= new JLabel ("      ");
c.add(lineupLabel6);

lineupLabel7= new JLabel ("                                ");
c.add(lineupLabel7);

ButtonHandler handlerBut = new ButtonHandler ();

okButton = new JButton ("    OK    ");
c.add(okButton);
okButton.addActionListener(handlerBut);

lineupLabel8 = new JLabel ("                                ");
c.add(lineupLabel8);

cancelButton = new JButton ("Cancel");
c.add(cancelButton);
cancelButton.addActionListener(handlerBut);

} // End of createPolicyFourthGuiBuilderMethod

/**
 * Class      : CheckBoxHandler
 * Purpose    : Inner class for check box event handling
 */

private class CheckBoxHandler implements ItemListener {

    public void itemStateChanged( ItemEvent e ) {

        if ( e.getSource() == denyCheckBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                policyAction.setHasDenyAction(true);
                resetCheckBoxSelections (); // If the user decides to
select deny after he creates other actions, then these actions are
cleared since deny cannot be selected with any other other action.
                setCheckBoxesMethod (false); // If the user selects deny
action, then all other actions are disabled since deny cannot be
selected with any other other action.
            } // End of if
            else {
                policyAction.setHasDenyAction(false);
                setCheckBoxesMethod (true); // If the user deselects
deny action, then he can select other actions again.
            } // End of else
        } // End of if ( e.getSource() == denyCheckBox )

        if ( e.getSource() == permitCheckBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {

```

```

        policyAction.setHasPermitAction(true);
    } // End of if
    else {
        policyAction.setHasPermitAction(false);
    } // End of else
} // End of if ( e.getSource() == permitCheckBox )

if ( e.getSource() == priorityCheckBox ) {
    if ( e.getStateChange() == ItemEvent.SELECTED ) {
        priorityValueTextField.setEditable(true);
    } // End of if
    else {
        priorityValueTextField.setText("");
        policyAction.setPriorityValue(0);
        priorityValueTextField.setEditable(false);
    } // End of else
} // End of if ( e.getSource() == priorityCheckBox )

if ( e.getSource() == hopCountCheckBox ) {
    if ( e.getStateChange() == ItemEvent.SELECTED ) {
        hopCountValueTextField.setEditable(true);
    } // End of if
    else {
        hopCountValueTextField.setText("");
        policyAction.setHopCountValue(0);
        hopCountValueTextField.setEditable(false);
    } // End of else
} // End of if ( e.getSource() == hopCountCheckBox )

if ( e.getSource() == allocatedBWCheckBox ) {
    if ( e.getStateChange() == ItemEvent.SELECTED ) {
        allocatedBWValueTextField.setEditable(true);
    } // End of if
    else {
        allocatedBWValueTextField.setText("");
        policyAction.setAllocatedBWValue(0);
        allocatedBWValueTextField.setEditable(false);
    } // End of else
} // End of if ( e.getSource() == allocatedBWCheckBox )

if ( e.getSource() == maxLossRateCheckBox ) {
    if ( e.getStateChange() == ItemEvent.SELECTED ) {
        maxLossRateValueTextField.setEditable(true);
    } // End of if
    else {
        maxLossRateValueTextField.setText("");
        policyAction.setMaxLossRateValue(0);
        maxLossRateValueTextField.setEditable(false);
    } // End of else
} // End of if ( e.getSource() == maxLossRateCheckBox )

```

```

        if ( e.getSource() == delayBoundCheckBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                delayBoundValueTextField.setEditable(true);
            } // End of if
            else {
                delayBoundValueTextField.setText("");
                policyAction.setDelayBoundValue(0);
                delayBoundValueTextField.setEditable(false);
            } // End of else
        } // End of if ( e.getSource() == delayBoundCheckBox )

        if ( e.getSource() == securityLevelCheckBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                securityLevelValueTextField.setEditable(true);
            } // End of if
            else {
                securityLevelValueTextField.setText("");
                policyAction.setSecurityLevelValue(0);
                securityLevelValueTextField.setEditable(false);
            } // End of else
        } // End of if ( e.getSource() == securityLevelCheckBox )

    } // End of method itemStateChanged

/**
 * Method      : resetCheckBoxSelections
 * Purpose     : If the user decides to select deny after he creates
 * other actions, then these actions are cleared since deny cannot
 * be selected with any other other action.
 * Parameters  : None
 */

private void resetCheckBoxSelections () {
    permitCheckBox.setSelected(false);
    priorityCheckBox.setSelected(false);
    hopCountCheckBox.setSelected(false);
    allocatedBWCheckBox.setSelected(false);
    maxLossRateCheckBox.setSelected(false);
    delayBoundCheckBox.setSelected(false);
    securityLevelCheckBox.setSelected(false);
} // End of method resetCheckBoxSelections

/**
 * Method      : setCheckBoxesMethod
 * Purpose     : If the user selects deny action, then all other
 * actions are disabled since deny cannot be selected with any
 * other other action. If the user deselects deny action then other
 * actions can be created again. Thus, they are enabled again.
 * Parameters  : boolean b
 */

private void setCheckBoxesMethod (boolean b) {

```



```

        permitCheckBox.setEnabled(b);
        priorityCheckBox.setEnabled(b);
        hopCountCheckBox.setEnabled(b);
        allocatedBWCheckBox.setEnabled(b);
        maxLossRateCheckBox.setEnabled(b);
        delayBoundCheckBox.setEnabled(b);
        securityLevelCheckBox.setEnabled(b);
    } // End of method setFlagsMethod

} // End of class CheckBoxHandler

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for OK and Cancel button event
 * handling.
 */
private class ButtonHandler implements ActionListener {

    public void actionPerformed ( ActionEvent e) {

        if ( e.getSource() == okButton ) {
            inputError = false;

            if ( ( priorityCheckBox.isSelected () ) ) {
                try {
                    int temp = Integer.parseInt (
priorityValueTextField.getText() );
                    if ( temp >= 1 ) {
                        policyAction.setPriorityValue(temp);
                    } // End of if
                    else { // If the priority value is less than 1
                        JOptionPane.showMessageDialog
(CreatePolicyFourth.this, "Priority value must be 1 or greater.",
"Invalid priority value", JOptionPane.ERROR_MESSAGE);
                        inputError = true;
                    } // End of else
                } // End of try
                catch (NumberFormatException nfe) {
                    JOptionPane.showMessageDialog
(CreatePolicyFourth.this, "Please provide an integer number as a
priority value.", "Invalid input", JOptionPane.ERROR_MESSAGE);
                    inputError = true;
                } // End of catch
            } // End of if ( ( priorityCheckBox.isSelected () ) )
            if ( ( hopCountCheckBox.isSelected () ) ) {
                try {
                    int temp = Integer.parseInt (
hopCountValueTextField.getText() );
                    if ( temp >= 0 ) {
                        policyAction.setHopCountValue(temp);
                    } // End of if
                    else { // If the hop count value is negative
                        JOptionPane.showMessageDialog
(CreatePolicyFourth.this, "Hop count value must be positive.", "Invalid
hop count value", JOptionPane.ERROR_MESSAGE);

```

```

        inputError = true;
    } // End of else
} // End of try
catch (NumberFormatException nfe) {
    JOptionPane.showMessageDialog
(CreatePolicyFourth.this, "Please provide an integer number as a hop
count value.", "Invalid input", JOptionPane.ERROR_MESSAGE);
    inputError = true;
} // End of catch
} // End of if ( ( hopCountCheckBox.isSelected () ) )

if ( ( allocatedBWCheckBox.isSelected () ) ) {
    try {
        float temp = Float.parseFloat (
allocatedBWValueTextField.getText() );
        if ( temp >= 0 ) {
            policyAction.setAllocatedBWValue(temp);
        } // End of if
        else { // If the allocated BW value is negative
            JOptionPane.showMessageDialog
(CreatePolicyFourth.this, "Allocated bw value must be positive.",
"Invalid allocated BW value", JOptionPane.ERROR_MESSAGE);
            inputError = true;
        } // End of else
    } // End of try
    catch (NumberFormatException nfe) {
        JOptionPane.showMessageDialog
(CreatePolicyFourth.this, "Please provide a float number as an
allocated bw value.", "Invalid input", JOptionPane.ERROR_MESSAGE);
        inputError = true;
    } // End of catch
} // End of if ( ( allocatedBWCheckBox.isSelected () ) )
if ( ( maxLossRateCheckBox.isSelected () ) ) {
    try {
        float temp = Float.parseFloat (
maxLossRateValueTextField.getText() );
        if ( temp >= 0 && temp <= 100 ) {
            policyAction.setMaxLossRateValue(temp);
        } // End of if
        else { // If the maximum loss rate value is negative
            JOptionPane.showMessageDialog
(CreatePolicyFourth.this, "Maximum loss rate value must be between 0 -
100.", "Invalid maximum loss rate value", JOptionPane.ERROR_MESSAGE);
            inputError = true;
        } // End of else
    } // End of try
    catch (NumberFormatException nfe) {
        JOptionPane.showMessageDialog
(CreatePolicyFourth.this, "Please provide a float number as a maximum
loss rate value.", "Invalid input", JOptionPane.ERROR_MESSAGE);
        inputError = true;
    } // End of catch
} // End of if ( ( maxLossRateCheckBox.isSelected () ) )
if ( ( delayBoundCheckBox.isSelected () ) ) {
    try {

```

```

        float temp = Float.parseFloat (
delayBoundValueTextField.getText() );
        if ( temp >= 0 ) {
            policyAction.setDelayBoundValue(temp);
        } // End of if
        else { // If the delay bound value is negative
            JOptionPane.showMessageDialog
(CreatePolicyFourth.this, "Delay bound value must be positive.",
"Invalid delay bound value", JOptionPane.ERROR_MESSAGE);
            inputError = true;
        } // End of else
    } // End of try
    catch (NumberFormatException nfe) {
        JOptionPane.showMessageDialog
(CreatePolicyFourth.this, "Please provide a float number as delay bound
value.", "Invalid input", JOptionPane.ERROR_MESSAGE);
        inputError = true;
    } // End of catch
} // End of if ( ( delayBoundCheckBox.isSelected() ) )

    if ( ( securityLevelCheckBox.isSelected() ) ) {
        try {
            int temp = Integer.parseInt (
securityLevelValueTextField.getText() );
            if ( temp >= 1 ) {
                policyAction.setSecurityLevelValue(temp);
            } // End of if
            else { // If the security level value is less than 1
                JOptionPane.showMessageDialog
(CreatePolicyFourth.this, "Security level value must be 1 or greater.",
"Invalid security level value", JOptionPane.ERROR_MESSAGE);
                inputError = true;
            } // End of else
        } // End of try
        catch (NumberFormatException nfe) {
            JOptionPane.showMessageDialog
(CreatePolicyFourth.this, "Please provide an integer number as a
security level value.", "Invalid input", JOptionPane.ERROR_MESSAGE);
            inputError = true;
        } // End of catch
    } // End of if ( ( securityLevelCheckBox.isSelected() ) )

    if ( ( ! denyCheckBox.isSelected() ) && ( !
permitCheckBox.isSelected() ) && ( ! priorityCheckBox.isSelected() ) &&
( ! hopCountCheckBox.isSelected() ) && ( !
allocatedBWCheckBox.isSelected() ) && ( !
maxLossRateCheckBox.isSelected() ) && ( !
delayBoundCheckBox.isSelected() ) && ( !
securityLevelCheckBox.isSelected() ) ) {
        JOptionPane.showMessageDialog (CreatePolicyFourth.this,
"Please select a policy action.", "Action selection needed",
JOptionPane.ERROR_MESSAGE);
        inputError = true;
    }

    if ( ! inputError ) {

```

```

        if ( createPolicyIsCalled ) {
            createPolicyForAction.setNetworkPolicyAction
(policyAction);
        }
        else {
            modifyPolicySecondForAction.setNetworkPolicyAction
(policyAction);
        }
        CreatePolicyFourth.this.dispose();
    } // End of if ( ! inputError )

} // End of if ( e.getSource() == okButton )

if ( e.getSource() == cancelButton ) {
    CreatePolicyFourth.this.dispose();
}

} // End of method actionPerformed

} // End of class ButtonHandler

} // End of class CreatePolicyFourth

```

```

/*****
File: CreatePolicyID.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/23/2001
Description: In this file, I will design the window which will be used
to create a policy ID.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreatePolicyID extends JDialog {
    private DirectorClass boss;

    private CreatePolicy createPolicyForID;

    private ModifyPolicySecond modifyPolicySecondForID;

    private boolean createPolicyIsCalled = false;

    private boolean hasWhiteSpace = false;

    private JLabel policyIDLabel, lineupLabel, lineupLabel2;
    private JTextField policyIDTextField;
    private JButton okButton, cancelButton;
    private JPanel policyPanel, buttonPanel;

    /**
     * Method      : CreatePolicyID
     * Purpose     : Constructor for policy ID creation
     * Parameters  : DirectorClass x, CreatePolicy c
     */

    public CreatePolicyID ( DirectorClass x, CreatePolicy c ) {
        super ( c, "Policy ID Creation", true ); // Parent Frame is
CreatePolicy window (c), and this JDialog is modal
        setLocation (115, 115);
        this.boss = x;
        this.createPolicyForID = c;
        createPolicyIsCalled = true;
        createPolicyIDGuiBuilderMethod ( );
        setResizable (false);
        setSize(350, 160);
        show();
    } // End of constructor for policy ID creation

    /**
     * Method      : CreatePolicyID

```

```

* Purpose      : Constructor for policy ID modification
* Parameters   : DirectorClass x, ModifyPolicySecond c
*/

public CreatePolicyID ( DirectorClass x, ModifyPolicySecond c ) {
    super ( c, "Policy ID Modification", true );
    setLocation (115, 115);
    this.boss = x;
    this.modifyPolicySecondForID = c;
    createPolicyIDGuiBuilderMethod ( );
    setResizable (false);
    setSize(350, 240);
    show();
} // End of constructor for policy ID modification

/**
* Method       : createPolicyIDGuiBuilderMethod
* Purpose      : This method builds the GUI for the creation of
* policy ID.
* Parameters   : None
*/

private void createPolicyIDGuiBuilderMethod () {
    Container c = getContentPane ();
    policyPanel = new JPanel ();

    lineupLabel = new JLabel ("");
    policyPanel.add(lineupLabel);

    policyIDLabel = new JLabel ("Policy ID");
    policyPanel.add(policyIDLabel);

    policyIDTextField = new JTextField (10);
    policyPanel.add(policyIDTextField);

    c.add (policyPanel, BorderLayout.CENTER);

    buttonPanel = new JPanel ();

    ButtonHandler handlerBut = new ButtonHandler ();

    okButton = new JButton (" OK ");
    okButton.addActionListener(handlerBut);
    buttonPanel.add ( okButton );

    lineupLabel2 = new JLabel ("");
    buttonPanel.add(lineupLabel2);

    cancelButton = new JButton ("Cancel");
    cancelButton.addActionListener(handlerBut);
    buttonPanel.add ( cancelButton );

    c.add ( buttonPanel, BorderLayout.SOUTH );
} // End of method createPolicyIDGuiBuilderMethod

```

```

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for OK and Cancel button event
 * handling.
 */
private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == okButton ) {
            if ( (policyIDTextField.getText() ).equals("") ) {
                JOptionPane.showMessageDialog (CreatePolicyID.this,
                "Please provide a policy ID.", "Policy ID needed",
                JOptionPane.ERROR_MESSAGE);
            }
            else if ( !(Character.isLetter ( (
                (policyIDTextField.getText()).charAt (0) ) ) ) ) {
                JOptionPane.showMessageDialog (CreatePolicyID.this, "The
                first letter of policy ID must be a letter.", "First letter warning",
                JOptionPane.ERROR_MESSAGE);
            }
            else if ( whiteSpaceControl ( policyIDTextField.getText() )
            ) {
                JOptionPane.showMessageDialog (CreatePolicyID.this,
                "White spaces are not allowed in policy ID", "White space warning",
                JOptionPane.ERROR_MESSAGE);
            }
            // method policyRedefinitionCheck returns true if there is
            no redefinition
            else if ( !( boss.policyRedefinitionCheck (
            policyIDTextField.getText() ) ) ) {
                JOptionPane.showMessageDialog (CreatePolicyID.this, "A
                policy with the same ID already exists. Please change ID.",
                "Redefinition error", JOptionPane.ERROR_MESSAGE);
            }
            else {
                if ( createPolicyIsCalled ) {
                    createPolicyForID.setNetworkPolicyID (
                    policyIDTextField.getText() ); // set the policy ID in policy creation
                    process
                }
                else {
                    modifyPolicySecondForID.setNetworkPolicyID (
                    policyIDTextField.getText() ); // set the policy ID in policy
                    modification process
                }
                CreatePolicyID.this.dispose();
            }
        }

        } // End of if ( e.getSource() == okButton )

        if (e.getSource() == cancelButton) {
            CreatePolicyID.this.dispose();
        } // End of if

    } // End of method actionPerformed

```

```

/**
 * Method      : whiteSpaceControl
 * Purpose     : This method checks the strings that the user
 * creates to put them in a format that PPL compiler accepts. Policy
 * ID should not contain white space characters. This method returns
 * true if a string has white space character, false otherwise.
 * Parameters  : String s
 */

private boolean whiteSpaceControl (String s) {
    hasWhiteSpace = false;
    char charArray [] = s.toCharArray();
    for ( int i = 0; i < charArray.length; ++i ) {
        if ( Character.isWhitespace ( charArray[i] ) ) {
            hasWhiteSpace = true;
            break;
        } // End of if
    } // End of for
    if ( hasWhiteSpace == true) {
        return true;
    }
    else {
        return false;
    }
} // End of whiteSpaceControl Method

} // End of class ButtonHandler

} // End of class CreatePolicyID

```



```

/*****
File: CreatePolicyMaker.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/26/2001
Description: In this file, the frame that will be used to create a
policy maker will be formed. Only an administrator will be able to
create a policy maker.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreatePolicyMaker extends JDialog {
    private DirectorClass boss;
    private PolicyMaker networkPolicyMaker = new PolicyMaker ("", "",
1);

    private JLabel loginIDLabel, passwordLabel, priorityLabel,
lineupLabel, lineupLabel2, lineupLabel3, lineupLabel4;
    private JLabel lineupLabel5, lineupLabel6, lineupLabel7,
lineupLabel8;
    private JTextField loginIDTextField, priorityTextField;
    private JPasswordField passwordField;
    private JButton okButton, cancelButton;

    private boolean inputError = false;
    private boolean hasWhiteSpace = false;

    /**
     * Method      : CreatePolicyID
     * Purpose      : Constructor for policy maker creation
     * Parameters   : GuiMenu gui, DirectorClass x
     */

    public CreatePolicyMaker ( GuiMenu gui, DirectorClass x ) {
        super ( gui, "Create User", true ); // Parent Frame is main GUI,
and this JDialog is modal
        setLocation (115, 115);
        this.boss = x;
        createPolicyMakerGuiBuilderMethod ( );
        setResizable (false);
        setSize(420, 280);
        show();
    } // End of constructor

    /**
     * Method      : createPolicyMakerGuiBuilderMethod

```

```

* Purpose      : This method builds the GUI for policy maker
* creation.
* Parameters   : None
*/

private void createPolicyMakerGuiBuilderMethod ( ) {
    Container c = getContentPane ();
    c.setLayout( new FlowLayout () );

    loginIDLabel = new JLabel ("User login ID                ");
    c.add(loginIDLabel);

    loginIDTextField = new JTextField (10);
    c.add(loginIDTextField);

    lineupLabel = new JLabel ("                ");
    c.add(lineupLabel);

    lineupLabel5 = new JLabel ("                ");
    c.add(lineupLabel5);

    passwordLabel = new JLabel ("Password                ");
    c.add(passwordLabel);

    passwordField = new JPasswordField (10);
    c.add(passwordField);

    lineupLabel2 = new JLabel ("                ");
    c.add(lineupLabel2);

    lineupLabel6 = new JLabel ("                ");
    c.add(lineupLabel6);

    priorityLabel = new JLabel ("Priority                ");
    c.add(priorityLabel);

    priorityTextField = new JTextField (10);
    c.add(priorityTextField);

    ButtonHandler handlerBut = new ButtonHandler ();

    lineupLabel3 = new JLabel ("                ");
    c.add(lineupLabel3);

    lineupLabel7 = new JLabel ("                ");
    c.add(lineupLabel7);

    lineupLabel8 = new JLabel ("                ");
    c.add(lineupLabel8);

    okButton = new JButton ("    OK    ");
    c.add(okButton);
    okButton.addActionListener(handlerBut);

    lineupLabel4 = new JLabel ("                ");

```

```

        c.add(lineupLabel4);

        cancelButton = new JButton ("Cancel");
        c.add(cancelButton);
        cancelButton.addActionListener(handlerBut);
    } // End of createTypeGuiBuilderMethod

    /**
     * Class          : ButtonHandler
     * Purpose        : Inner class for OK and Cancel button event
     * handling.
     */

    private class ButtonHandler implements ActionListener {
        public void actionPerformed ( ActionEvent e) {

            if ( e.getSource() == okButton ) {
                inputError = false;

                if ( (loginIDTextField.getText() ).equals("") ) {
                    JOptionPane.showMessageDialog (CreatePolicyMaker.this,
                    "Please provide a login ID for the user.", "Login ID needed",
                    JOptionPane.ERROR_MESSAGE);
                    inputError = true;
                }
                else if ( whiteSpaceControl ( loginIDTextField.getText() )
            ) {
                    JOptionPane.showMessageDialog (CreatePolicyMaker.this,
                    "White spaces are not allowed in login ID", "White space warning",
                    JOptionPane.ERROR_MESSAGE);
                    inputError = true;
                }
                // method policyMakerRedefinitionCheck returns false if the
                user is a redefinition
                else if ( !( boss.policyMakerRedefinitionCheck (
                loginIDTextField.getText() ) ) ) { // If there is a redefinition
                    JOptionPane.showMessageDialog (CreatePolicyMaker.this,
                    "A user with the same login id already exists. Please change the login
                    ID.", "Redefinition error", JOptionPane.ERROR_MESSAGE);
                    inputError = true;
                }
                else {
                    networkPolicyMaker.setLoginID (
                    loginIDTextField.getText() );
                }

                char passwordFieldCharArray [] =
                passwordField.getPassword();

                if ( passwordFieldCharArray.length == 0 ) {
                    JOptionPane.showMessageDialog (CreatePolicyMaker.this,
                    "Provide a password for the user.", "Password needed",
                    JOptionPane.ERROR_MESSAGE);
                    inputError = true;
                }
            }
        }
    }

```

```

        else {
            String passwordString = new String
(passwordFieldCharArray);
            networkPolicyMaker.setPasswordOfPolicyMaker(
passwordString );
        }

        try {
            int temp = Integer.parseInt (
priorityTextField.getText() );
            if ( temp >= 1 ) {
                networkPolicyMaker.setPolicyMakerPriority ( temp );
            }
            else {
                JOptionPane.showMessageDialog
(CreatePolicyMaker.this, "Priority value must be 1 or greater.",
"Invalid priority value", JOptionPane.ERROR_MESSAGE);
                inputError = true;
            }
        } // End of try
        catch (NumberFormatException nfe) {
            JOptionPane.showMessageDialog (CreatePolicyMaker.this,
"Please provide an integer as a priority value.", "Invalid number",
JOptionPane.ERROR_MESSAGE);
            inputError = true;
        } // End of catch

        if ( !(inputError) ) {
            boss.addPolicyMaker (networkPolicyMaker); // add the
policy maker to policy maker vector in the Director class.
            CreatePolicyMaker.this.dispose(); // Releases resources
allocated for a context (current instance of CreateType class).
        }

    } // End of if ( e.getSource() == okButton )

    if ( e.getSource() == cancelButton ) {
        CreatePolicyMaker.this.dispose();
    } // End of if ( e.getSource() == cancelButton )

} // End of public void actionPerformed ( ActionEvent e)

/**
 * Method      : whiteSpaceControl
 * Purpose     : This method checks the strings that the user
 * creates to put them in a format that PPL compiler accepts. Login
 * ID should not contain white space characters. This method returns
 * true if a string has white space character, false otherwise.
 */

private boolean whiteSpaceControl (String s) {
    hasWhiteSpace = false;
    char charArray [] = s.toCharArray();
    for ( int i = 0; i < charArray.length; ++i ) {
        if ( Character.isWhitespace ( charArray[i] ) ) {

```

```

        hasWhiteSpace = true;
        break;
    } // End of if
} // End of for
if ( hasWhiteSpace == true) {
    return true;
}
else {
    return false;
}
} // End of whiteSpaceControl Method

} // End of private class ButtonHandler

} // End of CreatePolicyMaker class

```

```

/*****
File: CreatePolicySecond.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/03/2001
Description: In this file, I will design the second part of the GUI
which will be used to create a policy. In this part, the user will
create the target element of a policy.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreatePolicySecond extends JDialog {
    private DirectorClass bossForTarget;

    private CreatePolicy createPolicyForTarget;

    private ModifyPolicySecond modifyPolicySecondForTarget;

    private boolean createPolicyIsCalled = false;

    private Target policyTarget = new Target ();
    private OneTarget oneTargetObject = new OneTarget ();

    Vector tempVectorToSetTargetClassVector = new Vector (1);

    Vector relationVector = new Vector (1);

    private JCheckBox targetAllCheckBox;
    private JComboBox classesComboBox, relationComboBox,
classMembersComboBox;
    private JButton addTargetToPolicyButton, okButton, cancelButton;

    private JLabel targetsAreLabel, lineupLabel, lineupLabel2,
lineupLabel3, lineupLabel4, lineupLabel5, lineupLabel6;
    private JList policyTargetList;

    /**
     * Method      : CreatePolicySecond
     * Purpose      : Constructor for policy target creation
     * Parameters   : DirectorClass x, CreatePolicy c
     */

    public CreatePolicySecond ( DirectorClass x, CreatePolicy c ) {
        super ( c, "Policy Target Creation", true );
        setLocation (115, 115);
        this.bossForTarget = x;
        this.createPolicyForTarget = c;
        createPolicyIsCalled = true;
        createPolicySecondGuiBuilderMethod ();
        setResizable (false);
        setSize(560, 450);
        show();
    } // End of constructor for policy target creation

```

```

/**
 * Method      : CreatePolicySecond
 * Purpose     : Constructor for policy target modification
 * Parameters  : DirectorClass x, ModifyPolicySecond c
 */

public CreatePolicySecond ( DirectorClass x, ModifyPolicySecond c )
{
    super ( c, "Policy Target Modification", true );
    setLocation (115, 115);
    this.bossForTarget = x;
    this.modifyPolicySecondForTarget = c;
    createPolicySecondGuiBuilderMethod ();
    setResizable (false);
    setSize(560, 450);
    show();
} // End of constructor for policy target modification

/**
 * Method      : createPolicySecondGuiBuilderMethod
 * Purpose     : This method builds the GUI for the creation of the
 * second screen (Where the user creates the target element) of
 * policy creation process.
 * Parameters  : None
 */

private void createPolicySecondGuiBuilderMethod () {
    Container c = getContentPane ();
    c.setLayout( new FlowLayout () );

    CheckBoxHandler handler = new CheckBoxHandler ();
    targetAllCheckBox = new JCheckBox ("Target all");
    targetAllCheckBox.addItemListener(handler);
    c.add(targetAllCheckBox);

    lineupLabel = new JLabel ("");
    c.add(lineupLabel);

    lineupLabel2 = new JLabel ("");
    c.add(lineupLabel2);

    classesComboBox = new JComboBox ( bossForTarget.getClassVector()
);

    if ( ( bossForTarget.getClassVector() ).isEmpty() ) {
        classesComboBox.setEnabled(false); // If there is no class to
select, then the class combo box will be disabled.
    }

// The following if statement sets the target class name of the one
target object in create policy second class, when the target element
creation window is first created

```

```

        if ( !( ( bossForTarget.getClassVector() ).isEmpty() ) ) {
            oneTargetObject.setTargetClassName( ( (Class)
classesComboBox.getSelectedItemAt() ).getName() );
        } // End of if

        classesComboBox.setMaximumRowCount(7);
        c.add ( new JScrollPane (classesComboBox) ); // Provide a scroll
bar if there are more than 7 elements in the combo box.
        classesComboBox.addItemListener(handler);
        Dimension d = new Dimension (175, 25); // dimension object
        (width, height) to be used as the dimension of the combo box
        classesComboBox.setPreferredSize(d); // set the size of the combo
box so that it will not be too big or too small

        relationVector.addElement("==");
        relationVector.addElement("!=");
        relationComboBox = new JComboBox (relationVector);

        if ( ( bossForTarget.getClassVector() ).isEmpty() ) {
            relationComboBox.setEnabled(false); // If there is no class to
select, then the realtion combo box will be disabled.
        }

// The following if statement sets the is equal and is not equal data
members of the one target object in create policy second class, when
the target element creation window is first created
        if ( !( ( bossForTarget.getClassVector() ).isEmpty() ) ) {
            oneTargetObject.setIsEqual(true);
            oneTargetObject.setIsNotEqual(false);
        } // End of if

        c.add(relationComboBox);
        relationComboBox.addItemListener(handler);

        classMembersComboBox = new JComboBox ();

        if ( ( bossForTarget.getClassVector() ).isEmpty() ) {
            classMembersComboBox.setEnabled(false); // If there is no
class to select, then the class members combo box will be disabled.
        }

        if ( !( ( bossForTarget.getClassVector() ).isEmpty() ) ) { // if
the class vector in the director class is not empty. If the user has at
least one class then the members of the first class will be displayed
in the class members combo box.
            // The class member vector of the first class (that is
automatically selected from the class combo box when it is first
created is assigned to an enum so that the elements of this member
vector will be used to set the objects in class member combo box
            Enumeration enum = ( ( (Class)
classesComboBox.getSelectedItemAt() ).getClassMembersVector()
).elements();
            while ( enum.hasMoreElements() ) {

```



```

        classMembersComboBox.addItem ( enum.nextElement() ); // Add
the members of the first (selected) class from class combo box to the
class members combo box
    } // End of while
// The following statement sets the target class member name data
member of the one target object in create policy second class, when the
target element creation window (second window of policy creation
wizard) is first created
    oneTargetObject.setTargetClassMemberName( (String)
classMembersComboBox.getSelectedItem() );
    } // End of if

classMembersComboBox.setMaximumRowCount(7);
c.add ( new JScrollPane (classMembersComboBox) );
classMembersComboBox.addItemListener(handler);
classMembersComboBox.setPreferredSize(d);

ButtonHandler handlerBut = new ButtonHandler ();
addTargetToPolicyButton = new JButton ("Add to policy");
if ( ( bossForTarget.getClassVector() ).isEmpty() ) {
    addTargetToPolicyButton.setEnabled(false); // If there is no
class to select, then the add target to policy button will be disabled.
}
addTargetToPolicyButton.addActionListener(handlerBut);
c.add(addTargetToPolicyButton);

lineupLabel3 = new JLabel ("");
c.add(lineupLabel3);

targetsAreLabel = new JLabel ("Policy targets are:");
c.add(targetsAreLabel);

policyTargetList = new JList ();
policyTargetList.setBackground(Color.lightGray);
c.add ( new JScrollPane (policyTargetList) );

lineupLabel4 = new JLabel ("");
c.add(lineupLabel4);

lineupLabel5 = new JLabel ("");
c.add(lineupLabel5);

okButton = new JButton (" OK ");
okButton.addActionListener(handlerBut);
c.add(okButton);

lineupLabel6 = new JLabel ("");
c.add(lineupLabel6);

cancelButton = new JButton ("Cancel");
cancelButton.addActionListener(handlerBut);
c.add(cancelButton);

```

```

} // End of createPolicySecondGuiBuilderMethod

/**
 * Class      : CheckBoxHandler
 * Purpose    : Inner class for check box event handling
 */

private class CheckBoxHandler implements ItemListener {

    public void itemStateChanged( ItemEvent e ) {

        if ( e.getSource() == targetAllCheckBox ) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                policyTarget.setTargetAllProperty(true); // meaning
there is a wild card character in the target element of a policy
                addTargetToPolicyButton.setEnabled(false); // If the
user selects "target all" property then he will not add any class as a
target element to the policy. So add target to policy button is
disabled.
                // If there is any element in the target vector when the
target all check box is selected
                // remove those elements.
                tempVectorToSetTargetClassVector.removeAllElements();
                policyTargetList.setListData (
tempVectorToSetTargetClassVector );
            }
            if ( ( e.getStateChange() == ItemEvent.DESELECTED ) && ( !(
( bossForTarget.getClassVector() ).isEmpty() ) ) ) {
                policyTarget.setTargetAllProperty(false); // meaning
there is not a wild card character in the target element of a policy
                addTargetToPolicyButton.setEnabled(true); // If the user
deselects the target all property and we have at least one class to add
as a target, then add target button is enabled again
            }
        } // End of if (e.getSource() == targetAllCheckBox)

        if (e.getSource() == classesComboBox) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                oneTargetObject.setTargetClassName( ( (Class)
classesComboBox.getSelectedItem() ).getName() );
                classMembersComboBox.removeAllItems(); // Before adding
the members of the newly selected class, I delete the members of the
old selected class from the class members combo box

                Enumeration enum = ( ( (Class)
classesComboBox.getSelectedItem() ).getClassMembersVector()
).elements(); // the class member vector of the class that is selected
from the class combo box is assigned to an enum so that the elements of
this member vector will be used to set the objects in class member
combo box

                while ( enum.hasMoreElements() ) {

```

```

        classMembersComboBox.addItem ( enum.nextElement() );
// Add the members of the selected class from class combo box to the
class members combo box
    } // End of while
    } // End of if ( e.getStateChange()...)
} // End of classesComboBox if

    if (e.getSource() == classMembersComboBox) {
        if ( e.getStateChange() == ItemEvent.SELECTED ) {
            oneTargetObject.setTargetClassName( (String)
classMembersComboBox.getSelectedItem() );
        }
    } // End of classMembersComboBox if

    if (e.getSource() == relationComboBox) {
        if ( e.getStateChange() == ItemEvent.SELECTED ) {
            if ( ( (String) relationComboBox.getSelectedItem()
).equals("") ) {
                oneTargetObject.setIsEqual(true);
                oneTargetObject.setIsNotEqual(false);
            }
            if ( ( (String) relationComboBox.getSelectedItem()
).equals("!=") ) {
                oneTargetObject.setIsNotEqual(true);
                oneTargetObject.setIsEqual(false);
            }
        }
    } // End of relationComboBox if

} // End of itemStateChanged method

} // End of class CheckBoxHandler

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for "Add target to policy", "Next" and
 * "Cancel" button event handling. When the user clicks add target
 * to policy button one target object (which was set according to
 * the combo box choices) is placed in the policy Targets vector
 * (which is a data member of the Target class). When the user
 * clicks next button, current object of the Target class will be
 * sent to network Policy object of the Policy class which will
 * eventually be placed in the policy vector of the director class.
 * If the user checks the Target All check box, since he will not be
 * able to add any one target object in the policy targets vector of
 * the target class, this vector will be empty. Thus target element
 * will be wild card character (*). When the user clicks on the
 * cancel button, policy creation wizard will be dismissed.
 */

```

```

private class ButtonHandler implements ActionListener {

```

```

        public void actionPerformed ( ActionEvent e) {
            if ( e.getSource() == addTargetToPolicyButton ) {
                /* The reason why I create a new OneTarget object is that
each time I add a new target to the vector, I do not want the new
target to make the old ones in the vector the same as itself (new
target). I use this new OneTarget object to put the copy of the
OneTarget object that comes from the GUI in it. And then I add this new
OneTarget object to the vector. So, each new OneTarget object that
comes from the GUI does not affect the ones already in the vector. In
other words, they are not referencing to the same OneTarget object
anymore. */
                OneTarget OneTargetDifferentObject = new OneTarget ();
                OneTargetDifferentObject.copy(oneTargetObject);

                if ( !(
tempVectorToSetTargetClassVector.contains(OneTargetDifferentObject) ) )
                {

tempVectorToSetTargetClassVector.addElement(OneTargetDifferentObject);
                    policyTargetList.setListData(
tempVectorToSetTargetClassVector ); // Display the selected targets in
the policy target JList.
                }
                else {
                    JOptionPane.showMessageDialog (CreatePolicySecond.this,
"Create a different target", "Different target creation needed",
JOptionPane.ERROR_MESSAGE);
                }

            } // End of if ( e.getSource() == addTargetToPolicyButton )

            if ( e.getSource() == okButton ) {
                if ( tempVectorToSetTargetClassVector.isEmpty() && !(
targetAllCheckBox.isSelected() ) ) {
                    JOptionPane.showMessageDialog (CreatePolicySecond.this,
"Either select target all or create a target", "Target creation
needed", JOptionPane.ERROR_MESSAGE);
                }
                else {
                    policyTarget.setPolicyTargets (
tempVectorToSetTargetClassVector );
                    if ( createPolicyIsCalled ) {
                        createPolicyForTarget.setNetworkPolicyTarget (
policyTarget);
                    }
                    else {
                        modifyPolicySecondForTarget.setNetworkPolicyTarget (
policyTarget);
                    }
                    CreatePolicySecond.this.dispose();
                }
            } // End of if ( e.getSource() == okButton )

            if ( e.getSource() == cancelButton ) {
                CreatePolicySecond.this.dispose();
            }
        }
    }

```

```
        } // End of if ( e.getSource() == cancelButton )  
    } // End of method actionPerformed  
} // End of class BittonHandler  
} // End of class CreatePolicySecond
```

```

/*****
File: CreatePolicyThird.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/07/2001
Description: In this file, I will design the third part of the GUI
which will be used to create a policy. The user will create the
condition element of a policy by using this window. The user will be
able to choose no condition option (*) or will be able to choose one or
more of the eight condition types. These eight condition types are
1.User defined type 2.Network element parameter 3.UserID 4.BW 5.Source
IP Address 6.Time 7.Hopcount 8.Priority
Appropriate window will be created according to the selection made by
the user from this CreatePolicyThird window.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreatePolicyThird extends JDialog {
    private GuiMenu guiMenuForCondition;
    private DirectorClass bossForCondition;

    private CreatePolicy createPolicyForCondition;

    private ModifyPolicySecond modifyPolicySecondForCondition;

    private boolean createPolicyIsCalled = false;

    private Condition policyCondition = new Condition ();

    private Vector tempVector = new Vector (1);
    private Vector tempVector2 = new Vector (1);
    private Vector tempVector3 = new Vector (1);
    private Vector tempVector4 = new Vector (1);
    private Vector tempVector5 = new Vector (1);
    private Vector tempVector6 = new Vector (1);
    private Vector tempVector7 = new Vector (1);
    private Vector tempVector8 = new Vector (1);

    private boolean networkHasDelayParameter = false; // This flag shows
if the network (any node, link or path) has a delay () parameter or
not.
    private boolean networkHasLossRateParameter = false; // This flag
shows if the network (any node, link or path) has a loss rate ()
parameter or not.
    private boolean networkHasJitterParameter = false; // This flag
shows if the network (any node, link or path) has a jitter () parameter
or not.
    private boolean networkHasUsedBWParameter = false; // This flag
shows if the network (any node, link or path) has a used bw ()
parameter or not.

```

```

    private Node a; // A node reference for Link class constructor used
in methods networkHasDelayParameter, networkHasLossRateParameter,
networkHasJitterParameter and networkHasUsedBWPParameter.
    private Node b; // A node reference for Link class constructor used
in methods networkHasDelayParameter, networkHasLossRateParameter,
networkHasJitterParameter and networkHasUsedBWPParameter.
    private Vector cons; // A Vector reference for Path class
constructor used in methods networkHasDelayParameter,
networkHasLossRateParameter, networkHasJitterParameter and
networkHasUsedBWPParameter.

    private JCheckBox noConditionCheckBox;
    private JButton priorityButton, hopCountButton, timeButton,
srcIPAddressButton, BWButton, userIDButton;
    private JButton typeButton, parameterButton, nextButton,
cancelButton;
    private JLabel lineupLabel, lineupLabel2, lineupLabel3,
lineupLabel4, lineupLabel5, lineupLabel6, lineupLabel7;
    private JLabel explanationLabel;
    private JPanel mainPanel, nextPanel;

    /**
     * Method      : CreatePolicyThird
     * Purpose     : Constructor for policy condition creation
     * Parameters  : GuiMenu g, DirectorClass d, CreatePolicy c
     */

    public CreatePolicyThird ( GuiMenu g, DirectorClass d, CreatePolicy
c ) {
        super ( g, "Policy Condition Creation", true ); // Parent Frame
is main GUI, and this JDialog is modal
        setLocation (115, 115);
        this.guiMenuForCondition = g;
        this.bossForCondition = d;
        this.createPolicyForCondition = c;
        createPolicyIsCalled = true;
        // Vector data members of the object of Condition class are
instantiated to prevent NullPointerException
        // when other classes will try to set these vectors if they are
empty or add elements to them if they
        // are not empty.
        policyCondition.setTypeConditionVector (tempVector);
        policyCondition.setPriorityConditionVector(tempVector2);
        policyCondition.setHopCountConditionVector(tempVector3);
        policyCondition.setTimeConditionVector(tempVector4);
        policyCondition.setSrcIPAddressConditionVector(tempVector5);
        policyCondition.setBWConditionVector(tempVector6);
        policyCondition.setUserIDConditionVector(tempVector7);
        policyCondition.setParameterConditionVector(tempVector8);

        createPolicyThirdGuiBuilderMethod ();
        setResizable (false);
        setSize(510, 390);
        show();
    } // End of constructor for policy condition creation

```

```

/**
 * Method      : CreatePolicyThird
 * Purpose     : Constructor for policy condition modification
 * Parameters  : GuiMenu g, DirectorClass d, ModifyPolicySecond c
 */

    public CreatePolicyThird ( GuiMenu g, DirectorClass d,
ModifyPolicySecond c ) {
        super ( g, "Policy Condition Modification", true ); // Parent
Frame is main GUI, and this JDialog is modal
        setLocation (115, 115);
        this.guiMenuForCondition = g;
        this.bossForCondition = d;
        this.modifyPolicySecondForCondition = c;
        // Vector data members of the object of Condition class are
instantiated to prevent NullPointerException
        // when other classes will try to set these vectors if they are
empty or add elements to them if they
        // are not empty.
        policyCondition.setTypeConditionVector (tempVector);
        policyCondition.setPriorityConditionVector(tempVector2);
        policyCondition.setHopCountConditionVector(tempVector3);
        policyCondition.setTimeConditionVector(tempVector4);
        policyCondition.setSrcIPAddressConditionVector(tempVector5);
        policyCondition.setBWConditionVector(tempVector6);
        policyCondition.setUserIDConditionVector(tempVector7);
        policyCondition.setParameterConditionVector(tempVector8);

        createPolicyThirdGuiBuilderMethod ();
        setResizable (false);
        setSize(510, 390);
        show();
    } // End of constructor for policy condition modification

/**
 * Method      : createPolicyThirdGuiBuilderMethod
 * Purpose     : This method builds the GUI for the creation of the
 * third screen from which the user will select the codition
 * elements that he will put in the policy condition.
 * Parameters  : None
 */

private void createPolicyThirdGuiBuilderMethod () {
    Container c = getContentPane ();

    mainPanel = new JPanel ();

    CheckBoxHandler handler = new CheckBoxHandler ();
    noConditionCheckBox = new JCheckBox ("No condition");
    noConditionCheckBox.addItemListener(handler);
    mainPanel.add(noConditionCheckBox);

    lineupLabel = new JLabel ("          ");
    mainPanel.add(lineupLabel);

```



```

        lineupLabel7 = new JLabel ("");
        mainPanel.add(lineupLabel7);

        explanationLabel = new JLabel ("Click on the type of condition
you want to create:");
        mainPanel.add(explanationLabel);

        lineupLabel2 = new JLabel ("");
        mainPanel.add(lineupLabel2);

        ButtonHandler handlerBut = new ButtonHandler ();

        priorityButton = new JButton ("Priority");
        priorityButton.addActionListener(handlerBut);
        mainPanel.add(priorityButton);

        hopCountButton = new JButton ("Hop Count");
        hopCountButton.addActionListener(handlerBut);
        mainPanel.add(hopCountButton);

        timeButton = new JButton (" Time ");
        timeButton.addActionListener(handlerBut);
        mainPanel.add(timeButton);

        srcIPAddressButton = new JButton ("Source IP Address");
        srcIPAddressButton.addActionListener(handlerBut);
        mainPanel.add(srcIPAddressButton);

        lineupLabel3 = new JLabel ("");
        mainPanel.add(lineupLabel3);

        lineupLabel4 = new JLabel ("");
        mainPanel.add(lineupLabel4);

        BWButton = new JButton ("Bandwidth");
        BWButton.addActionListener(handlerBut);
        mainPanel.add(BWButton);

        userIDButton = new JButton (" User ID ");
        userIDButton.addActionListener(handlerBut);
        mainPanel.add(userIDButton);

        typeButton = new JButton (" Type ");
        typeButton.addActionListener(handlerBut);
        mainPanel.add(typeButton);
        if ( ( bossForCondition.getTypeVector() ).isEmpty() ) {
            typeButton.setEnabled(false); // If the user did not create
any types before creating policy conditions, then there is no meaning
for him to reach the type condition creation window.
            typeButton.setToolTipText("Type creation needed first");
        }

        parameterButton = new JButton ("Parameter");
        parameterButton.addActionListener(handlerBut);
        mainPanel.add(parameterButton);

```

```

        /* By the networkHasParameterCheck () method call, If the network
        (any node, link or path) has a delay () parameter,
        networkHasDelayParameter flag will be set to true, if not it will be
        set to false. If the network (any node, link or path) has a loss rate
        () parameter, networkHasLossRateParameter flag will be set to true, if
        not it will be set to false. If the network (any node, link or path)
        has a jitter () parameter, networkHasJitterParameter flag will be set
        to true, if not it will be set to false. If the network (any node, link
        or path) has a used bw () parameter, networkHasUsedBWParameter flag
        will be set to true, if not it will be set to false. */
        networkHasParameterCheck ();

```

```

        if ( (! networkHasDelayParameter) && (!
networkHasLossRateParameter) && (! networkHasJitterParameter) && (!
networkHasUsedBWParameter) ) {
            parameterButton.setEnabled(false); // If the network does not
            have delay (), loss rate (), jitter () and used bw () parameters then
            the user cannot create any parameter condition.
            parameterButton.setToolTipText("Delay (), loss rate (), jitter
            () and used bw () parameters are not defined");
        }

```

```

        c.add (mainPanel, BorderLayout.CENTER);

```

```

        lineupLabel5 = new JLabel ("                                ");
        mainPanel.add(lineupLabel5);

```

```

        nextPanel = new JPanel ();

```

```

        nextButton = new JButton (" OK ");
        nextButton.addActionListener(handlerBut);
        nextPanel.add(nextButton);

```

```

        lineupLabel6 = new JLabel ("                                ");
        nextPanel.add(lineupLabel6);

```

```

        cancelButton = new JButton ("Cancel");
        cancelButton.addActionListener(handlerBut);
        nextPanel.add(cancelButton);

```

```

        c.add (nextPanel, BorderLayout.SOUTH);

```

```

    } // End of createPolicySecondGuiBuilderMethod method

```

```

/**
 * Class      : CheckBoxHandler
 * Purpose    : Inner class for check box event handling
 */

```

```

private class CheckBoxHandler implements ItemListener {
    public void itemStateChanged( ItemEvent e ) {
        if ( e.getStateChange() == ItemEvent.SELECTED ) {
            policyCondition.setNoConditionProperty(true); // meaning
there is a wild card character in the condition element of a policy.

```

```

        priorityButton.setEnabled(false); // If the user selects
        "no condition" property then he will not add any condition to the
        condition element of the policy. So 8 buttons which are used to add
        different conditions to the condition element of a policy are disabled.
        hopCountButton.setEnabled(false);
        timeButton.setEnabled(false);
        srcIPAddressButton.setEnabled(false);
        BWButton.setEnabled(false);
        userIDButton.setEnabled(false);
        typeButton.setEnabled(false);
        parameterButton.setEnabled(false);
        /* If the user selects no condition property and he created
        some conditions before this selection, all those conditions are
        cleared. */
        ( policyCondition.getBWConditionVector()
        ).removeAllElements();
        ( policyCondition.getHopCountConditionVector()
        ).removeAllElements();
        ( policyCondition.getParameterConditionVector()
        ).removeAllElements();
        ( policyCondition.getPriorityConditionVector()
        ).removeAllElements();
        ( policyCondition.getSrcIPAddressConditionVector()
        ).removeAllElements();
        ( policyCondition.getTimeConditionVector()
        ).removeAllElements();
        ( policyCondition.getTypeConditionVector()
        ).removeAllElements();
        ( policyCondition.getUserIDConditionVector()
        ).removeAllElements();

        } // End of if
        else {
            policyCondition.setNoConditionProperty(false);

            priorityButton.setEnabled(true);
            hopCountButton.setEnabled(true);
            timeButton.setEnabled(true);
            srcIPAddressButton.setEnabled(true);
            BWButton.setEnabled(true);
            userIDButton.setEnabled(true);
            // If there is a type, then make type button enabled.
            Otherwise stay disabled.
            if ( ! ( ( bossForCondition.getTypeVector() ).isEmpty() ) )
            {
                typeButton.setEnabled(true);
            }
            // If one of network elements has one of four parameters
            then make parameter button enabled, otherwise
            // stay disabled.
            if ( networkHasDelayParameter ||
            networkHasLossRateParameter || networkHasJitterParameter ||
            networkHasUsedBWParameter ) {
                parameterButton.setEnabled(true);
            }

```

```

        } // End of else

    } // End of itemStateChanged method

} // End of class CheckBoxHandler

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for each condition button (total 8
 * buttons), "OK" and cancel buttons event handling
 */
private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == priorityButton ) {
            CreatePriorityCondition priorityConditionObject = new
CreatePriorityCondition ( guiMenuForCondition, policyCondition );
        } // End of if

        if ( e.getSource() == hopCountButton ) {
            CreateHopCountCondition hopCountConditionObject = new
CreateHopCountCondition ( guiMenuForCondition, policyCondition );
        } // End of if

        if ( e.getSource() == timeButton ) {
            CreateTimeCondition timeConditionObject = new
CreateTimeCondition ( guiMenuForCondition, policyCondition );
        } // End of if

        if ( e.getSource() == srcIPAddressButton ) {
            CreateSrcIPAddressCondition srcIPAddressConditionObject =
new CreateSrcIPAddressCondition ( guiMenuForCondition, policyCondition
);
        } // End of if

        if ( e.getSource() == BWButton ) {
            CreateBWCondition BWConditionObject = new CreateBWCondition
( guiMenuForCondition, policyCondition );
        } // End of if

        if ( e.getSource() == userIDButton ) {
            CreateUserIDCondition userIDConditionObject = new
CreateUserIDCondition ( guiMenuForCondition, policyCondition );
        } // End of if

        if ( e.getSource() == typeButton ) {
            CreateTypeCondition typeConditionObject = new
CreateTypeCondition ( guiMenuForCondition, policyCondition,
bossForCondition );
        } // End of if

        if ( e.getSource() == parameterButton ) {
            CreateParameterCondition parameterConditionObject = new
CreateParameterCondition ( guiMenuForCondition, policyCondition,

```

```

networkHasDelayParameter, networkHasLossRateParameter,
networkHasJitterParameter, networkHasUsedBWParameter );
    } // End of if

    if ( e.getSource() == nextButton ) {
        // if "no condition" option is not selected and none of 8
        conditions is defined, then the user is warned to do one of them.
        if ( ! ( policyCondition.getNoConditionProperty() ) && ( (
policyCondition.getPriorityConditionVector() ).isEmpty() ) && ( (
policyCondition.getHopCountConditionVector() ).isEmpty() ) && ( (
policyCondition.getTimeConditionVector() ).isEmpty() ) && ( (
policyCondition.getSrcIPAddressConditionVector() ).isEmpty() ) && ( (
policyCondition.getBWConditionVector() ).isEmpty() ) && ( (
policyCondition.getUserIDConditionVector() ).isEmpty() ) && ( (
policyCondition.getTypeConditionVector() ).isEmpty() ) && ( (
policyCondition.getParameterConditionVector() ).isEmpty() ) ) ) {
            JOptionPane.showMessageDialog (CreatePolicyThird.this,
"No condition is defined.", "No condition warning",
JOptionPane.ERROR_MESSAGE);
        } // End of if
        else {
            if ( createPolicyIsCalled ) {
                createPolicyForCondition.setNetworkPolicyCondition
(policyCondition);
            }
            else {
modifyPolicySecondForCondition.setNetworkPolicyCondition
(policyCondition);
            }
            CreatePolicyThird.this.dispose ();
        } // End of else

    } // End of if ( e.getSource() == nextButton )

    if ( e.getSource() == cancelButton ) {
        CreatePolicyThird.this.dispose();
    } // End of if

    } // End of actionPerformed method

} // End of class ButtonHandler

/**
 * Method      : networkHasParameterCheck
 * Purpose     : If a node, link or a path has a delay (), loss rate
 * (), jitter () or used bw () parameter, then the user can define a
 * condition using any number of these parameters. This method
 * identifies if the network has any of these four parameters or
 * not. If it has appropriate flag is set to true, false otherwise.
 * Parameters  : None
 */

private void networkHasParameterCheck () {

```

```

Enumeration enumNode = ( bossForCondition.getNodeVector()
).elements();

while ( enumNode.hasMoreElements() ) {
    Node tempNode = new Node ();
    tempNode = ( Node ) enumNode.nextElement();
    if ( tempNode.getHasDelayParameter() ) {
        networkHasDelayParameter = true;
    } // End of if
    if ( tempNode.getHasLossRateParameter() ) {
        networkHasLossRateParameter = true;
    } // End of if
    if ( tempNode.getHasJitterParameter() ) {
        networkHasJitterParameter = true;
    } // End of if
    if ( tempNode.getHasUsedBwParameter() ) {
        networkHasUsedBWPParameter = true;
    } // End of if
} // End of while

Enumeration enumLink = ( bossForCondition.getLinkVector()
).elements();

while ( enumLink.hasMoreElements() ) {
    Link tempLink = new Link ("", a, b, 0, false, false, false,
false, false);

    tempLink = ( Link ) enumLink.nextElement();
    if ( tempLink.getHasDelayParameter() ) {
        networkHasDelayParameter = true;
    } // End of if
    if ( tempLink.getHasLossRateParameter() ) {
        networkHasLossRateParameter = true;
    } // End of if
    if ( tempLink.getHasJitterParameter() ) {
        networkHasJitterParameter = true;
    } // End of if
    if ( tempLink.getHasUsedBwParameter() ) {
        networkHasUsedBWPParameter = true;
    } // End of if

} // End of while

Enumeration enumPath = ( bossForCondition.getPathVector()
).elements();

while ( enumPath.hasMoreElements() ) {
    Path tempPath = new Path ("", cons, 0, false, false, false,
false, false);
    tempPath = ( Path ) enumPath.nextElement();
    if ( tempPath.getHasDelayParameter() ) {
        networkHasDelayParameter = true;
    } // End of if
    if ( tempPath.getHasLossRateParameter() ) {

```

```

        networkHasLossRateParameter = true;
    } // End of if
    if ( tempPath.getHasJitterParameter() ) {
        networkHasJitterParameter = true;
    } // End of if
    if ( tempPath.getHasUsedBwParameter() ) {
        networkHasUsedBWParameter = true;
    } // End of if

    } // End of while

    } // End of method networkHasParameterCheck ()

} // End of class CreatePolicyThird

```

```

/*****
File: CreatePriorityCondition.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/12/2001
Description: In this file, I will design the one of the 8 condition
types. By means of this window the user will be able to form policy
conditions by using priority.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreatePriorityCondition extends JDialog {
    private GuiMenu guiMenuForPriorityCondition;
    private Condition policyConditionForPriority;
    // I will set the priority condition vector of condition class
    object (when the user clicks on the "priority"
    // button in the policy condition window) in this class. And other 7
    vectors of the same object of the
    // condition class will be set in 7 different classes and then when
    the user clicks the next button
    // this condition class object will be put in the policy class
    object.

    private Vector tempVectorToSetPriorityConditionVector = new Vector
(1);

    private OnePriority onePriorityObject = new OnePriority ();

    private Vector relationVector = new Vector (1);

    private JLabel priorityLabel, conditionPrioritiesAreLabel;
    private JLabel lineupLabel, lineupLabel2, lineupLabel3,
lineupLabel4, lineupLabel5, lineupLabel6;
    private JComboBox relationComboBox;
    private JTextField priorityValueTextField;
    private JButton addPriorityConditionToPolicyButton, okButton,
cancelButton;
    private JList conditionPriorityList;

    /**
     * Method      : CreatePriorityCondition
     * Purpose     : Constructor for priority condition creation
     * Parameters  : GuiMenu g, Condition c
     */

    public CreatePriorityCondition (GuiMenu g, Condition c ) {
        super ( g, "Priority conditions", true ); // Parent Frame is main
        GUI, and this JDialog is modal
        setLocation (115, 115);
        this.guiMenuForPriorityCondition = g;
        this.policyConditionForPriority = c;
        createPriorityConditionGuiBuilderMethod ();
        setResizable (false);
    }
}

```



```

        setSize(460, 350);
        show();
    } // End of constructor

    /**
     * Method      : createPriorityConditionGuiBuilderMethod
     * Purpose     : This method builds the GUI for the creation of
     * priority conditions
     * Parameters  : None
     */

    private void createPriorityConditionGuiBuilderMethod () {
        Container c = getContentPane ();
        c.setLayout( new FlowLayout () );

        priorityLabel = new JLabel ("Priority");
        c.add(priorityLabel);

        CheckBoxHandler handler = new CheckBoxHandler ();

        relationVector.addElement(">=");
        relationVector.addElement("<=");

        relationComboBox = new JComboBox (relationVector);
        c.add(relationComboBox);
        relationComboBox.addItemListener(handler);

        onePriorityObject.setIsGreaterThanOrEqual(true);
        onePriorityObject.setIsSmallerThanOrEqual(false);

        priorityValueTextField = new JTextField (10);
        c.add(priorityValueTextField);

        ButtonHandler handlerBut = new ButtonHandler ();
        addPriorityConditionToPolicyButton = new JButton ("Add to
policy");
        addPriorityConditionToPolicyButton.addActionListener(handlerBut);
        c.add(addPriorityConditionToPolicyButton);

        lineupLabel = new JLabel ("");
        c.add(lineupLabel);

        conditionPrioritiesAreLabel = new JLabel ("Priority conditions
are: ");
        c.add(conditionPrioritiesAreLabel);

        conditionPriorityList = new JList ();
        conditionPriorityList.setBackground(Color.lightGray);
        c.add ( new JScrollPane (conditionPriorityList) );

        lineupLabel2 = new JLabel ("");
        c.add(lineupLabel2);

```

```

        lineupLabel3 = new JLabel ( "                                ");
        c.add(lineupLabel3);

        okButton = new JButton ( "    OK    ");
        okButton.addActionListener(handlerBut);
        c.add(okButton);

        lineupLabel4 = new JLabel ( "                                ");
        c.add(lineupLabel4);

        cancelButton = new JButton ("Cancel");
        cancelButton.addActionListener(handlerBut);
        c.add(cancelButton);

    } // End of createPriorityConditionGuiBuilderMethod

    /**
     * Class          : CheckBoxHandler
     * Purpose        : Inner class for combo box event handling
     */

    private class CheckBoxHandler implements ItemListener {

        public void itemStateChanged( ItemEvent e ) {

            if (e.getSource() == relationComboBox) {
                if ( e.getStateChange() == ItemEvent.SELECTED ) {

                    if ( ( (String) relationComboBox.getSelectedItem()
).equals(">=") ) {
                        onePriorityObject.setIsGreaterThanOrEqualTo(true);
                        onePriorityObject.setIsSmallerThanOrEqualTo(false);
                    } // End of if

                    if ( ( (String) relationComboBox.getSelectedItem()
).equals("<=") ) {
                        onePriorityObject.setIsGreaterThanOrEqualTo(false);
                        onePriorityObject.setIsSmallerThanOrEqualTo(true);
                    } // End of if

                } // End of if ( e.getStateChange() == ItemEvent.SELECTED )

            } // End of relationComboBox if

        } // End of itemStateChanged method

    } // End of class CheckBoxHandler

    /**
     * Class          : ButtonHandler

```

```

* Purpose      : Inner class for Add priority condition to policy,
* OK and Cancel button event handling. When the user clicks add
* priority condition to policy button, one priority object
* (which was set according to the combo box and text field) is
* placed in the tempVectorToSetPriorityConditionVector vector. When
* the user clicks ok button, current temporary vector which has the
* priority conditions in it will set the priority condition
* vector of the condition class object or add its elements to it.
* When the user clicks on the cancel button, priority condition
* creation window will be dismissed.
*/

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == addPriorityConditionToPolicyButton ) {
            try {
                int temp = Integer.parseInt (
priorityValueTextField.getText() );
                if ( temp >= 1 ) {
                    onePriorityObject.setPriorityValue ( temp );
                    OnePriority OnePriorityDifferentObject = new
OnePriority ();
                    OnePriorityDifferentObject.copy(onePriorityObject);
                    // If the policy maker did not add the same priority
condition before.
                    if ( !(
tempVectorToSetPriorityConditionVector.contains(OnePriorityDifferentObj
ect) ) &&
                        !(
(policyConditionForPriority.getPriorityConditionVector()).contains(OneP
riorityDifferentObject) ) ) {

tempVectorToSetPriorityConditionVector.addElement(OnePriorityDifferentO
bject);

                    conditionPriorityList.setListData(
tempVectorToSetPriorityConditionVector ); // Display the created
OnePriority objects in the JList.
                    priorityValueTextField.setText(""); // Text field
is cleared for the ease of use for the next time
                }
                else {
                    JOptionPane.showMessageDialog
(CreatePriorityCondition.this, "Create a different priority condition",
"Different priority condition needed", JOptionPane.ERROR_MESSAGE);
                }

            } // End of if ( temp >= 1 )
            else { // If the priority value is negative
                JOptionPane.showMessageDialog
(CreatePriorityCondition.this, "Priority value must be 1 or greater.",
"Invalid priority value", JOptionPane.ERROR_MESSAGE);
            } // End of else

        } // End of try
        catch (NumberFormatException nfe) {

```

```

        JOptionPane.showMessageDialog
(CreatePriorityCondition.this, "Please provide an integer as a priority
value.", "Invalid number", JOptionPane.ERROR_MESSAGE);
    } // End of catch

    } // End of if ( e.getSource() ==
addPriorityConditionToPolicyButton )

    if ( e.getSource() == okButton ) {
        if ( tempVectorToSetPriorityConditionVector.isEmpty() ) {
            JOptionPane.showMessageDialog
(CreatePriorityCondition.this, "Please create a priority condition", "
Priority condition needed", JOptionPane.ERROR_MESSAGE);
        }
        else {
            if ( (
policyConditionForPriority.getPriorityConditionVector() ).isEmpty() ) {

policyConditionForPriority.setPriorityConditionVector(tempVectorToSetPr
iorityConditionVector);
            }
            else { // If the priority condition vector of the
condition class object is not empty, of course I do not want to set
this vector (since I will loose data in it). Instead, I will add new
elements to it.
                Enumeration enum =
tempVectorToSetPriorityConditionVector.elements();
                while ( enum.hasMoreElements() ) {
                    OnePriority tempOnePriority = new OnePriority ();
                    tempOnePriority = ( OnePriority )
enum.nextElement();
                    (
policyConditionForPriority.getPriorityConditionVector()
).addElement(tempOnePriority);
                } // End of while
            } // End of else (when the priority condition vector of
conditon class object is not empty)

            CreatePriorityCondition.this.dispose();

        } // End of else

    } // End of if ( e.getSource() == okButton )

    if ( e.getSource() == cancelButton ) {
        CreatePriorityCondition.this.dispose();
    } // End of if ( e.getSource() == cancelButton )

    } // End of method actionPerformed

    } // End of class ButtonHandler

} // End of CreatePriorityCondition class

```

```

/*****
File: CreateSrcIPAddressCondition.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/13/2001
Description: In this file, I will design the one of the 8 condition
types. By means of this window the user will be able to form policy
conditions by using source IP adress.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreateSrcIPAddressCondition extends JDialog {

    private GuiMenu guiMenuForSrcIPAddressCondition;
    private Condition policyConditionForSrcIPAddress;
    /* I will set the srcIPAddressCondition vector of condition class
object (when the user clicks on the "Source IP Address" button in the
policy condition window) in this class. And other 7 vectors of the same
object of the condition class will be set in 7 different classes and
then when the user clicks the next button this condition class object
will be put in the policy class object. */

    private Vector tempVectorToSetSrcIPAddressConditionVector = new
Vector (1);

    private OneSrcIPAddress oneSrcIPAddressObject = new OneSrcIPAddress
();

    private Vector relationVector = new Vector (1);

    private boolean formatIsWrong = false; // formatIsWrong flag is
false when the format is not wrong, and true when the format is wrong.

    private JLabel srcIPAddressLabel, conditionsrcIPAddressesAreLabel;
    private JLabel lineupLabel, lineupLabel2, lineupLabel3;
    private JComboBox relationComboBox;
    private JTextField quadDot1TextField, quadDot2TextField,
quadDot3TextField, quadDot4TextField;
    private JButton addSrcIPAddressConditionToPolicyButton, okButton,
cancelButton;
    private JList conditionSrcIPAddressList;

    /**
     * Method : CreateSrcIPAddressCondition

```

```

* Purpose      : Constructor for Source IP Address condition
* creation
* Parameters   : GuiMenu g, Condition c
*/

public CreateSrcIPAddressCondition (GuiMenu g, Condition c ) {
    super ( g, "Source IP Address conditions", true ); // Parent
    Frame is main GUI, and this JDialog is modal
    setLocation (115, 115);
    this.guiMenuForSrcIPAddressCondition = g;
    this.policyConditionForSrcIPAddress = c;
    createSrcIPAddressConditionGuiBuilderMethod ();
    setResizable (false);
    setSize(450, 350);
    show();
} // End of constructor

/**
* Method      : createSrcIPAddressConditionGuiBuilderMethod
* Purpose     : This method builds the GUI for the creation of
* source IP address conditions
* Parameters  : None
*/

private void createSrcIPAddressConditionGuiBuilderMethod () {
    Container c = getContentPane ();
    c.setLayout( new FlowLayout () );

    srcIPAddressLabel = new JLabel ("Source Ip Address");
    c.add(srcIPAddressLabel);

    CheckBoxHandler handler = new CheckBoxHandler ();

    relationVector.addElement("==");
    relationVector.addElement("!=");

    relationComboBox = new JComboBox (relationVector);
    c.add(relationComboBox);
    relationComboBox.addItemListener(handler);

    oneSrcIPAddressObject.setIsEqual(true);
    oneSrcIPAddressObject.setIsNotEqual(false);

    quadDot1TextField = new JTextField (3);
    c.add(quadDot1TextField);
    quadDot1TextField.setToolTipText("* or 0 - 255");

    quadDot2TextField = new JTextField (3);
    c.add(quadDot2TextField);
    quadDot2TextField.setToolTipText("* or 0 - 255");

    quadDot3TextField = new JTextField (3);
    c.add(quadDot3TextField);
    quadDot3TextField.setToolTipText("* or 0 - 255");
}

```

```

quadDot4TextField = new JTextField (3);
c.add(quadDot4TextField);
quadDot4TextField.setToolTipText("* or 0 - 255");

ButtonHandler handlerBut = new ButtonHandler ();
addSrcIPAddressConditionToPolicyButton = new JButton ("Add to
policy");

addSrcIPAddressConditionToPolicyButton.addActionListener(handlerBut);
c.add(addSrcIPAddressConditionToPolicyButton);

lineupLabel = new JLabel ("");
c.add(lineupLabel);

conditionsrcIPAddressesAreLabel = new JLabel ("Source IP
conditions are: ");
c.add(conditionsrcIPAddressesAreLabel);

conditionSrcIPAddressList = new JList ();
conditionSrcIPAddressList.setBackground(Color.lightGray);
c.add ( new JScrollPane (conditionSrcIPAddressList) );

lineupLabel2 = new JLabel ("");
c.add(lineupLabel2);

okButton = new JButton (" OK ");
okButton.addActionListener(handlerBut);
c.add(okButton);

lineupLabel3 = new JLabel ("");
c.add(lineupLabel3);

cancelButton = new JButton ("Cancel");
cancelButton.addActionListener(handlerBut);
c.add(cancelButton);

} // End of createTimeConditionGuiBuilderMethod

/**
 * Class      : CheckBoxHandler
 * Purpose    : Inner class for combo box event handling
 */

private class CheckBoxHandler implements ItemListener {

    public void itemStateChanged( ItemEvent e ) {

        if (e.getSource() == relationComboBox) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {

                if ( (String) relationComboBox.getSelectedItem()
).equals("") ) {
                    oneSrcIPAddressObject.setIsEqual(true);
                    oneSrcIPAddressObject.setIsNotEqual(false);

```

```

        } // End of if

        if ( ( (String) relationComboBox.getSelectedItem()
).equals("!=") ) {
            oneSrcIPAddressObject.setIsEqual(false);
            oneSrcIPAddressObject.setIsNotEqual(true);
        } // End of if

    } // End of if ( e.getStateChange() == ItemEvent.SELECTED )

} // End of relationComboBox if

} // End of itemStateChanged method

} // End of class CheckBoxHandler

```

```

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for Add source IP address condition to
 * policy, OK and Cancel button event handling. When the user clicks
 * add source IP Address condition to policy button,
 * oneSrcIPAddressObject (which was set according to the combo box
 * and text field) is placed in the
 * tempVectorToSetSrcIPAddressConditionVector vector. When the user
 * clicks ok button, current temporary vector which has the source
 * Ip Address conditions in it will set the source IP address
 * condition vector of the condition class object or add its
 * elements to it. When the user clicks on the cancel button, time
 * condition creation window will be dismissed.
 */

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == addSrcIPAddressConditionToPolicyButton )
        {

            formatIsWrong = false;

            if ( ( quadDot1TextField.getText() ).equals("") ) {
                oneSrcIPAddressObject.setQuadDot1("");
            }
            else {
                try {
                    int temp = Integer.parseInt (
quadDot1TextField.getText() );
                    if ( temp >= 0 && temp <= 255 ) {
                        oneSrcIPAddressObject.setQuadDot1(
quadDot1TextField.getText() );
                    } // End of if
                    else { // If the source IP address value is not right
                        formatIsWrong = true;
                        JOptionPane.showMessageDialog
(CreateSrcIPAddressCondition.this, "A field in a source IP address must

```



```

be between 0 - 255.", "Invalid number in field 1",
JOptionPane.ERROR_MESSAGE);
    } // End of else
} // End of try

    catch (NumberFormatException nfe) {
        formatIsWrong = true;
        JOptionPane.showMessageDialog
(CreateSrcIPAddressCondition.this, "A field in a source IP address must
be either * or a number between 0-255", "Invalid number in field 1",
JOptionPane.ERROR_MESSAGE);
    } // End of catch
} // End of else

    if ( ( quadDot2TextField.getText() ).equals("*") ) {
        oneSrcIPAddressObject.setQuadDot2("*");
    }
    else {
        try {
            int temp = Integer.parseInt (
quadDot2TextField.getText() );
            if ( temp >= 0 && temp <= 255 ) {
                oneSrcIPAddressObject.setQuadDot2(
quadDot2TextField.getText() );
            } // End of if
            else { // If the source IP address value is not right
                formatIsWrong = true;
                JOptionPane.showMessageDialog
(CreateSrcIPAddressCondition.this, "A field in a source IP address must
be between 0 - 255.", "Invalid number in field 2",
JOptionPane.ERROR_MESSAGE);
            } // End of else
        } // End of try

        catch (NumberFormatException nfe) {
            formatIsWrong = true;
            JOptionPane.showMessageDialog
(CreateSrcIPAddressCondition.this, "A field in a source IP address must
be either * or a number between 0-255", "Invalid number in field 2",
JOptionPane.ERROR_MESSAGE);
        } // End of catch
    } // End of else

    if ( ( quadDot3TextField.getText() ).equals("*") ) {
        oneSrcIPAddressObject.setQuadDot3("*");
    }
    else {
        try {
            int temp = Integer.parseInt (
quadDot3TextField.getText() );
            if ( temp >= 0 && temp <= 255 ) {
                oneSrcIPAddressObject.setQuadDot3(
quadDot3TextField.getText() );
            } // End of if

```

```

        else { // If the source IP address value is not right
            formatIsWrong = true;
            JOptionPane.showMessageDialog
(CreateSrcIPAddressCondition.this, "A field in a source IP address must
be between 0 - 255.", "Invalid number in field 3",
JOptionPane.ERROR_MESSAGE);
        } // End of else
    } // End of try

    catch (NumberFormatException nfe) {
        formatIsWrong = true;
        JOptionPane.showMessageDialog
(CreateSrcIPAddressCondition.this, "A field in a source IP address must
be either * or a number between 0-255", "Invalid number in field 3",
JOptionPane.ERROR_MESSAGE);
    } // End of catch
} // End of else

    if ( ( quadDot4TextField.getText() ).equals("") ) {
        oneSrcIPAddressObject.setQuadDot4("");
    }
    else {
        try {
            int temp = Integer.parseInt (
quadDot4TextField.getText() );
            if ( temp >= 0 && temp <= 255 ) {
                oneSrcIPAddressObject.setQuadDot4(
quadDot4TextField.getText() );
            } // End of if
            else { // If the source IP address value is not right
                formatIsWrong = true;
                JOptionPane.showMessageDialog
(CreateSrcIPAddressCondition.this, "A field in a source IP address must
be between 0 - 255.", "Invalid number in field 4",
JOptionPane.ERROR_MESSAGE);
            } // End of else
        } // End of try

        catch (NumberFormatException nfe) {
            formatIsWrong = true;
            JOptionPane.showMessageDialog
(CreateSrcIPAddressCondition.this, "A field in a source IP address must
be either * or a number between 0-255", "Invalid number in field 4",
JOptionPane.ERROR_MESSAGE);
        } // End of catch
    } // End of else

    if ( ! formatIsWrong ) { // If the user did not enter wrong
formatted source IP address (in any of the 4 fields), then that valid
IP address will be added to the vector. Otherwise there is no need for
any action, the user will again input IP address.
        OneSrcIPAddress oneSrcIPAddressDifferentObject = new
OneSrcIPAddress ();

```

```

oneSrcIPAddressDifferentObject.copy(oneSrcIPAddressObject);

        // If the policy maker did not add the same srcIPAddress
condition before.
        if ( !(
tempVectorToSetSrcIPAddressConditionVector.contains(oneSrcIPAddressDiff
erentObject) ) &&
            !(
(policyConditionForSrcIPAddress.getSrcIPAddressConditionVector()).conta
ins(oneSrcIPAddressDifferentObject) ) ) {

tempVectorToSetSrcIPAddressConditionVector.addElement(oneSrcIPAddressDi
fferentObject);
        conditionSrcIPAddressList.setListData(
tempVectorToSetSrcIPAddressConditionVector ); // Display the selected
OneSrcIPAddress objects in the JList.
        quadDot1TextField.setText(""); // Text Fields are
cleared for the ease of use for the next data
        quadDot2TextField.setText("");
        quadDot3TextField.setText("");
        quadDot4TextField.setText("");
        }
        else {
            JOptionPane.showMessageDialog
(CreateSrcIPAddressCondition.this, "Create a different source IP
address condition", "Different source IP address condition needed",
JOptionPane.ERROR_MESSAGE);
        }

        } // End of if ( ! formatIsWrong )

    } // End of if ( e.getSource() ==
addTimeConditionToPolicyButton )

    if ( e.getSource() == okButton ) {
        if ( tempVectorToSetSrcIPAddressConditionVector.isEmpty()
) {
            JOptionPane.showMessageDialog
(CreateSrcIPAddressCondition.this, "Please create a source IP address
condition", " source IP adress condition needed",
JOptionPane.ERROR_MESSAGE);
        }
        else {
            if ( (
policyConditionForSrcIPAddress.getSrcIPAddressConditionVector()
).isEmpty() ) {

policyConditionForSrcIPAddress.setSrcIPAddressConditionVector(tempVecto
rToSetSrcIPAddressConditionVector);
            }
            else { // If the source IP address condition vector of
the condition class object is not empty, of course I do not want to set

```

```

this vector (since I will loose data in it). Instead, I will add new
elements to it.
        Enumeration enum =
tempVectorToSetSrcIPAddressConditionVector.elements();
        while ( enum.hasMoreElements() ) {
            OneSrcIPAddress tempOneSrcIPAddress = new
OneSrcIPAddress ();
            tempOneSrcIPAddress = ( OneSrcIPAddress )
enum.nextElement();
            (
policyConditionForSrcIPAddress.getSrcIPAddressConditionVector()
).addElement(tempOneSrcIPAddress);
            } // End of while
        } // End of else (when the source IP address condition
vector of conditon class object is not empty)

        CreateSrcIPAddressCondition.this.dispose();

    } // End of else

} // End of if ( e.getSource() == okButton )

if ( e.getSource() == cancelButton ) {
    CreateSrcIPAddressCondition.this.dispose();
} // End of if ( e.getSource() == cancelButton )

} // End of method actionPerformed

} // End of class BittonHandler

} // End of class CreateSrcIPAddressCondition

```

```

/*****
File: CreateTimeCondition.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/13/2001
Description: In this file, I will design the one of the 8 condition
types. By means of this window the user will be able to form policy
conditions by using time.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreateTimeCondition extends JDialog {

    private GuiMenu guiMenuForTimeCondition;
    private Condition policyConditionForTime;
    // I will set the timeCondition vector of condition class object
    (when the user clicks on the "time"
    // button in the policy condition window) in this class. And other 7
    vectors of the same object of the
    // condition class will be set in 7 different classes and then when
    the user clicks the next button
    // this condition class object will be put in the policy class
    object.

    private Vector tempVectorToSetTimeConditionVector = new Vector (1);

    private OneTime oneTimeObject = new OneTime ();

    private Vector relationVector = new Vector (1);

    private JLabel timeLabel, conditionTimesAreLabel;
    private JLabel lineupLabel, lineupLabel2, lineupLabel3,
lineupLabel4;
    private JComboBox relationComboBox;
    private JTextField timeValueTextField;
    private JButton addTimeConditionToPolicyButton, okButton,
cancelButton;
    private JList conditionTimeList;

    /**
     * Method      : CreateTimeCondition
     * Purpose     : Constructor for time condition creation
     * Parameters  : GuiMenu g, Condition c
     */

    public CreateTimeCondition (GuiMenu g, Condition c ) {
        super ( g, "Time conditions", true ); // Parent Frame is main
        GUI, and this JDialog is modal
        setLocation (115, 115);
        this.guiMenuForTimeCondition = g;
        this.policyConditionForTime = c;
        createTimeConditionGuiBuilderMethod ();
        setResizable (false);
    }
}

```

```

        setSize(460, 350);
        show();
    } // End of constructor

    /**
     * Method      : createTimeConditionGuiBuilderMethod
     * Purpose     : This method builds the GUI for the creation of time
     * conditions
     * Parameters  : None
     */

    private void createTimeConditionGuiBuilderMethod () {
        Container c = getContentPane ();
        c.setLayout( new FlowLayout () );

        timeLabel = new JLabel ("Time ");
        c.add(timeLabel);

        CheckBoxHandler handler = new CheckBoxHandler ();

        relationVector.addElement(">=");
        relationVector.addElement("<=");

        relationComboBox = new JComboBox (relationVector);
        c.add(relationComboBox);
        relationComboBox.addItemListener(handler);

        oneTimeObject.setIsGreaterThanOrEqual(true);
        oneTimeObject.setIsSmallerThanOrEqual(false);

        timeValueTextField = new JTextField (10);
        c.add(timeValueTextField);
        timeValueTextField.setToolTipText("0000 - 2359");

        ButtonHandler handlerBut = new ButtonHandler ();
        addTimeConditionToPolicyButton = new JButton ("Add to policy");
        addTimeConditionToPolicyButton.addActionListener(handlerBut);
        c.add(addTimeConditionToPolicyButton);

        lineupLabel = new JLabel ("");
        c.add(lineupLabel);

        conditionTimesAreLabel = new JLabel ("Time conditions are: ");
        c.add(conditionTimesAreLabel);

        conditionTimeList = new JList ();
        conditionTimeList.setBackground(Color.lightGray);
        c.add ( new JScrollPane (conditionTimeList) );

        lineupLabel2 = new JLabel ("");
        c.add(lineupLabel2);

        lineupLabel3 = new JLabel ("");
        c.add(lineupLabel3);
    }

```

```

        okButton = new JButton ( "   OK   " );
        okButton.addActionListener(handlerBut);
        c.add(okButton);

        lineupLabel4 = new JLabel ( "                                " );
        c.add(lineupLabel4);

        cancelButton = new JButton ("Cancel");
        cancelButton.addActionListener(handlerBut);
        c.add(cancelButton);

    } // End of createTimeConditionGuiBuilderMethod

    /**
     * Class      : CheckBoxHandler
     * Purpose    : Inner class for combo box event handling
     */

    private class CheckBoxHandler implements ItemListener {

        public void itemStateChanged( ItemEvent e ) {

            if (e.getSource() == relationComboBox) {
                if ( e.getStateChange() == ItemEvent.SELECTED ) {

                    if ( ( (String) relationComboBox.getSelectedItem()
).equals(">=") ) {
                        oneTimeObject.setIsGreaterThanOrEqualTo(true);
                        oneTimeObject.setIsSmallerThanOrEqualTo(false);
                    } // End of if

                    if ( ( (String) relationComboBox.getSelectedItem()
).equals("<=") ) {
                        oneTimeObject.setIsGreaterThanOrEqualTo(false);
                        oneTimeObject.setIsSmallerThanOrEqualTo(true);
                    } // End of if

                } // End of if ( e.getStateChange() == ItemEvent.SELECTED )

            } // End of relationComboBox if

        } // End of itemStateChanged method

    } // End of class CheckBoxHandler

    /**
     * Class      : ButtonHandler

```

```

* Purpose      : Inner class for Add time condition to policy, OK
* and Cancel button event handling. When the user clicks add time
* condition to policy button, oneTimeObject (which was set
* according to the combo box and text field) is placed in the
* tempVectorToSetTimeConditionVector vector. When the user clicks
* ok button, current temporary vector which has the time conditions
* in it will set the time condition vector of the condition class
* object or add its elements to it. When the user clicks on the
* cancel button, time condition creation window will be dismissed.
*/

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == addTimeConditionToPolicyButton ) {
            try {
                int temp = Integer.parseInt (
timeValueTextField.getText() );
                if ( temp >= 0 && temp <= 2359) {
                    oneTimeObject.setTimeValue ( temp );

/* The reason why I create a new OneTimeObject is that each time I add
a new oneTime to the vector, I do not want the new oneTime to make the
old ones in the vector the same as itself. I use this new OneTime
object to put the copy (clone) of the OneTime object that comes from
the GUI. And then I add this new OneTime object to the vector. So, each
new OneTime object that comes from the GUI does not affect the ones
already in the vector. In other words, they are not referencing to the
same OneTime object anymore. */

                    OneTime oneTimeDifferentObject = new OneTime ();
                    oneTimeDifferentObject.copy(oneTimeObject);

                    // If the policy maker did not add the same time
condition before.
                    if ( !(
tempVectorToSetTimeConditionVector.contains(oneTimeDifferentObject) )
&&
                        !(
(policyConditionForTime.getTimeConditionVector()).contains(oneTimeDiffe
rentObject) ) ) {

tempVectorToSetTimeConditionVector.addElement(oneTimeDifferentObject);
                    conditionTimeList.setListData(
tempVectorToSetTimeConditionVector ); // Display the selected OneTime
objects in the JList.
                    timeValueTextField.setText(""); // Text field is
cleared for the ease of use for the next time
                }
                else {
                    JOptionPane.showMessageDialog
(CreateTimeCondition.this, "Create a different time condition",
"Different time condition needed", JOptionPane.ERROR_MESSAGE);
                }
            } // End of if ( temp >= 0 && temp <= 2359)
            else { // If the time format is not right

```



```

        JOptionPane.showMessageDialog (CreateTimeCondition.this,
        "Time value must be between 0000 - 2359.", "Invalid time format",
        JOptionPane.ERROR_MESSAGE);
    } // End of else

    } // End of try

    catch (NumberFormatException nfe) {
        JOptionPane.showMessageDialog
        (CreateTimeCondition.this, "Please provide an integer as a time
        value.", "Invalid number", JOptionPane.ERROR_MESSAGE);
    } // End of catch

    } // End of if ( e.getSource() ==
    addTimeConditionToPolicyButton )

    if ( e.getSource() == okButton ) {
        if ( tempVectorToSetTimeConditionVector.isEmpty() ) {
            JOptionPane.showMessageDialog (CreateTimeCondition.this,
            "Please create a time condition", " Time condition needed",
            JOptionPane.ERROR_MESSAGE);
        }
        else {
            if ( ( policyConditionForTime.getTimeConditionVector()
            ).isEmpty() ) {

policyConditionForTime.setTimeConditionVector(tempVectorToSetTimeCondit
ionVector);

            }
            else { // If the time condition vector of the condition
            class object is not empty, of course I do not want to set this vector
            (since I will loose data in it). Instead, I will add new elements to
            it.

                Enumeration enum =
tempVectorToSetTimeConditionVector.elements();
                while ( enum.hasMoreElements() ) {
                    OneTime tempOneTime = new OneTime ();
                    tempOneTime = ( OneTime ) enum.nextElement();
                    ( policyConditionForTime.getTimeConditionVector()
                    ).addElement(tempOneTime);
                } // End of while
            } // End of else (when the time condition vector of
            conditon class object is not empty)

                CreateTimeCondition.this.dispose();

            } // End of else

    } // End of if ( e.getSource() == okButton )

    if ( e.getSource() == cancelButton ) {
        CreateTimeCondition.this.dispose();
    } // End of if ( e.getSource() == cancelButton )

```

```
        } // End of method actionPerformed  
    } // End of class ButtonHandler  
} // End of class CreateTimeCondition
```

```

/*****
File: CreateType.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/30/2001
Description: In this file, the frame that will be used to create a user
defined type will be formed.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreateType extends JDialog {

    private DirectorClass boss;
    private Vector tempTypeMemberVectorForEachType = new Vector (1);
    private Vector a; // A Vector reference for Class class constructor
    private Type networkType = new Type ("", a);

    private JLabel typeNameLabel, typeMemberLabel, membersAreLabel,
lineupLabel, lineupLabel2, lineupLabel3;
    private JTextField typeNameTextField, typeMemberTextField;
    private JButton addTypeMemberButton, okButton, cancelButton;
    private JList typeMembersList;

    private boolean hasWhiteSpace = false;

    /**
     * Method      : CreateType
     * Purpose     : Constructor for CreateType class
     * Parameters  : GuiMenu gui, DirectorClass x
     */

    public CreateType ( GuiMenu gui, DirectorClass x ) {
        super ( gui, "Create Type", true ); // Parent Frame is main GUI,
and this JDialog is modal
        setLocation (115, 115);
        this.boss = x;
        createTypeGuiBuilderMethod ( );
        setResizable (false);
        setSize(420, 350);
        show();
    } // End of constructor

    /**
     * Method      : createTypeGuiBuilderMethod
     * Purpose     : This method builds the GUI for type creation.
     * Parameters  : None
     */

    private void createTypeGuiBuilderMethod ( ) {
        Container c = getContentPane ( );
        c.setLayout( new FlowLayout ( ) );

        typeNameLabel = new JLabel ("Type name

```

```

c.add(typeNameLabel);

typeNameTextField = new JTextField (10);
c.add(typeNameTextField);

lineupLabel = new JLabel ("");
c.add(lineupLabel);

typeMemberLabel = new JLabel ("Type member");
c.add(typeMemberLabel);

typeMemberTextField = new JTextField (10);
c.add(typeMemberTextField);

ButtonHandler handlerBut = new ButtonHandler ();

addTypeMemberButton = new JButton (" Add type member ");
addTypeMemberButton.addActionListener(handlerBut);
c.add(addTypeMemberButton);

membersAreLabel = new JLabel ("Type members are: ");
c.add(membersAreLabel);

typeMembersList = new JList ( );
typeMembersList.setBackground(Color.lightGray);
c.add ( new JScrollPane (typeMembersList) );

lineupLabel2 = new JLabel ("");
c.add(lineupLabel2);

okButton = new JButton (" OK ");
c.add(okButton);
okButton.addActionListener(handlerBut);

lineupLabel3 = new JLabel ("");
c.add(lineupLabel3);

cancelButton = new JButton ("Cancel");
c.add(cancelButton);
cancelButton.addActionListener(handlerBut);
} // End of createTypeGuiBuilderMethod

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for Add type member to class, OK and
 * Cancel button event handling. This method checks the validity of
 * the type creation. For example: empty type name, empty type
 * member and white spaces are all checked. And after the checks
 * calls the addRecord method for adding the type and then destroys
 * the creation window.
 */

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == addTypeMemberButton ) {

```

```

        if ( ( typeMemberTextField.getText() ).equals("") ) {
            JOptionPane.showMessageDialog (CreateType.this, "Please
provide a member name.", "Member name needed",
JOptionPane.ERROR_MESSAGE);
        }
        else if ( whiteSpaceControl ( typeMemberTextField.getText()
) ) {
            JOptionPane.showMessageDialog (CreateType.this, "White
spaces are not allowed in type member name", "White space warning",
JOptionPane.ERROR_MESSAGE);
        }
        else if ( tempTypeMemberVectorForEachType.contains(
typeMemberTextField.getText() ) ) {
            JOptionPane.showMessageDialog (CreateType.this, "Please
create different type members.", "Invalid type member",
JOptionPane.ERROR_MESSAGE);
        }
        else {
            tempTypeMemberVectorForEachType.addElement(
typeMemberTextField.getText() );
            typeMembersList.setListData
(tempTypeMemberVectorForEachType);
            typeMemberTextField.setText(""); // clear the text field
to make it easy for user's editing the next type member (if there are
more than 1 members)
        }
    }

    if ( e.getSource() == okButton ) {
        if ( (typeNameTextField.getText() ).equals("") ) {
            JOptionPane.showMessageDialog (CreateType.this, "Please
provide a type name.", "Type name needed", JOptionPane.ERROR_MESSAGE);
        }
        else if ( tempTypeMemberVectorForEachType.isEmpty() ) {
            JOptionPane.showMessageDialog (CreateType.this, "Please
provide type member(s).", "Type member needed",
JOptionPane.ERROR_MESSAGE);
        }
        else if ( whiteSpaceControl ( typeNameTextField.getText() )
) {
            JOptionPane.showMessageDialog (CreateType.this, "White
spaces are not allowed in type name", "White space warning",
JOptionPane.ERROR_MESSAGE);
        }
        else {
            addRecord();
        }
    } // End of if ( e.getSource() == okButton )

    if ( e.getSource() == cancelButton ) {
        CreateType.this.dispose();
    } // End of if ( e.getSource() == cancelButton )

} // End of public void actionPerformed ( ActionEvent e)

```

```

/**
 * Method      : whiteSpaceControl
 * Purpose     : This method checks the strings that the user
 * creates to put them in a format that PPL compiler accepts. Type
 * name and member names should not contain white space characters.
 * This method returns true if a string has white space character,
 * false otherwise.
 * Parameters  : String s
 */

private boolean whiteSpaceControl (String s) {
    hasWhiteSpace = false;
    char charArray [] = s.toCharArray();
    for ( int i = 0; i < charArray.length; ++i ) {
        if ( Character.isWhitespace ( charArray[i] ) ) {
            hasWhiteSpace = true;
            break;
        } // End of if
    } // End of for
    if ( hasWhiteSpace == true) {
        return true;
    }
    else {
        return false;
    }
} // End of whiteSpaceControl Method

} // End of private class ButtonHandler implements ActionListener

/**
 * Method      : addRecord
 * Purpose     : This method adds a type to the type vector, if that
 * type is valid. Otherwise, warns the policy maker about the
 * redefinition problem.
 * Parameters  : None
 */

private void addRecord () {
    networkType.setName ( typeNameTextField.getText() ); // set the
name of the type
    networkType.setTypeMembersVector
(tempTypeMemberVectorForEachType); // set the type member vector of
type

    // Newly created type is checked against the previous ones to see
if a redefinition occurs or not.
    if ( boss.typeRedefinitionCheck (networkType) ) { // method
typeRedefinitionCheck returns true if there is no redefinition
        boss.addType (networkType); // add the type to type vector in
the Director class.
        this.dispose(); // Releases resources allocated for a context
(current instance of CreateType class).
    }
    else {

```

```
        JOptionPane.showMessageDialog (this, "A type with the same  
name already exists. Please change the name.", "Redefinition error",  
JOptionPane.ERROR_MESSAGE);  
    }  
  
    } // End of public void addRecord ()  
  
} // End of CreateType class
```

```

/*****
File: CreateTypeCondition.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/08/2001
Description: In this file, I will design the one of the 8 condition
types. By means of this window the user will be able to form policy
conditions by using user defined types.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreateTypeCondition extends JDialog {
    private GuiMenu guiMenuForConditionType;
    private DirectorClass bossForTypePartOfConditon; // To get the
current type vector in the director class
    private Condition policyConditionForType;

    /* I will set the type condition vector of condition class object (when
the user clicks on the "type" button in the policy condition window) in
this class. And other 7 vectors of the same object of the condition
class will be set in 7 different classes and then when the user clicks
the next button this condition class object will be put in the policy
class object. */

    private Vector tempVectorToSetTypeConditionVector = new Vector (1);

    private OneType oneTypeObject = new OneType ();

    private Vector relationVector = new Vector (1);

    private JComboBox typesComboBox, relationComboBox,
typeMembersComboBox;
    private JButton addTypeConditionToPolicyButton, okButton,
cancelButton;

    private JLabel conditionTypesAreLabel, lineupLabel, lineupLabel2,
lineupLabel3, lineupLabel4, lineupLabel5, lineupLabel6;
    private JList conditionTypeList;

    /**
     * Method      : CreateTypeCondition
     * Purpose     : Constructor for type condition creation
     * Parameters  : GuiMenu g, Condition c, DirectorClass d
     */

    public CreateTypeCondition (GuiMenu g, Condition c, DirectorClass d
) {
        super ( g, "Type conditions", true ); // Parent Frame is main
GUI, and this JDialog is modal
        setLocation (115, 115);
        this.guiMenuForConditionType = g;
        this.policyConditionForType = c;
        this.bossForTypePartOfConditon = d;

```



```

        createTypeConditionGuiBuilderMethod ();
        setResizable (false);
        setSize(560, 450);
        show();
    } // End of constructor

    /**
     * Method      : createTypeConditionGuiBuilderMethod
     * Purpose     : This method builds the GUI for the creation of type
     * conditions
     * Parameters  : None
     */

    private void createTypeConditionGuiBuilderMethod () {
        Container c = getContentPane ();
        c.setLayout( new FlowLayout () );

        typesComboBox = new JComboBox (
            bossForTypePartOfConditon.getTypeVector() );

        oneTypeObject.setConditionTypeName ( ( (Type)
            typesComboBox.getSelectedItem() ).getName() );

        CheckBoxHandler handler = new CheckBoxHandler ();

        typesComboBox.setMaximumRowCount(7);
        c.add ( new JScrollPane (typesComboBox) ); // Provide a scroll
        bar if there are more than 7 elements in the combo box.
        typesComboBox.addItemListener(handler);
        Dimension d = new Dimension (175, 25); // dimension object
        (width, height) to be used as the dimension of the combo box
        typesComboBox.setPreferredSize(d); // set the size of the combo
        box so that it will not be too big or too small

        relationVector.addElement("==");
        relationVector.addElement("!=");
        relationVector.addElement(">=");
        relationVector.addElement("<=");
        relationComboBox = new JComboBox (relationVector);

        oneTypeObject.setIsEqual(true);
        oneTypeObject.setIsNotEqual(false);
        oneTypeObject.setIsGreaterThanOrEqual(false);
        oneTypeObject.setIsSmallerThanOrEqual(false);

        c.add(relationComboBox);
        relationComboBox.addItemListener(handler);

        typeMembersComboBox = new JComboBox ();

        /* The type member vector of the first type (that is automatically
        selected from the type combo box when it is first created is assigned

```

```

to an enum so that the elements of this member vector will be used to
set the objects in type member combo box. */
    Enumeration enum = ( (Type) typesComboBox.getSelectedItem()
).getTypeMembersVector() ).elements();
    while ( enum.hasMoreElements() ) {
        typeMembersComboBox.addItem ( enum.nextElement() ); // Add the
members of the first (selected) type from type combo box to the type
members combo box
    } // End of while
/* The following statement sets the target class member name data
member of the one target object in create policy second class, when the
target element creation window (second window of policy creation
wizard) is first created. */
    oneTypeObject.setConditionTypeMemberName ( (String)
typeMembersComboBox.getSelectedItem() );

typeMembersComboBox.setMaximumRowCount(7);
c.add ( new JScrollPane (typeMembersComboBox) );
typeMembersComboBox.addItemListener(handler);
typeMembersComboBox.setPreferredSize(d);

ButtonHandler handlerBut = new ButtonHandler ();
addTypeConditionToPolicyButton = new JButton ("Add to policy");
addTypeConditionToPolicyButton.addActionListener(handlerBut);
c.add(addTypeConditionToPolicyButton);

lineupLabel3 = new JLabel ("");
c.add(lineupLabel3);

conditionTypesAreLabel = new JLabel ("Condition types are: ");
c.add(conditionTypesAreLabel);

conditionTypeList = new JList ();
conditionTypeList.setBackground(Color.lightGray);
c.add ( new JScrollPane (conditionTypeList) );

lineupLabel4 = new JLabel ("");
c.add(lineupLabel4);

lineupLabel5 = new JLabel ("");
c.add(lineupLabel5);

okButton = new JButton (" OK ");
okButton.addActionListener(handlerBut);
c.add(okButton);

lineupLabel6 = new JLabel ("");
c.add(lineupLabel6);

cancelButton = new JButton ("Cancel");
cancelButton.addActionListener(handlerBut);
c.add(cancelButton);

```

```

} // End of createTypeConditionGuiBuilderMethod

/**
 * Class      : CheckBoxHandler
 * Purpose    : Inner class for combo box event handling
 */

private class CheckBoxHandler implements ItemListener {

    public void itemStateChanged( ItemEvent e ) {

        if (e.getSource() == typesComboBox) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                oneTypeObject.setConditionTypeName ( ( (Type)
typesComboBox.getSelectedItemAt() ).getName() );
                typeMembersComboBox.removeAllItems(); // Before adding
the members of the newly selected type, I delete the members of the old
selected type from the type members combo box

                Enumeration enum = ( ( (Type)
typesComboBox.getSelectedItemAt() ).getTypeMembersVector() ).elements();
// the type member vector of the type that is selected from the type
combo box is assigned to an enum so that the elements of this member
vector will be used to set the objects in type member combo box
                while ( enum.hasMoreElements() ) {
                    typeMembersComboBox.addItem ( enum.nextElement() );
// Add the members of the selected type from type combo box to the type
members combo box
                } // End of while
            } // End of if ( e.getStateChange()...)
        } // End of typesComboBox if

        if (e.getSource() == typeMembersComboBox) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                oneTypeObject.setConditionTypeMemberName( (String)
typeMembersComboBox.getSelectedItemAt() );
            }
        } // End of typeMembersComboBox if

        if (e.getSource() == relationComboBox) {
            if ( e.getStateChange() == ItemEvent.SELECTED ) {
                if ( ( (String) relationComboBox.getSelectedItemAt()
).equals("") ) {
                    oneTypeObject.setIsEqual(true);
                    oneTypeObject.setIsNotEqual(false);
                    oneTypeObject.setIsGreaterThanOrEqual(false);
                    oneTypeObject.setIsSmallerThanOrEqual(false);
                } // End of if

                if ( ( (String) relationComboBox.getSelectedItemAt()
).equals("!=") ) {
                    oneTypeObject.setIsEqual(false);

```

```

        oneTypeObject.setIsNotEqual(true);
        oneTypeObject.setIsGreaterThanOrEqual(false);
        oneTypeObject.setIsSmallerThanOrEqual(false);
    } // End of if

    if ( ( (String) relationComboBox.getSelectedItem()
).equals(">=") ) {
        oneTypeObject.setIsEqual(false);
        oneTypeObject.setIsNotEqual(false);
        oneTypeObject.setIsGreaterThanOrEqual(true);
        oneTypeObject.setIsSmallerThanOrEqual(false);
    } // End of if

    if ( ( (String) relationComboBox.getSelectedItem()
).equals("<=") ) {
        oneTypeObject.setIsEqual(false);
        oneTypeObject.setIsNotEqual(false);
        oneTypeObject.setIsGreaterThanOrEqual(false);
        oneTypeObject.setIsSmallerThanOrEqual(true);
    } // End of if

    } // End of if ( e.getStateChange() == ItemEvent.SELECTED )

    } // End of relationComboBox if

    } // End of itemStateChanged method

} // End of class CheckBoxHandler

```

```

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for Add type condition to policy, OK
 * and Cancel button event handling. When the user clicks add type
 * condition to policy button, one type object (which was set
 * according to the combo box choices) is placed in the
 * tempVectorToSetTypeConditionVector vector. When the user clicks
 * ok button, current temporary vector which has the type conditions
 * in it will set the type condition vector of the condition class
 * object or add its elements to it. When the user clicks on the
 * cancel button, type condition creation window will be dismissed.
 */

```

```

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == addTypeConditionToPolicyButton ) {

```

/* The reason why I create a new OneType object is that each time I add a new type to the vector, I do not want the new type to make the old ones in the vector the same as itself (new type). I use this new OneType object to put the copy (clone) of the OneType object that comes from the GUI in it. And then I add this new OneType object to the vector. So, each new OneType object that comes from the GUI does not affect the ones already in the vector. In other words, they are not referencing to the same OneType object anymore. */

```

        OneType OneTypeDifferentObject = new OneType ();
        OneTypeDifferentObject.copy(oneTypeObject);

        if ( !(
tempVectorToSetTypeConditionVector.contains(OneTypeDifferentObject) )
&&
                !(
(policyConditionForType.getTypeConditionVector()).contains(OneTypeDiffe
rentObject) ) ) {

tempVectorToSetTypeConditionVector.addElement(OneTypeDifferentObject);
        conditionTypeList.setListData(
tempVectorToSetTypeConditionVector );
        }
        else {
                JOptionPane.showMessageDialog (CreateTypeCondition.this,
"Create a different type condition", "Different type condition needed",
JOptionPane.ERROR_MESSAGE);
        }

        } // End of if ( e.getSource() ==
addTypeConditionToPolicyButton )

        if ( e.getSource() == okButton ) {
                if ( tempVectorToSetTypeConditionVector.isEmpty() ) {
                        JOptionPane.showMessageDialog (CreateTypeCondition.this,
"Please create a condition type", "Condition type creation needed",
JOptionPane.ERROR_MESSAGE);
                }
                else {
                        if ( ( policyConditionForType.getTypeConditionVector()
).isEmpty() ) {

policyConditionForType.setTypeConditionVector(tempVectorToSetTypeCondit
ionVector);
                                }
                                else { // If the type condition vector of the condition
class object is not empty, of course I do not want to set this vector
(since I will loose data in it). Instead, I will add new elements to
it.

                                        Enumeration enum =
tempVectorToSetTypeConditionVector.elements();
                                        while ( enum.hasMoreElements() ) {
                                                OneType tempOneType = new OneType ();
                                                tempOneType = ( OneType ) enum.nextElement();
                                                ( policyConditionForType.getTypeConditionVector()
).addElement(tempOneType);
                                        } // End of while
                                } // End of else (when the type condition vector of
conditon class object is not empty)

                                        CreateTypeCondition.this.dispose();

```

```
        } // End of else

    } // End of if ( e.getSource() == okButton )

    if ( e.getSource() == cancelButton ) {
        CreateTypeCondition.this.dispose();
    } // End of if ( e.getSource() == cancelButton )

    } // End of method actionPerformed

    } // End of class ButtonHandler

} // End of class CreateTypeCondition
```

```

/*****
File: CreateUserIDCondition.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/17/2001
Description: In this file, I will design the one of the 8 condition
types. By means of this window the user will be able to form policy
conditions by using userID.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class CreateUserIDCondition extends JDialog {
    private GuiMenu guiMenuForUserIDCondition;
    private Condition policyConditionForUserID;
    /* I will set the userIDCondition vector of condition class object
    (when the user clicks on the "UserID" button in the policy condition
    window) in this class. And other 7 vectors of the same object of the
    condition class will be set in 7 different classes and then when the
    user clicks the next button this condition class object will be put in
    the policy class object. */

    private Vector tempVectorToSetUserIDConditionVector = new Vector
(1);

    private OneUserID oneUserIDObject = new OneUserID ();

    private Vector relationVector = new Vector (1);

    private boolean hasWhiteSpace = false;

    private JLabel userIDLabel, conditionUserIDAreLabel;
    private JLabel lineupLabel, lineupLabel2, lineupLabel3,
lineupLabel4;
    private JComboBox relationComboBox;
    private JTextField userIDValueTextField;
    private JButton addUserIDConditionToPolicyButton, okButton,
cancelButton;
    private JList conditionUserIDList;

    /**
     * Method      : CreateUserIDCondition
     * Purpose     : Constructor for user ID condition creation
     * Parameters  : GuiMenu g, Condition c
     */

    public CreateUserIDCondition (GuiMenu g, Condition c ) {
        super ( g, "User ID conditions", true ); // Parent Frame is main
        GUI, and this JDialog is modal
        setLocation (115, 115);
        this.guiMenuForUserIDCondition = g;
        this.policyConditionForUserID = c;
        createUserIDConditionGuiBuilderMethod ();
        setResizable (false);
    }
}

```

```

        setSize(460, 350);
        show();
    } // End of constructor

/**
 * Method      : createUserIDConditionGuiBuilderMethod
 * Purpose     : This method builds the GUI for the creation of user
 * ID conditions
 * Parameters  : None
 */

private void createUserIDConditionGuiBuilderMethod () {
    Container c = getContentPane ();
    c.setLayout( new FlowLayout () );

    userIDLabel = new JLabel ("User ID ");
    c.add(userIDLabel);

    CheckBoxHandler handler = new CheckBoxHandler ();

    relationVector.addElement("==");
    relationVector.addElement("!=");

    relationComboBox = new JComboBox (relationVector);
    c.add(relationComboBox);
    relationComboBox.addItemListener(handler);

    oneUserIDObject.setIsEqual (true);
    oneUserIDObject.setIsNotEqual (false);

    userIDValueTextField = new JTextField (10);
    c.add(userIDValueTextField);

    ButtonHandler handlerBut = new ButtonHandler ();
    addUserIDConditionToPolicyButton = new JButton ("Add to policy");
    addUserIDConditionToPolicyButton.addActionListener(handlerBut);
    c.add(addUserIDConditionToPolicyButton);

    lineupLabel = new JLabel ("                ");
    c.add(lineupLabel);

    conditionUserIDAreLabel = new JLabel ("User ID conditions are:");
    c.add(conditionUserIDAreLabel);

    conditionUserIDList = new JList ();
    conditionUserIDList.setBackground(Color.lightGray);
    c.add ( new JScrollPane (conditionUserIDList) );

    lineupLabel2 = new JLabel ("                ");
    c.add(lineupLabel2);

    lineupLabel3 = new JLabel ("                ");
    c.add(lineupLabel3);

```



```

        okButton = new JButton ( "   OK   " );
        okButton.addActionListener(handlerBut);
        c.add(okButton);

        lineupLabel4 = new JLabel ( "                                " );
        c.add(lineupLabel4);

        cancelButton = new JButton ("Cancel");
        cancelButton.addActionListener(handlerBut);
        c.add(cancelButton);

    } // End of createUserIDConditionGuiBuilderMethod

    /**
    * Class          : CheckBoxHandler
    * Purpose        : Inner class for combo box event handling
    */

    private class CheckBoxHandler implements ItemListener {

        public void itemStateChanged( ItemEvent e ) {

            if ( e.getStateChange() == ItemEvent.SELECTED ) {

                if ( ( (String) relationComboBox.getSelectedItem()
).equals("==") ) {
                    oneUserIDObject.setIsEqual(true);
                    oneUserIDObject.setIsNotEqual(false);
                } // End of if

                if ( ( (String) relationComboBox.getSelectedItem()
).equals("!=") ) {
                    oneUserIDObject.setIsEqual(false);
                    oneUserIDObject.setIsNotEqual(true);
                } // End of if

            } // End of if ( e.getStateChange() == ItemEvent.SELECTED )

        } // End of itemStateChanged method

    } // End of class CheckBoxHandler

    /**
    * Class          : ButtonHandler
    * Purpose        : Inner class for Add user ID condition to policy, OK
    * and Cancel button event handling. When the user clicks add user
    * ID condition to policy button, oneUserIDObject (which was set
    * according to the combo box and text field) is placed in the
    * tempVectorToSetUserIDConditionVector vector. When the user clicks
    * ok button, current temporary vector which has the UserID
    * conditions in it will set the UserID condition vector of the
    * condition class object or add its elements to it. When the user
    * clicks on the cancel button, time condition creation window will
    * be dismissed.

```

```

*/

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == addUserIDConditionToPolicyButton ) {

            if ( (userIDValueTextField.getText() ).equals("") ) {
                JOptionPane.showMessageDialog
(CreateUserIDCondition.this, "Please provide a user ID.", "User ID
needed", JOptionPane.ERROR_MESSAGE);
            }
            else if ( whiteSpaceControl (
userIDValueTextField.getText() ) ) {
                JOptionPane.showMessageDialog
(CreateUserIDCondition.this, "White spaces are not allowed in user ID",
"White space warning", JOptionPane.ERROR_MESSAGE);
            }
            else{
                oneUserIDObject.setUserIDValue (
userIDValueTextField.getText() );
                OneUserID oneUserIDDifferentObject = new OneUserID ();
                oneUserIDDifferentObject.copy(oneUserIDObject);

                if ( !(
tempVectorToSetUserIDConditionVector.contains(oneUserIDDifferentObject)
) &&
                    !(
(policyConditionForUserID.getUserIDConditionVector()).contains(oneUserI
DDifferentObject) ) ) {

tempVectorToSetUserIDConditionVector.addElement(oneUserIDDifferentObjec
t);

                    conditionUserIDList.setListData(
tempVectorToSetUserIDConditionVector ); // Display the selected
OneUserID objects in the JList.
                    userIDValueTextField.setText(""); // Text field is
cleared for the ease of use for the next time
                }
                else {
                    JOptionPane.showMessageDialog
(CreateUserIDCondition.this, "Create a different user ID condition",
"Different user ID condition needed", JOptionPane.ERROR_MESSAGE);
                }

            } // End of else

        } // End of if ( e.getSource() ==
addTimeConditionToPolicyButton )

        if ( e.getSource() == okButton ) {
            if ( tempVectorToSetUserIDConditionVector.isEmpty() ) {
                JOptionPane.showMessageDialog
(CreateUserIDCondition.this, "Please create a User ID condition", "
User ID condition needed", JOptionPane.ERROR_MESSAGE);
            }
        }
    }
}

```

```

        else {
            if ( (
policyConditionForUserID.getUserIDConditionVector() ).isEmpty() ) {

policyConditionForUserID.setUserIDConditionVector(tempVectorToSetUserID
ConditionVector);
            }
            else { // If the UserID condition vector of the
condition class object is not empty, of course I do not want to set
this vector (since I will loose data in it). Instead, I will add new
elements to it.
                Enumeration enum =
tempVectorToSetUserIDConditionVector.elements();
                while ( enum.hasMoreElements() ) {
                    OneUserID tempOneUserID = new OneUserID ();
                    tempOneUserID = ( OneUserID ) enum.nextElement();
                    (
policyConditionForUserID.getUserIDConditionVector()
).addElement(tempOneUserID);
                } // End of while
            } // End of else (when the UserID condition vector of
conditon class object is not empty)

                CreateUserIDCondition.this.dispose();

            } // End of else

        } // End of if ( e.getSource() == okButton )

        if ( e.getSource() == cancelButton ) {
            CreateUserIDCondition.this.dispose();
        } // End of if ( e.getSource() == cancelButton )

    } // End of method actionPerformed

/**
 * Method      : whiteSpaceControl
 * Purpose     : This method checks the strings that the user
 * creates to put them in a format that PPL compiler accepts. User
 * ID should not contain white space characters. This method returns
 * true if a string has white space character, false otherwise.
 * Parameters  : String s
 */

private boolean whiteSpaceControl (String s) {
    hasWhiteSpace = false;
    char charArray [] = s.toCharArray();
    for ( int i = 0; i < charArray.length; ++i ) {
        if ( Character.isWhitespace ( charArray[i] ) ) {
            hasWhiteSpace = true;
            break;
        } // End of if
    } // End of for
    if ( hasWhiteSpace == true) {

```

```
        return true;
    }
    else {
        return false;
    }
} // End of whiteSpaceControl Method

} // End of class ButtonHandler

} // End of class CreateUserIDCondition
```

```

/*****
File: DeleteClass.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/24/2001
Description: By means of this class, the user will be able to delete a
class.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class DeleteClass extends JDialog {
    private DirectorClass boss;
    private Vector a; // A Vector reference for Class class constructor
    private Class networkClassToBeDeleted = new Class ( "", a );

    private JLabel deleteClassLabel, lineupLabel;
    private JComboBox classesComboBox;
    private JPanel deletePanel, buttonPanel;
    private JButton deleteButton, cancelButton;

    /**
     * Method      : DeleteClass
     * Purpose     : Constructor for DeleteClass class
     * Parameters  : GuiMenu gui, DirectorClass x
     */

    public DeleteClass ( GuiMenu gui, DirectorClass x ) { // GuiMenu gui
is only used to tell the JDialog who its parent component is (GuiMenu
instance in this case) for the modality purpose
        super ( gui, "Delete Class", true ); // parent Frame is main GUI,
and this JDialog is modal
        /* Following line of code Moves this component to a new location.
The top-left corner of the new location is specified by the x and y
parameters in the coordinate space of this component's parent. */
        setLocation (115, 115);
        this.boss = x;
        deleteClassGuiBuilderMethod (); // A method call for building the
GUI for delete class JDialog
        setResizable (false);
        setSize(455,370);
        show();
    } // End of constructor for DeleteClass class

    /**
     * Method      : deleteClassGuiBuilderMethod

```

```

* Purpose      : This method places the components of delete class
* JDialog.
* Parameters   : None
*/

private void deleteClassGuiBuilderMethod () {
    // BorderLayout is default for content panes of JFrames.
    FlowLayout is default for JPanel.
    Container c = getContentPane ();

    deletePanel = new JPanel ();

    deleteClassLabel = new JLabel ( "                Select the class
you want to delete:                ");
    deletePanel.add ( deleteClassLabel );

    classesComboBox = new JComboBox ( (boss.getClassVector()) );
    classesComboBox.setMaximumRowCount(7);
    deletePanel.add ( new JScrollPane (classesComboBox) ); // Provide
a scroll bar if there are more than 7 classes in the combo box.
    Dimension d = new Dimension (185, 25); // dimension object
(width, height) to be used as the dimension of class combo box
    classesComboBox.setPreferredSize(d); // set the size of the class
combo box so that it will not be too big or too small
    deletePanel.add ( classesComboBox );

    ButtonHandler handlerBut = new ButtonHandler ();

    c.add (deletePanel, BorderLayout.CENTER);

    buttonPanel = new JPanel ();

    deleteButton = new JButton ("Delete");
    buttonPanel.add(deleteButton);
    deleteButton.addActionListener(handlerBut);
    if ( ( boss.getClassVector() ).isEmpty() ) {
        deleteButton.setEnabled(false);
        deleteButton.setToolTipText("No class is defined for this
network yet.");
    }

    lineupLabel = new JLabel ( "                ");
    buttonPanel.add(lineupLabel);

    cancelButton = new JButton ("Cancel");
    buttonPanel.add(cancelButton);
    cancelButton.addActionListener(handlerBut);

    c.add ( buttonPanel, BorderLayout.SOUTH );
} // End of deleteClassGuiBuilderMethod

/**
* Class      : ButtonHandler

```

```

* Purpose      : Inner class for event handling of OK and Cancel
* buttons. The purpose is to delete a class and then destroy the
* deletion window.
*/

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == deleteButton ) {
            networkClassToBeDeleted = ( Class )
classesComboBox.getSelectedItemAt(); // selected class is assigned to a
Class object for deletion purpose
            boss.removeClass ( networkClassToBeDeleted ); //
removeClass method of DirectorClass is called
            DeleteClass.this.dispose();
        } // End of if

        if ( e.getSource() == cancelButton ) {
            DeleteClass.this.dispose();
        } // End of if

    } // End of method actionPerformed

} // End of class ButtonHandler

} // End of class DeleteClass

```

```

/*****
File: DeleteLink.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/24/2001
Description: By means of this class, the user will be able to delete a
link.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class DeleteLink extends JDialog {

    /* Director class referencing: myDirector (in GuiMenu class constr.) =
    pplDirector (in DirectorClass class), owner ( as a GuiMenu class data
    member) = myDirector, x (in Create/DeleteNode class constr.) = owner,
    boss ( as a Create/DeleteNode class data member) = x. */

    private DirectorClass boss;
    private Node a; // A node reference for Link class constructor
    private Node b; // A node reference for Link class constructor
    private Link networkLinkToBeDeleted = new Link ( "", a, b, 0, false,
false, false, false, false );

    private JLabel deleteLinkLabel, lineupLabel;
    private JComboBox linksComboBox;
    private JPanel deletePanel, buttonPanel;
    private JButton deleteButton, cancelButton;

    /**
     * Method      : DeleteLink
     * Purpose      : Constructor for DeleteLink class
     * Parameters   : GuiMenu gui, DirectorClass x
     */

    public DeleteLink ( GuiMenu gui, DirectorClass x ) { // GuiMenu gui
is only used to tell the JDialog who its parent component is (GuiMenu
instance in this case) for the modality purpose
        super ( gui, "Delete Link", true ); // parent Frame is main GUI,
and this JDialog is modal
        /* Following line of code Moves this component to a new location.
The top-left corner of the new location is specified by the x and y
parameters in the coordinate space of this component's parent. */

        setLocation (115, 115);
        this.boss = x;
        deleteLinkGuiBuilderMethod (); // A method call for building the
GUI for delete link JDialog
        setResizable (false);
        setSize(455,370);
        show();
    } // End of constructor for DeleteLink class
    /**
     * Method      : deleteLinkGuiBuilderMethod

```



```

* Purpose      : This method places the components of delete link
* JDialog.
* Parameters   : None
*/

private void deleteLinkGuiBuilderMethod () {
    // BorderLayout is default for content panes of JFrames.
    FlowLayout is default for JPanel.
    Container c = getContentPane ();

    deletePanel = new JPanel ();

    deleteLinkLabel = new JLabel ( "                Select the link
you want to delete:                ");
    deletePanel.add ( deleteLinkLabel );

    linksComboBox = new JComboBox ( (boss.getLinkVector()) );
    linksComboBox.setMaximumRowCount(7);
    deletePanel.add ( new JScrollPane (linksComboBox) ); // Provide a
scroll bar if there are more than 7 links in the combo box.
    Dimension d = new Dimension (185, 25); // dimension object
(width, height) to be used as the dimension of link combo box
    linksComboBox.setPreferredSize(d); // set the size of the link
combo box so that it will not be too big or too small
    deletePanel.add ( linksComboBox );

    ButtonHandler handlerBut = new ButtonHandler ();

    c.add (deletePanel, BorderLayout.CENTER);

    buttonPanel = new JPanel ();

    deleteButton = new JButton ("Delete");
    buttonPanel.add(deleteButton);
    deleteButton.addActionListener(handlerBut);
    if ( ( boss.getLinkVector() ).isEmpty() ) {
        deleteButton.setEnabled(false);
        deleteButton.setToolTipText("No link is defined for this
network yet.");
    }

    lineupLabel = new JLabel ( "                ");
    buttonPanel.add(lineupLabel);

    cancelButton = new JButton ("Cancel");
    buttonPanel.add(cancelButton);
    cancelButton.addActionListener(handlerBut);

    c.add ( buttonPanel, BorderLayout.SOUTH );
} // End of deleteLinkGuiBuilderMethod

/**
* Class      : ButtonHandler

```

```

* Purpose      : Inner class for event handling of OK and Cancel
* buttons. The purpose is to delete a link and then destroy the
* deletion window.
*/

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == deleteButton ) {
            networkLinkToBeDeleted = ( Link )
linksComboBox.getSelectedItemAt(); // selected link is assigned to a Link
object for deletion purpose
            boss.removeLink ( networkLinkToBeDeleted ); // removeLink
method of DirectorClass is called
            DeleteLink.this.dispose();
        } // End of if

        if ( e.getSource() == cancelButton ) {
            DeleteLink.this.dispose();
        } // End of if

    } // End of method actionPerformed

} // End of class ButtonHandler

} // End of DeleteLink class

```

```

/*****
File: DeleteNode.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/12/2001
Description: By means of this class, the user will be able to delete a
node .
*****/

```

```

import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

```

```

public class DeleteNode extends JDialog {
    private DirectorClass boss;
    private Node networkNodeToBeDeleted = new Node ();

    private JLabel deleteNodeLabel, lineupLabel;
    private JComboBox nodesComboBox;
    private JPanel deletePanel, buttonPanel;
    private JButton okButton, cancelButton;

    /**
     * Method      : DeleteNode
     * Purpose     : Constructor for DeleteNode class
     * Parameters  : GuiMenu gui, DirectorClass x
     */

    public DeleteNode ( GuiMenu gui, DirectorClass x ) {
        super ( gui, "Delete Node", true ); // parent Frame is main GUI,
        and this JDialog is modal
        /* Following line of code Moves this component to a new location.
        The top-left corner of the new location is specified by the x and y
        parameters in the coordinate space of this component's parent. */
        setLocation (115, 115);
        this.boss = x;
        deleteNodeGuiBuilderMethod (); // A method call for building the
        GUI for delete node JDialog
        setResizable (false);
        setSize(455,370);
        show();
    } // End of constructor for DeleteNode class

```

```

/**
 * Method      : deleteNodeGuiBuilderMethod

```

```

* Purpose      : This method places the components of delete node
* JDialog.
* Parameters   : None
*/

private void deleteNodeGuiBuilderMethod () {
    // BorderLayout is default for content panes of JFrames.
    FlowLayout is default for JPanel.
    Container c = getContentPane ();

    deletePanel = new JPanel ();

    deleteNodeLabel = new JLabel ( "Select the node you want to
delete (Node Name_Domain Name):");
    deletePanel.add ( deleteNodeLabel );

    nodesComboBox = new JComboBox ( (boss.getNodeVector()) );
    nodesComboBox.setMaximumRowCount(7);
    deletePanel.add ( new JScrollPane (nodesComboBox) ); // Provide a
scroll bar if there are more than 7 nodes in the combo box.
    Dimension d = new Dimension (175, 25); // dimension object
(width, height) to be used as the dimension of node combo box
    nodesComboBox.setPreferredSize(d); // set the size of the node
combo box so that it will not be too big or too small
    deletePanel.add ( nodesComboBox );

    ButtonHandler handlerBut = new ButtonHandler ();

    c.add (deletePanel, BorderLayout.CENTER);

    buttonPanel = new JPanel ();

    okButton = new JButton ("Delete");
    buttonPanel.add(okButton);
    okButton.addActionListener(handlerBut);
    if ( ( boss.getNodeVector() ).isEmpty() ) {
        okButton.setEnabled(false);
        okButton.setToolTipText("No node is defined for this network
yet.");
    }

    lineupLabel = new JLabel ("");
    buttonPanel.add(lineupLabel);

    cancelButton = new JButton ("Cancel");
    buttonPanel.add(cancelButton);
    cancelButton.addActionListener(handlerBut);

    c.add ( buttonPanel, BorderLayout.SOUTH );
} // End of deleteNodeGuiBuilderMethod

/**
* Class      : ButtonHandler

```

```

* Purpose      : Inner class for event handling of OK and Cancel
* buttons. The purpose is to delete a node and then destroy the
* deletion window.
*/

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == okButton ) {
            networkNodeToBeDeleted = ( Node )
nodesComboBox.getSelectedItem(); // selected node is assigned to a Node
object for deletion purpose
            boss.removeNode ( networkNodeToBeDeleted ); // removeNode
method of DirectorClass is called
            DeleteNode.this.dispose();
        } // End of if

        if ( e.getSource() == cancelButton ) {
            DeleteNode.this.dispose();
        } // End of if

    } // End of method actionPerformed

} // End of class ButtonHandler

} // End of class DeleteNode

```

```

/*****
File: DeletePath.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/24/2001
Description: By means of this class, the user will be able to delete a
path.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class DeletePath extends JDialog {
    private DirectorClass boss;
    private Vector a; // A Vector reference for Path class constructor
    private Path networkPathToBeDeleted = new Path ("", a, 0, false,
false, false, false, false);

    private JLabel deletePathLabel, lineupLabel;
    private JComboBox pathsComboBox;
    private JPanel deletePanel, buttonPanel;
    private JButton deleteButton, cancelButton;

    /**
     * Method      : DeletePath
     * Purpose     : Constructor for DeletePath class
     * Parameters  : GuiMenu gui, DirectorClass x
     */

    public DeletePath ( GuiMenu gui, DirectorClass x ) { // GuiMenu gui
is only used to tell the JDialog who its parent component is (GuiMenu
instance in this case) for the modality purpose
        super ( gui, "Delete Path", true ); // parent Frame is main GUI,
and this JDialog is modal
        /* Following line of code Moves this component to a new location.
The top-left corner of the new location is specified by the x and y
parameters in the coordinate space of this component's parent. */
        setLocation (115, 115);
        this.boss = x;
        deletePathGuiBuilderMethod (); // A method call for building the
GUI for delete path JDialog
        setResizable (false);
        setSize(455,370);
        show();
    } // End of constructor for DeletePath class

    /**
     * Method      : deletePathGuiBuilderMethod

```

```

* Purpose      : This method places the components of delete path
* JDialog.
* Parameters   : None
*/

private void deletePathGuiBuilderMethod () {
    // BorderLayout is default for content panes of JFrames.
    FlowLayout is default for JPanel.
    Container c = getContentPane ();

    deletePanel = new JPanel ();

    deletePathLabel = new JLabel ( "                Select the path
you want to delete:                ");
    deletePanel.add ( deletePathLabel );

    pathsComboBox = new JComboBox ( (boss.getPathVector()) );
    pathsComboBox.setMaximumRowCount(7);
    deletePanel.add ( new JScrollPane (pathsComboBox) ); // Provide a
scroll bar if there are more than 7 paths in the combo box.
    Dimension d = new Dimension (185, 25); // dimension object
(width, height) to be used as the dimension of path combo box
    pathsComboBox.setPreferredSize(d); // set the size of the path
combo box so that it will not be too big or too small
    deletePanel.add ( pathsComboBox );

    ButtonHandler handlerBut = new ButtonHandler ();

    c.add (deletePanel, BorderLayout.CENTER);

    buttonPanel = new JPanel ();

    deleteButton = new JButton ("Delete");
    buttonPanel.add(deleteButton);
    deleteButton.addActionListener(handlerBut);
    if ( ( boss.getPathVector() ).isEmpty() ) {
        deleteButton.setEnabled(false);
        deleteButton.setToolTipText("No path is defined for this
network yet.");
    }

    lineupLabel = new JLabel ( "                ");
    buttonPanel.add(lineupLabel);

    cancelButton = new JButton ("Cancel");
    buttonPanel.add(cancelButton);
    cancelButton.addActionListener(handlerBut);

    c.add ( buttonPanel, BorderLayout.SOUTH );
} // End of deletePathGuiBuilderMethod

/**
* Class      : ButtonHandler

```

```

* Purpose      : Inner class for event handling of OK and Cancel
* buttons. The purpose is to delete a path and then destroy the
* deletion window.
*/

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == deleteButton ) {
            networkPathToBeDeleted = ( Path )
pathsComboBox.getSelectedItemAt(); // selected path is assigned to a Path
object for deletion purpose
            boss.removePath ( networkPathToBeDeleted ); // removePath
method of DirectorClass is called
            DeletePath.this.dispose();
        } // End of if

        if ( e.getSource() == cancelButton ) {
            DeletePath.this.dispose();
        } // End of if

    } // End of method actionPerformed

} // End of class ButtonHandler

} // End of class DeletePath

```



```

/*****
File: DeletePolicy.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/20/2001
Description: By means of this class, the policy maker will be able to
delete a policy.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class DeletePolicy extends JDialog {
    private DirectorClass boss;
    private Policy networkPolicyToBeDeleted = new Policy ();

    private JLabel deletePolicyLabel, lineupLabel;
    private JComboBox policiesComboBox;
    private JPanel deletePanel, buttonPanel;
    private JButton okButton, cancelButton;

    /**
     * Method      : DeletePolicy
     * Purpose      : Constructor for DeletePolicy class
     * Parameters   : GuiMenu gui, DirectorClass x
     */

    public DeletePolicy ( GuiMenu gui, DirectorClass x ) {
        super ( gui, "Delete Policy", true ); // parent Frame is main
        GUI, and this JDialog is modal
        setLocation (115, 115);
        this.boss = x;
        deletePolicyGuiBuilderMethod (); // A method call for building
the GUI for delete policy JDialog
        setResizable (false);
        setSize(455,370);
        show();
    } // End of constructor

    /**
     * Method      : deletePolicyGuiBuilderMethod
     * Purpose      : This method places the components of delete policy
     * window.
     * Parameters   : None
     */

    private void deletePolicyGuiBuilderMethod () {
        // BorderLayout is default for content panes of JFrames.
        FlowLayout is default for JPanel.
        Container c = getContentPane ();

        deletePanel = new JPanel ();

```

```

        deletePolicyLabel = new JLabel ( "                Select the policy
you want to delete:                ");
        deletePanel.add ( deletePolicyLabel );

        policiesComboBox = new JComboBox ( ( boss.getPolicyVector() ) );
        policiesComboBox.setMaximumRowCount(7);
        deletePanel.add ( new JScrollPane (policiesComboBox) ); //
Provide a scroll bar if there are more than 7 policies in the combo
box.
        Dimension d = new Dimension (175, 25); // dimension object
(width, height) to be used as the dimension of policy combo box
        policiesComboBox.setPreferredSize(d); // set the size of the node
policy box so that it will not be too big or too small
        deletePanel.add ( policiesComboBox );

        ButtonHandler handlerBut = new ButtonHandler ();

        c.add (deletePanel, BorderLayout.CENTER);

        buttonPanel = new JPanel ();

        okButton = new JButton ("Delete");
        buttonPanel.add(okButton);
        okButton.addActionListener(handlerBut);
        if ( ( boss.getPolicyVector() ).isEmpty() ) {
            okButton.setEnabled(false);
            okButton.setToolTipText("No policy is defined for this network
yet.");
        }

        lineupLabel = new JLabel ( "                ");
        buttonPanel.add(lineupLabel);

        cancelButton = new JButton ("Cancel");
        buttonPanel.add(cancelButton);
        cancelButton.addActionListener(handlerBut);

        c.add ( buttonPanel, BorderLayout.SOUTH );

    } // End of deletePolicyGuiBuilderMethod

/**
 * Class          : ButtonHandler
 * Purpose        : Inner class for event handling of OK and Cancel
 * buttons. The purpose is to delete a policy and then destroy the
 * deletion window.
 */

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == okButton ) {
            networkPolicyToBeDeleted = ( Policy )
policiesComboBox.getSelectedItem(); // selected policy is assigned to a
Policy object for deletion purpose

```

```

        boss.removePolicy ( networkPolicyToBeDeleted ); //
removePolicy method of DirectorClass is called
        DeletePolicy.this.dispose();
    } // End of if

    if ( e.getSource() == cancelButton ) {
        DeletePolicy.this.dispose();
    } // End of if

    } // End of method actionPerformed

    } // End of class ButtonHandler

} // End of class DeletePolicy

```

```

/*****
File: DeletePolicyMaker.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/27/2001
Description: By means of this class, administrator will be able to
delete a policy maker.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class DeletePolicyMaker extends JDialog {

    private DirectorClass boss;
    private PolicyMaker networkPolicyMakerToBeDeleted = new PolicyMaker
(" ", " ", 1);

    private JLabel deletePolicyMakerLabel, lineupLabel;
    private JComboBox policyMakersComboBox;
    private JPanel deletePanel, buttonPanel;
    private JButton deleteButton, cancelButton;

    /**
     * Method      : DeletePolicyMaker
     * Purpose     : Constructor for DeletePolicyMaker class
     * Parameters  : GuiMenu gui, DirectorClass x
     */

    public DeletePolicyMaker ( GuiMenu gui, DirectorClass x ) {
        super ( gui, "Delete User", true ); // parent Frame is main GUI,
and this JDialog is modal
        setLocation (115, 115);
        this.boss = x;
        deletePolicyMakerGuiBuilderMethod ();
        setResizable (false);
        setSize(455,370);
        show();
    } // End of constructor

    /**
     * Method      : deletePolicyMakerGuiBuilderMethod
     * Purpose     : This method places the components of delete policy
     * maker JDialog.
     * Parameters  : None
     */

    private void deletePolicyMakerGuiBuilderMethod () {
        Container c = getContentPane ();

        deletePanel = new JPanel ();

        deletePolicyMakerLabel = new JLabel ( "                Select the
user you want to delete:                ");

```

```

        deletePanel.add ( deletePolicyMakerLabel );

        policyMakersComboBox = new JComboBox ( (
boss.getPolicyMakerVector() ) );
        policyMakersComboBox.setMaximumRowCount(7);
        deletePanel.add ( new JScrollPane (policyMakersComboBox) );
        Dimension d = new Dimension (185, 25);
        policyMakersComboBox.setPreferredSize(d);
        deletePanel.add ( policyMakersComboBox );

        ButtonHandler handlerBut = new ButtonHandler ();

        c.add (deletePanel, BorderLayout.CENTER);

        buttonPanel = new JPanel ();

        deleteButton = new JButton ("Delete");
        buttonPanel.add(deleteButton);
        deleteButton.addActionListener(handlerBut);

        lineupLabel = new JLabel ("");
        buttonPanel.add(lineupLabel);

        cancelButton = new JButton ("Cancel");
        buttonPanel.add(cancelButton);
        cancelButton.addActionListener(handlerBut);

        c.add ( buttonPanel, BorderLayout.SOUTH );

    } // End of deleteTypeGuiBuilderMethod

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for event handling of OK and Cancel
 * buttons. The purpose is to delete a policy maker and then destroy
 * the deletion window.
 */

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == deleteButton ) {
            networkPolicyMakerToBeDeleted = ( PolicyMaker )
policyMakersComboBox.getSelectedItem();
            if ( ( networkPolicyMakerToBeDeleted.getLoginID()
).equalsIgnoreCase("administrator") ) {
                JOptionPane.showMessageDialog (DeletePolicyMaker.this,
"Administrator account cannot be deleted.", "Deletion error",
JOptionPane.ERROR_MESSAGE);
            }
            else {
                boss.removePolicyMaker ( networkPolicyMakerToBeDeleted
);
                DeletePolicyMaker.this.dispose();
            }
        } // End of if
    }
}

```

```
        if ( e.getSource() == cancelButton ) {
            DeletePolicyMaker.this.dispose();
        } // End of if

    } // End of method actionPerformed

} // End of class ButtonHandler

} // End of class DeletePolicyMaker
```

```

/*****
File: DeleteType.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/24/2001
Description: By means of this class, the user will be able to delete a
type.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class DeleteType extends JDialog {
    private DirectorClass boss;
    private Vector a; // A Vector reference for Type class constructor
    private Type networkTypeToBeDeleted = new Type ( "", a );

    private JLabel deleteTypeLabel, lineupLabel;
    private JComboBox typesComboBox;
    private JPanel deletePanel, buttonPanel;
    private JButton deleteButton, cancelButton;

    /**
     * Method      : DeleteType
     * Purpose     : Constructor for DeleteType class
     * Parameters  : GuiMenu gui, DirectorClass x
     */

    public DeleteType ( GuiMenu gui, DirectorClass x ) { // GuiMenu gui
is only used to tell the JDialog who its parent component is (GuiMenu
instance in this case) for the modality purpose
        super ( gui, "Delete Type", true ); // parent Frame is main GUI,
and this JDialog is modal
        /* Following line of code Moves this component to a new location.
The top-left corner of the new location is specified by the x and y
parameters in the coordinate space of this component's parent. */
        setLocation (115, 115);
        this.boss = x;
        deleteTypeGuiBuilderMethod (); // A method call for building the
GUI for delete type JDialog
        setResizable (false);
        setSize(455,370);
        show();
    } // End of constructor for DeleteType class

    /**
     * Method      : deleteTypeGuiBuilderMethod

```

```

* Purpose      : This method places the components of delete type
* JDialog.
* Parameters   : None
*/

private void deleteTypeGuiBuilderMethod () {
    // BorderLayout is default for content panes of JFrames.
    FlowLayout is default for JPanel.
    Container c = getContentPane ();

    deletePanel = new JPanel ();

    deleteTypeLabel = new JLabel ( "                Select the type
you want to delete:                ");
    deletePanel.add ( deleteTypeLabel );

    typesComboBox = new JComboBox ( (boss.getTypeVector()) );
    typesComboBox.setMaximumRowCount(7);
    deletePanel.add ( new JScrollPane (typesComboBox) ); // Provide a
scroll bar if there are more than 7 types in the combo box.
    Dimension d = new Dimension (185, 25); // dimension object
(width, height) to be used as the dimension of class combo box
    typesComboBox.setPreferredSize(d); // set the size of the type
combo box so that it will not be too big or too small
    deletePanel.add ( typesComboBox );

    ButtonHandler handlerBut = new ButtonHandler ();

    c.add (deletePanel, BorderLayout.CENTER);

    buttonPanel = new JPanel ();

    deleteButton = new JButton ("Delete");
    buttonPanel.add(deleteButton);
    deleteButton.addActionListener(handlerBut);
    if ( ( boss.getTypeVector() ).isEmpty() ) {
        deleteButton.setEnabled(false);
        deleteButton.setToolTipText("No type is defined for this
network yet.");
    }

    lineupLabel = new JLabel ( "                ");
    buttonPanel.add(lineupLabel);

    cancelButton = new JButton ("Cancel");
    buttonPanel.add(cancelButton);
    cancelButton.addActionListener(handlerBut);

    c.add ( buttonPanel, BorderLayout.SOUTH );
} // End of deleteTypeGuiBuilderMethod

/**
* Class      : ButtonHandler

```



```

* Purpose      : Inner class for event handling of OK and Cancel
* buttons. The purpose is to delete a type and then destroy the
* deletion window.
*/

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == deleteButton ) {
            networkTypeToBeDeleted = ( Type )
typesComboBox.getSelectedItem(); // selected type is assigned to a Type
object for deletion purpose
            boss.removeType ( networkTypeToBeDeleted ); // removeType
method of DirectorClass is called
            DeleteType.this.dispose();
        } // End of if

        if ( e.getSource() == cancelButton ) {
            DeleteType.this.dispose();
        } // End of if

    } // End of method actionPerformed

} // End of class ButtonHandler

} // End of class DeleteClass

```

```

/*****
File: DirectorClass.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/18/2001
Description: In this file, vectors containing network elements and
policies will be maintained and handled. Users.txt file contains the
login id and password of the users. Temporary.txt is a temporary file
used for compiling. Cygwin.bat is batch file for invoking BASH shell
under Windows OS. The path of these three files have to be written
correctly before the program is run.
*****/

```

```

import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;
import java.io.*;

public class DirectorClass {
    private GuiMenu myGui;
    private String policyMakerID;

    private Vector nodeVector;
    private boolean redefinedNode = false;

    private Vector linkVector;
    private boolean redefinedLink = false;

    private Vector pathVector;
    private boolean redefinedPath = false;

    private Vector classVector;
    private boolean redefinedClass = false;

    private Vector typeVector;
    private boolean redefinedType = false;

    private Vector policyVector;
    private boolean redefinedPolicy = false;

    private PasswordGui myPasswordGui;

    private Vector policyMakerVector;
    private boolean redefinedPolicyMaker = false;

    // For policy makers output and input
    private ObjectOutputStream output;
    private ObjectInputStream input;

    private ObjectOutputStream outputForNetwork;
    private File networkFile;
    private PrintStream pStream; // This PrintStream object pStream will
    compose the file to be compiled

    private ObjectInputStream inputForNetwork;

```

```

    private ObjectOutputStream outputForPolicy;
    private File policyFile; // This is a handle to the file to which
policies will be written
    private PrintStream poStream;

    private ObjectInputStream inputForPolicy;

    /* Instead of defining it in both savePolicyMakersToFileMethod and
readPolicyMakersFromFileMethod definition of the userFile is placed
here. Thus only one time change will be enough. */
    private File userFile = new File
("C:\\WINDOWS\\jbproject\\THESIS\\users.txt");

    private boolean parseWorked = false;
    private boolean policyScanned = false;

    private File temporaryFileToBeCompiled = new File
("C:\\MyDocuments\\GaryStoneZipDiskCopy\\Compiler\\temporary.txt");
    private PrintStream coStream; // This PrintStream will be used to
create the temporary file to be compiled

    private int compileModeNumber;
    private Vector compilerOutputVector = new Vector (1);

/* The following boolean value is used to detect if the user clicked
cancel option in the JFileChooser when saving a file if he selects
"Yes" option when he clicks on "New" menu item. ("New" menu item - Yes
option - Cancel option) */
    public boolean cancelOptionSelectedInSaveFile = false;

/**
 * Method      : DirectorClass
 * Purpose     : Constructor for DirectorClass class
 * Parameters  : None
 */

DirectorClass () {
    nodeVector = new Vector (1);
    linkVector = new Vector (1);
    pathVector = new Vector (1);
    classVector = new Vector (1);
    typeVector = new Vector (1);
    policyVector = new Vector (1);

    policyMakerVector = new Vector (1);
    PolicyMaker administratorPolicyMaker = new PolicyMaker
("administrator", "administrator", 1);
    addPolicyMaker ( administratorPolicyMaker );

    readPolicyMakersFromFileMethod ();

    myPasswordGui = new PasswordGui (this);

```

```

} // End of constructor

/**
 * Method      : compileFileMethod
 * Purpose     : This method is used for compiling a PPL file.
 * It invokes BASH shell and executes "./scan fileName.txt" command
 * Parameters  : int compileModeNo
 */

public void compileFileMethod (int compileModeNo) {
    compileModeNumber = compileModeNo;
    parseWorked = false;
    policyScanned = false;

    createFileToBeCompiledMethod ();

    Process p;
    try {
        p =
Runtime.getRuntime().exec("C:\\MyDocuments\\GaryStoneZipDiskCopy\\CYGWIN.BAT");
    }
    catch( java.io.IOException e ) {
        System.err.println(e);
        return;
    }

    OutputStuffThread th = new OutputStuffThread (p);
    th.start();

    BufferedInputStream buffer = new
BufferedInputStream(p.getInputStream());
    BufferedReader commandResult = new BufferedReader(new
InputStreamReader(buffer));

    BufferedInputStream errorBuffer = new
BufferedInputStream(p.getErrorStream());
    BufferedReader errorResult = new BufferedReader(new
InputStreamReader(errorBuffer));

    try {
        for ( int i = 0; i <= 2; i++ ) {
            String s = commandResult.readLine();

            if ( s.equalsIgnoreCase ("Parse failed") )    {
                compilerOutputVector.addElement("Parse failed");
                break;
            }
            if ( s.equalsIgnoreCase ("Parse worked") )    {
                parseWorked = true;
            }
            if ( s.endsWith ("scanned") ) {
                policyScanned = true;
            }
        }
    }
}

```

```

        }
        compilerOutputVector.addElement(s);

    } // End of for

    if ( parseWorked && policyScanned ) {
        for ( int i = 0; i <= 20; i++ ) {
            String str = errorResult.readLine();
            compilerOutputVector.addElement(str);

            } // End of for
            compilerOutputVector.addElement(" Policy conflict
results written to scan_out.txt");
        } // End of if

        commandResult.close();
        errorResult.close();

    } // End of try
    catch (java.io.IOException e) {
        JOptionPane.showMessageDialog(null, "Error ocured when trying
to read from the output of BASH", "Error", JOptionPane.ERROR_MESSAGE);
    } // End of catch

} // End of method compileFileMethod

/**
 * Class      : OutputStuffThread
 * Purpose    : This class creates a second thread which is used to
 * instantiate BASH shell in which the PPL compiler will work
 */

class OutputStuffThread extends Thread {
    private Process m_process;
    // Constructor for OutputStuffThread class
    public OutputStuffThread( Process process) {
        m_process = process;
    } // End of constructor

    public void run() {

        try {
            BufferedOutputStream bufferout = new
BufferedOutputStream(m_process.getOutputStream());
            PrintWriter commandInput = new PrintWriter((new
OutputStreamWriter(bufferout)), true);

            if (compileModeNumber == 0) {
                commandInput.println ("./scan " +
temporaryFileToBeCompiled.getName() );
                //System.out.println("Command was sent to BASH : " + "./scan " +
fileName);
            }
            if (compileModeNumber == 1) {

```

```

        commandInput.println ("../scan -no_wild " +
temporaryFileToBeCompiled.getName() );
//System.out.println("Command was sent to BASH : " + "./scan -no_wild "
+ fileName);
    }
    if (compileModeNumber == 2) {
        commandInput.println ("../scan -no_implicit_deny " +
temporaryFileToBeCompiled.getName() );
//System.out.println("Command was sent to BASH : " + "./scan -
no_implicit_deny " + fileName);
    }

    commandInput.close();
} // End of try
catch( Exception e ) {
    System.err.println(e);
} // End of catch

} // End of method run

} // End of class OutputStuffThread

/**
 * Method      : displayCompilerOutputMethod
 * Purpose     : This method returns the output of the PPL compiler
 * to the caller (MainGui class) in order to display the output in
 * the text area of the main GUI
 * Parameters  : None
 */

public Vector displayCompilerOutputMethod () {
    return compilerOutputVector;
}

/**
 * Method      : createFileToBeCompiledMethod
 * Purpose     : This method creates a temporary file and puts the
 * contents of the node, link, path, class, type and policy vectors
 * in this temporary file for the compilation process. Calls a
 * method for writing each of the vectors.
 * Parameters  : None
 */

private void createFileToBeCompiledMethod () {
    try {
        coStream = new PrintStream ( new FileOutputStream (
temporaryFileToBeCompiled ) );
    } // End of try
    catch (IOException e) {
        JOptionPane.showMessageDialog (null, "Error opening temporary
compilation file", "Error", JOptionPane.ERROR_MESSAGE);
    } // End of catch

    writeNodesToTemporaryFileMethod ();

```

```

writeLinksToTemporaryFileMethod ();
writePathsToTemporaryFileMethod ();
writeClassesToTemporaryFileMethod ();
writeTypesToTemporaryFileMethod ();
writePolicyMakerToTemporaryFileMethod ();
writePoliciesToTemporaryFileMethod ();

coStream.close();
} // End of method createFileToBeCompiledMethod

/**
 * Method      : writeNodesToTemporaryFileMethod
 * Purpose     : This method writes the contents of the node vector
 * to temporary file in the format that PPL compiler accepts
 * Parameters  : None
 */

private void writeNodesToTemporaryFileMethod () {
    if ( !( nodeVector.isEmpty() ) ) {
        coStream.println ("/*");
        coStream.println ("* Definitions of the nodes and the
parameters of those nodes in the network");
        coStream.println ("*/");
    } // End of if

    Enumeration enum = nodeVector.elements();

    while ( enum.hasMoreElements() ) {
        Node tempNode = new Node ();
        tempNode = ( Node ) enum.nextElement();
        coStream.println ( "define " + "node " + tempNode.toString() +
";" );

        Vector parametersVector = new Vector (1);

        if ( tempNode.getHasBwParameter() ) {
            String s = "BW := " + tempNode.getBwValue() + " MBPS";
            parametersVector.addElement (s);
        }
        if ( tempNode.getHasDelayParameter() ) {
            parametersVector.addElement ( "delay ( )" );
        }
        if ( tempNode.getHasLossRateParameter() ) {
            parametersVector.addElement ( "loss_rate ( )" );
        }
        if ( tempNode.getHasJitterParameter() ) {
            parametersVector.addElement ( "jitter ( )" );
        }
        if ( tempNode.getHasUsedBwParameter() ) {
            parametersVector.addElement ( "used_bw ( )" );
        }

        if ( !( parametersVector.isEmpty() ) ) {
            coStream.print ( "define " + "node_param " +
tempNode.toString() + " { " );

```

```

    }

    Enumeration enumForParametersVector =
parametersVector.elements();
    for ( int i = 0; enumForParametersVector.hasMoreElements ();
i++) {
        String tempString;
        tempString = ( String ) enumForParametersVector.nextElement
();
        if ( i >= 1 ) {
            coStream.print ( ", " );
        }

        coStream.print ( tempString );

    } // End of for

    if ( !( parametersVector.isEmpty() ) ) {
        coStream.print ( " };";
        coStream.println();
        coStream.println();
    }
    else {
        coStream.println();
    }

    } // End of while

} // End of method writeNodesToTemporaryFileMethod

/**
 * Method      : writeLinksToTemporaryFileMethod
 * Purpose     : This method writes the contents of the link vector
 * to temporary file in the format that PPL compiler accepts
 * Parameters  : None
 */

private void writeLinksToTemporaryFileMethod () {
    if ( !( linkVector.isEmpty() ) ) {
        coStream.println ("/*");
        coStream.println ("* Definitions the links and the parameters
of those links in the network");
        coStream.println ("*/");
    } // End of if

    Enumeration enum = linkVector.elements();

    while ( enum.hasMoreElements() ) {

        Node a = new Node ();
        Node b = new Node ();
        Link tempLink = new Link ("", a, b, 0, false, false, false,
false, false);
        tempLink = ( Link ) enum.nextElement();

```



```

        coStream.println ( "define " + "link " + tempLink.getName() +
" <" + ( tempLink.getNode1() ).toString() + ", " + (
tempLink.getNode2() ).toString() + ">;" );

        Vector parametersVector = new Vector (1);

        if ( tempLink.getHasBwParameter() ) {
            String s = "BW := " + tempLink.getBwValue() + " MBPS";
            parametersVector.addElement (s);
        }
        if ( tempLink.getHasDelayParameter() ) {
            parametersVector.addElement ( "delay ( )" );
        }
        if ( tempLink.getHasLossRateParameter() ) {
            parametersVector.addElement ( "loss_rate ( )" );
        }
        if ( tempLink.getHasJitterParameter() ) {
            parametersVector.addElement ( "jitter ( )" );
        }
        if ( tempLink.getHasUsedBwParameter() ) {
            parametersVector.addElement ( "used_bw ( )" );
        }

        if ( !( parametersVector.isEmpty() ) ) {
            coStream.print ( "define " + "link_param " +
tempLink.toString() + " { " );
        }

        Enumeration enumForParametersVector =
parametersVector.elements();
        for ( int i = 0; enumForParametersVector.hasMoreElements ();
i++) {
            String tempString;
            tempString = ( String ) enumForParametersVector.nextElement
();
            if ( i >= 1 ) {
                coStream.print ( ", " );
            }

            coStream.print ( tempString );

        } // End of for

        if ( !( parametersVector.isEmpty() ) ) {
            coStream.print ( " };" );
            coStream.println();
            coStream.println();
        }
        else {
            coStream.println();
        }

    } // End of while

} // End of method writeLinksToTemporaryFileMethod

```

```

/**
 * Method      : writePathsToTemporayFileMethod
 * Purpose     : This method writes the contents of the path vector
 * to temporary file in the format that PPL compiler accepts
 * Parameters  : None
 */

private void writePathsToTemporayFileMethod () {
    if ( !( pathVector.isEmpty() ) ) {
        coStream.println ("/*");
        coStream.println ("* Definitions the paths and the parameters
of those paths in the network");
        coStream.println ("*/");
    } // End of if

    Enumeration enum = pathVector.elements();

    while ( enum.hasMoreElements() ) {

        Path tempPath = new Path ();
        tempPath = ( Path ) enum.nextElement();

        coStream.print ( "define " + "path " + tempPath.getName() + "
{<" );

        // The names of the nodes in the path will be placed in
tempPathNodeVector
        Vector tempStringVector = new Vector (1);

        Enumeration enumString = ( tempPath.getPathNodesVector()
).elements();
        while ( enumString.hasMoreElements() ) {
            Node tempNode = new Node ();
            tempNode = ( Node ) enumString.nextElement();
            tempStringVector.addElement ( tempNode.toString() );
        }

        Enumeration enumPath = tempStringVector.elements();
        for ( int i = 0; enumPath.hasMoreElements (); i++) {
            String tempString;
            tempString = ( String ) enumPath.nextElement();
            if ( i >=1 ) {
                coStream.print ( ", " );
            } // End of if
            coStream.print ( tempString );
        } // End of for

        coStream.println ( ">};" );

        Vector parametersVector = new Vector (1);

        if ( tempPath.getHasBwParameter() ) {

```

```

        String s = "BW := " + tempPath.getBwValue() + " MBPS";
        parametersVector.addElement (s);
    }
    if ( tempPath.getHasDelayParameter() ) {
        parametersVector.addElement ( "delay ( )" );
    }
    if ( tempPath.getHasLossRateParameter() ) {
        parametersVector.addElement ( "loss_rate ( )" );
    }
    if ( tempPath.getHasJitterParameter() ) {
        parametersVector.addElement ( "jitter ( )" );
    }
    if ( tempPath.getHasUsedBwParameter() ) {
        parametersVector.addElement ( "used_bw ( )" );
    }

    if ( !( parametersVector.isEmpty() ) ) {
        coStream.print ( "define " + "path_param " +
tempPath.toString() + " { " );
    }

    Enumeration enumForParametersVector =
parametersVector.elements();
    for ( int i = 0; enumForParametersVector.hasMoreElements ();
i++) {
        String tempString;
        tempString = ( String ) enumForParametersVector.nextElement
        ();
        if ( i >= 1 ) {
            coStream.print ( ", " );
        }

        coStream.print ( tempString );

    } // End of for

    if ( !( parametersVector.isEmpty() ) ) {
        coStream.print ( " };");
        coStream.println ();
        coStream.println ();
    }
    else {
        coStream.println ();
    }

    } // End of while

} // End of method writePathsToTemporayFileMethod

/**
 * Method      : writeClassesToTemporaryFileMethod
 * Purpose     : This method writes the contents of the class vector
 * to temporary file in the format that PPL compiler accepts
 * Parameters  : None
 */

```

```

private void writeClassesToTemporaryFileMethod () {
    if ( !( classVector.isEmpty() ) ) {
        coStream.println ("/*");
        coStream.println ("* User defined classes which can be used in
the target element of policy rules");
        coStream.println ("*/");
    } // End of if

    Enumeration enum = classVector.elements();

    while ( enum.hasMoreElements() ) {
        Class tempClass = new Class ();
        tempClass = ( Class ) enum.nextElement();

        coStream.print ( "define " + "class " + tempClass.getName() +
" { " );

        // Class member names will be placed in tempStringVector
        Vector tempStringVector = new Vector (1);

        Enumeration enumString = ( tempClass.getClassMembersVector()
).elements();

        for ( int i = 0; enumString.hasMoreElements (); i++) {
            String tempString;
            tempString = ( String ) enumString.nextElement();
            if ( i >=1 ) {
                coStream.print ( ", " );
            } // End of if

            coStream.print ( tempString );
        } // End of for

        coStream.println ( "};");
    } // End of while

    coStream.println();

} // End of method writeClassesToTemporaryFileMethod

/**
 * Method      : writeTypesToTemporaryFileMethod
 * Purpose     : This method writes the contents of the type vector
 * to temporary file in the format that PPL compiler accepts
 * Parameters  : None
 */

private void writeTypesToTemporaryFileMethod () {
    if ( !( typeVector.isEmpty() ) ) {
        coStream.println ("/*");
        coStream.println ("* User defined types which can be used in
the conditional element of policy rules");
        coStream.println ("*/");
    } // End of if

```

```

Enumeration enum = typeVector.elements();

while ( enum.hasMoreElements() ) {
    Type tempType = new Type ();

    tempType = ( Type ) enum.nextElement();

    coStream.print ( "define " + "type " + tempType.getName() + "
{" );

    // Type member names will be placed in tempStringVector
    Vector tempStringVector = new Vector (1);

    Enumeration enumString = ( tempType.getTypeMembersVector()
).elements();

    for ( int i = 0; enumString.hasMoreElements (); i++) {
        String tempString;
        tempString = ( String ) enumString.nextElement();
        if ( i >=1 ) {
            coStream.print ( ", " );
        } // End of if

        coStream.print ( tempString );
    } // End of for

    coStream.println ( "};");
} // End of while

coStream.println();

} // End of method writeTypesToTemporaryFileMethod

/**
 * Method      : writePolicyMakerToTemporayFileMethod
 * Purpose     : This method writes the contents of the policy maker
 * vector to temporary file in the format that PPL compiler accepts
 * Parameters  : None
 */

private void writePolicyMakerToTemporayFileMethod () {

    coStream.println ("/*");
    coStream.println ("* Definition of policy makers");
    coStream.println ("*/");

    Enumeration enum = policyMakerVector.elements();
    while ( enum.hasMoreElements() ) {
        PolicyMaker tempPolicyMaker = new PolicyMaker ("", "", 1);
        tempPolicyMaker = ( PolicyMaker ) enum.nextElement();
        coStream.println("define " + "policy_maker " +
tempPolicyMaker.getLoginID() + "(" +
tempPolicyMaker.getPolicyMakerPriority() + ");");
    }
}

```

```

        coStream.println();

    } // End of method writePolicyMakerToTemporayFileMethod

/**
 * Method      : writePoliciesToTemporayFileMethod
 * Purpose     : This method writes the contents of the policy
 *               vector to temporary file in the format that PPL compiler accepts
 * Parameters  : None
 */

private void writePoliciesToTemporayFileMethod () {
    if ( !( policyVector.isEmpty() ) ) {
        coStream.println ("/*");
        coStream.println ("* Format of Policy Term: PolicyID UserID @
{paths} {targets} {conditions} {action_items};");
        coStream.println ("*/");
    } // End of if

    Enumeration enumForPolicies = policyVector.elements();
    while ( enumForPolicies.hasMoreElements() ) {

        Vector nodeLinkPathNames = new Vector (1); // Node, link ve
path names will be placed in it as Strings.
        Vector conditionElementVector = new Vector (1); // Condition
elements will be placed in it as Strings

        Policy tempPolicy = new Policy ();
        tempPolicy = ( Policy ) enumForPolicies.nextElement();

        coStream.print ( tempPolicy.getPolicyID() + "    " +
tempPolicy.getPolicyMakerID() + "    @    " + "{ " );

        Enumeration enumForPolicyNodeVector = (
tempPolicy.getNodesInPolicyPathVector() ).elements();
        while ( enumForPolicyNodeVector.hasMoreElements() ) {
            Node tempNode = new Node ();
            tempNode = ( Node ) enumForPolicyNodeVector.nextElement ();
            nodeLinkPathNames.addElement( tempNode.toString() );
        }

        Enumeration enumForPolicyLinkVector = (
tempPolicy.getLinksInPolicyPathVector() ).elements();
        while ( enumForPolicyLinkVector.hasMoreElements() ) {
            Link tempLink = new Link ();
            tempLink = ( Link ) enumForPolicyLinkVector.nextElement ();
            nodeLinkPathNames.addElement( tempLink.toString() );
        }

        Enumeration enumForPolicyPathVector = (
tempPolicy.getPathsInPolicyPathVector() ).elements();
        while ( enumForPolicyPathVector.hasMoreElements() ) {
            Path tempPath = new Path ();
            tempPath = ( Path ) enumForPolicyPathVector.nextElement ();
            nodeLinkPathNames.addElement( tempPath.toString() );

```

```

    }

    Enumeration enumForNodeLinkPathNamesVector =
nodeLinkPathNames.elements();
    for ( int i = 0;
enumForNodeLinkPathNamesVector.hasMoreElements (); i++) {
        String tempString;
        tempString = ( String )
enumForNodeLinkPathNamesVector.nextElement ();
        if ( i >= 1 ) {
            coStream.print ( ", " );
        }

        coStream.print ( tempString ); // Write the nodes, links
and paths of the policy

    } // End of for
    coStream.print ( " }    { " );

    // write target element to file

    // If this policy has a wild card character in its target
element.
    if ( ( tempPolicy.getPolicyTarget() ).getTargetAllProperty() )
    {
        coStream.print ( "*" }    " );
    } // End of if
    else {
        Enumeration enumForTargetVector = ( (
tempPolicy.getPolicyTarget() ).getPolicyTargets() ).elements();
        for ( int i = 0; enumForTargetVector.hasMoreElements ();
i++) {
            String tempString;
            tempString = ( ( OneTarget )
enumForTargetVector.nextElement () ).outputMethod();
            if ( i >=1 ) {
                coStream.print ( " || " );
            } // End of if

            coStream.print ( tempString ); // Write the target
elements of the policy

        } // End of for

        coStream.print ( "}"    " );

    } // End of else

    // write condition element to file

    coStream.print ( "{ " );

    // If this policy has a wild card character in its condition
element.

```

```

        if ( ( tempPolicy.getPolicyCondition()
).getNoConditionProperty() ) {
            coStream.print ( "*" }    " );
        } // End of if

        // collect all the condition names in the
conditionElementVector
        else{
            Enumeration enumForPriorityConditionVector = ( (
tempPolicy.getPolicyCondition() ).getPriorityConditionVector()
).elements();
            while ( enumForPriorityConditionVector.hasMoreElements() )
        {
                OnePriority tempOnePriority = new OnePriority ();
                tempOnePriority = ( OnePriority )
enumForPriorityConditionVector.nextElement ();
                conditionElementVector.addElement(
tempOnePriority.toString() );
            }

            Enumeration enumForHopCountConditionVector = ( (
tempPolicy.getPolicyCondition() ).getHopCountConditionVector()
).elements();
            while ( enumForHopCountConditionVector.hasMoreElements() )
        {
                OneHopCount tempOneHopCount = new OneHopCount ();
                tempOneHopCount = ( OneHopCount )
enumForHopCountConditionVector.nextElement ();
                conditionElementVector.addElement(
tempOneHopCount.toString() );
            }

            Enumeration enumForTimeConditionVector = ( (
tempPolicy.getPolicyCondition() ).getTimeConditionVector()
).elements();
            while ( enumForTimeConditionVector.hasMoreElements() ) {
                OneTime tempOneTime = new OneTime ();
                tempOneTime = ( OneTime )
enumForTimeConditionVector.nextElement ();
                conditionElementVector.addElement(
tempOneTime.toString() );
            }

            Enumeration enumForSrcIPAddressConditionVector = ( (
tempPolicy.getPolicyCondition() ).getSrcIPAddressConditionVector()
).elements();
            while (
enumForSrcIPAddressConditionVector.hasMoreElements() ) {
                OneSrcIPAddress tempOneSrcIPAddress = new
OneSrcIPAddress ();
                tempOneSrcIPAddress = ( OneSrcIPAddress )
enumForSrcIPAddressConditionVector.nextElement ();
                conditionElementVector.addElement(
tempOneSrcIPAddress.toString() );
            }

```



```

        Enumeration enumForBWConditionVector = ( (
tempPolicy.getPolicyCondition() ).getBWConditionVector() ).elements();
        while ( enumForBWConditionVector.hasMoreElements() ) {
            OneBW tempOneBW = new OneBW ();
            tempOneBW = ( OneBW )
enumForBWConditionVector.nextElement ();
            conditionElementVector.addElement( tempOneBW.toString()
);
        }

        Enumeration enumForUserIDConditionVector = ( (
tempPolicy.getPolicyCondition() ).getUserIDConditionVector()
).elements();
        while ( enumForUserIDConditionVector.hasMoreElements() ) {
            OneUserID tempOneUserID = new OneUserID ();
            tempOneUserID = ( OneUserID )
enumForUserIDConditionVector.nextElement ();
            conditionElementVector.addElement(
tempOneUserID.toString() );
        }

        Enumeration enumForTypeConditionVector = ( (
tempPolicy.getPolicyCondition() ).getTypeConditionVector()
).elements();
        while ( enumForTypeConditionVector.hasMoreElements() ) {
            OneType tempOneType = new OneType ();
            tempOneType = ( OneType )
enumForTypeConditionVector.nextElement ();
            conditionElementVector.addElement(
tempOneType.toString() );
        }

        Enumeration enumForParameterConditionVector = ( (
tempPolicy.getPolicyCondition() ).getParameterConditionVector()
).elements();
        while ( enumForParameterConditionVector.hasMoreElements() )
        {
            OneParameter tempOneParameter = new OneParameter ();
            tempOneParameter = ( OneParameter )
enumForParameterConditionVector.nextElement ();
            conditionElementVector.addElement(
tempOneParameter.toString() );
        }

        // All 8 types of conditions are placed in the
conditionElementVector and they are written to file by the foolowing
Enumeration
        Enumeration enumForConditionElementVector =
conditionElementVector.elements();
        for ( int i = 0;
enumForConditionElementVector.hasMoreElements (); i++) {
            String tempString;
            tempString = ( String )
enumForConditionElementVector.nextElement ();
            if ( i >= 1 ) {
                coStream.print ( " && " );

```

```

        }
        coStream.print ( tempString );
    } // End of for

    coStream.print ( " }    " );

} // End of else (condition part does not have a wild card
character)

// write action element of policy to JTextArea
coStream.print ( "{ " );

if ( ( tempPolicy.getPolicyAction() ).getHasDenyAction() ) {
    coStream.println ("deny };");
}
else {
    Vector actionElementVector = new Vector (1);
    if ( ( tempPolicy.getPolicyAction() ).getHasPermitAction()
) {
        actionElementVector.addElement("permit");
    }
    if ( ( tempPolicy.getPolicyAction() ).getPriorityValue() !=
0 ) {
        actionElementVector.addElement("priority" + " := " + (
tempPolicy.getPolicyAction() ).getPriorityValue() );
    }
    if ( ( tempPolicy.getPolicyAction() ).getHopCountValue() !=
0 ) {
        actionElementVector.addElement("hopCount" + " := " + (
tempPolicy.getPolicyAction() ).getHopCountValue() );
    }
    if ( ( tempPolicy.getPolicyAction() ).getAllocatedBWValue()
!= 0 ) {
        actionElementVector.addElement("allocated_bw" + " := " +
( tempPolicy.getPolicyAction() ).getAllocatedBWValue() + " MBPS");
    }
    if ( ( tempPolicy.getPolicyAction() ).getMaxLossRateValue()
!= 0 ) {
        actionElementVector.addElement("maxLossRate" + " := " +
( tempPolicy.getPolicyAction() ).getMaxLossRateValue() + " %");
    }
    if ( ( tempPolicy.getPolicyAction() ).getDelayBoundValue()
!= 0 ) {
        actionElementVector.addElement("delayBound" + " := " + (
tempPolicy.getPolicyAction() ).getDelayBoundValue() + " msec");
    }
    if ( ( tempPolicy.getPolicyAction()
).getSecurityLevelValue() != 0 ) {
        actionElementVector.addElement("securityLevel" + " := "
+ ( tempPolicy.getPolicyAction() ).getSecurityLevelValue() );
    }
}

Enumeration enumForActionElementVector =
actionElementVector.elements();

```

```

        for ( int i = 0; enumForActionElementVector.hasMoreElements
()); i++) {
            String tempString;
            tempString = ( String )
enumForActionElementVector.nextElement ();
            if ( i >= 1 ) {
                coStream.print ( ", " );
            }
            coStream.print ( tempString );
        } // End of for
        coStream.println ( " }; " );
    } // End of else
        coStream.println();
    } // End of while ( enumForPolicies.hasMoreElements() )

} // End of method writePoliciesToTemporayFileMethod

/**
 * Method      : saveToFileMethod
 * Purpose     : This method saves the current network definition
 * and policies to file by using ObjectOutputStream class. It saves
 * node, link, path, class, type and policy vectors. It is called
 * when the user chooses save as menu option.
 * Parameters  : None
 */

public void saveToFileMethod () {

    JFileChooser fileChooser = new JFileChooser ();
    fileChooser.setSelectionMode(JFileChooser.FILES_ONLY);
    // A file filter which allows only directories and .txt files is
instantiated and added to the JFileChooser
    txtFilter filter = new txtFilter ();
    fileChooser.addChoosableFileFilter(filter);

    int result = fileChooser.showDialog (null, "Save File");
    // If the user clicked Cancel button on dialog then do nothing
    if ( result == JFileChooser.CANCEL_OPTION ) {

        /*("New" menu item - Yes option - Cancel option) then the flag is
set to true so that the text area of the GUI will not be cleared and
the network element and topology vectors will not be emptied. */

        cancelOptionSelectedInSaveFile = true;
    }
    else {
        networkFile = fileChooser.getSelectedFile();
        if ( networkFile == null || (networkFile.getName()).equals("")
) {

            // Then do nothing
            /*("New" menu item - Yes option - Cancel option) then the flag is
set to true so that the text area of the GUI will not be cleared and
the network element and topology vectors will not be emptied. */

```

```

        cancelOptionSelectedInSaveFile = true;

    }
    else {
        try {
            outputForNetwork = new ObjectOutputStream ( new
FileOutputStream (networkFile) );
            outputForNetwork.writeObject(nodeVector);
            outputForNetwork.writeObject(linkVector);
            outputForNetwork.writeObject(pathVector);
            outputForNetwork.writeObject(classVector);
            outputForNetwork.writeObject(typeVector);
            outputForNetwork.writeObject(policyVector);
            outputForNetwork.flush();
            outputForNetwork.close();

            File fileInPPLFormat = new File (
networkFile.getParent(), "PPL" + networkFile.getName() );
            pStream = new PrintStream ( new FileOutputStream (
fileInPPLFormat ) );
            writeToFileInPPLFormatMethod ();

        } // End of try
        catch (IOException e) {
            JOptionPane.showMessageDialog (null, "Error opening
file", "Error", JOptionPane.ERROR_MESSAGE);
        } // End of catch

    } // End of else

} // End of else

} // End of saveToFileMethod

/**
 * Method      : savePoliciesToFileMethod
 * Purpose     : This method saves only policies together with the
 * definition of the current policy maker to file. It is called when
 * the user chooses save policy menu option.
 * Parameters  : None
 */

public void savePoliciesToFileMethod () {

    JFileChooser fileChooser = new JFileChooser ();
    fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
    // A file filter which allows only directories and .txt files is
instantiated and added to the JFileChooser
    txtFilter filter = new txtFilter ();
    fileChooser.addChoosableFileFilter(filter);

    int result = fileChooser.showDialog(null, "Save Policies");
    // If the user clicked Cancel button on dialog then do nothing
    if ( result == JFileChooser.CANCEL_OPTION ) {

```

```

    }
    else {
        policyFile = fileChooser.getSelectedFile();
        if ( policyFile == null || (policyFile.getName()).equals("") )
    {
        // Then do nothing
    }
    else {
        try {
            outputForPolicy = new ObjectOutputStream ( new
FileOutputStream (policyFile) );
            outputForPolicy.writeObject(policyVector);
            outputForPolicy.flush();
            outputForPolicy.close();

            File fileInPPLFormat = new File (
policyFile.getParent(), "PPL" + policyFile.getName() );
            poStream = new PrintStream ( new FileOutputStream (
fileInPPLFormat ) );
            writePoliciesToPolicyFileMethod ();
            poStream.close();

        } // End of try
        catch (IOException e) {
            JOptionPane.showMessageDialog (null, "Error opening
file", "Error", JOptionPane.ERROR_MESSAGE);
        } // End of catch

    } // End of else

} // End of else

} // End of savePoliciesToFileMethod

/**
 * Class      : txtFilter
 * Purpose    : By means of this class, file filter which will show
 * only directories and .txt files which do not start with "PPL"
 * will be instantiated. .txt files which starts with PPL are formed
 * for compiling and printing not for saving or opening. So it is
 * good to filter them for not confusing the user.
 */

private class txtFilter extends javax.swing.filechooser.FileFilter {
    public boolean accept (File f) {
        if ( f.isDirectory() || ( (f.getName()).endsWith(".txt") &&
!(f.getName()).startsWith("PPL") ) ) {
            return true;
        }
        else {
            return false;
        }
    }

    public String getDescription () {

```

```

        return "txt files";
    }

} // End of class txtFilter

```

```

/**
 * Method      : writeToFileInPPLFormatMethod
 * Purpose     : This method saves the current network definition
 * and policies to file by using PrintStream class. It saves node,
 * link, path, class, type and policy vectors in a readable form. It
 * is called from saveToFileMethod. It has 7 method calls for
 * writing each of the network elements, policy makers and policies.
 * Parameters  : None
 */

private void writeToFileInPPLFormatMethod () {
    writeNodesToFileMethod ();
    writeLinksToFileMethod ();
    writePathsToFileMethod ();
    writeClassesToFileMethod ();
    writeTypesToFileMethod ();
    writePolicyMakerToFileMethod ();
    writePoliciesToFileMethod ();

    pStream.close();
} // End of method writeToFileInPPLFormatMethod

/**
 * Method      : writeNodesToFileMethod
 * Purpose     : This method writes the node and node parameter
 * definitions to file in PPL format.
 * Parameters  : None
 */

private void writeNodesToFileMethod () {

    if ( !( nodeVector.isEmpty() ) ) {
        pStream.println ("/*");
        pStream.println ("* Definitions of the nodes and the
parameters of those nodes in the network");
        pStream.println ("*/");
    } // End of if

    Enumeration enum = nodeVector.elements();

    while ( enum.hasMoreElements() ) {

```

```

Node tempNode = new Node ();
tempNode = ( Node ) enum.nextElement();
pStream.println ( "define " + "node " + tempNode.toString() +
";" );

Vector parametersVector = new Vector (1);

if ( tempNode.getHasBwParameter() ) {
    String s = "BW := " + tempNode.getBwValue() + " MBPS";
    parametersVector.addElement (s);
}
if ( tempNode.getHasDelayParameter() ) {
    parametersVector.addElement ( "delay ( )" );
}
if ( tempNode.getHasLossRateParameter() ) {
    parametersVector.addElement ( "loss_rate ( )" );
}
if ( tempNode.getHasJitterParameter() ) {
    parametersVector.addElement ( "jitter ( )" );
}
if ( tempNode.getHasUsedBwParameter() ) {
    parametersVector.addElement ( "used_bw ( )" );
}

if ( !( parametersVector.isEmpty() ) ) {
    pStream.print ( "define " + "node_param " +
tempNode.toString() + " { " );
}

Enumeration enumForParametersVector =
parametersVector.elements();
for ( int i = 0; enumForParametersVector.hasMoreElements ();
i++) {
    String tempString;
    tempString = ( String ) enumForParametersVector.nextElement
();
    if ( i >= 1 ) {
        pStream.print ( ", " );
    }

    pStream.print ( tempString );
} // End of for

if ( !( parametersVector.isEmpty() ) ) {
    pStream.print ( " };" );
    pStream.println();
    pStream.println();
}
else {
    pStream.println();
}

} // End of while

} // End of method writeNodesToFileMethod

```

```

/**
 * Method      : writeLinksToFileMethod
 * Purpose     : This method writes the link and link parameter
 * definitions  to file in PPL format.
 * Parameters  : None
 */

private void writeLinksToFileMethod () {

    if ( !( linkVector.isEmpty() ) ) {
        pStream.println ("/*");
        pStream.println ("* Definitions the links and the parameters
of those links in the network");
        pStream.println ("*/");
    } // End of if

    Enumeration enum = linkVector.elements();

    while ( enum.hasMoreElements() ) {

        Node a = new Node ();
        Node b = new Node ();
        Link tempLink = new Link ("", a, b, 0, false, false, false,
false, false);
        tempLink = ( Link ) enum.nextElement();

        pStream.println ( "define " + "link " + tempLink.getName() + "
<" + ( tempLink.getNode1() ).toString() + ", " + ( tempLink.getNode2()
).toString() + ">;" );

        Vector parametersVector = new Vector (1);

        if ( tempLink.getHasBwParameter() ) {
            String s = "BW := " + tempLink.getBwValue() + " MBPS";
            parametersVector.addElement (s);
        }
        if ( tempLink.getHasDelayParameter() ) {
            parametersVector.addElement ( "delay ( )" );
        }
        if ( tempLink.getHasLossRateParameter() ) {
            parametersVector.addElement ( "loss_rate ( )" );
        }
        if ( tempLink.getHasJitterParameter() ) {
            parametersVector.addElement ( "jitter ( )" );
        }
        if ( tempLink.getHasUsedBwParameter() ) {
            parametersVector.addElement ( "used_bw ( )" );
        }

        if ( !( parametersVector.isEmpty() ) ) {
            pStream.print ( "define " + "link_param " +
tempLink.toString() + " { " );
        }
    }
}

```



```

        Enumeration enumForParametersVector =
parametersVector.elements();
        for ( int i = 0; enumForParametersVector.hasMoreElements ();
i++) {
            String tempString;
            tempString = ( String ) enumForParametersVector.nextElement
();
            if ( i >= 1 ) {
                pStream.print ( ", " );
            }

            pStream.print ( tempString );

        } // End of for

        if ( !( parametersVector.isEmpty() ) ) {
            pStream.print ( " };");
            pStream.println();
            pStream.println();
        }
        else {
            pStream.println();
        }

    } // End of while

} // End of method writeLinksToFileMethod

/**
 * Method      : writePathsToFileMethod
 * Purpose     : This method writes the path and path parameter
 * definitions to file in PPL format.
 * Parameters  : None
 */

private void writePathsToFileMethod () {

    if ( !( pathVector.isEmpty() ) ) {
        pStream.println ("/*");
        pStream.println ("* Definitions the paths and the parameters
of those paths in the network");
        pStream.println ("*/");
    } // End of if

    Enumeration enum = pathVector.elements();

    while ( enum.hasMoreElements() ) {

        Path tempPath = new Path ();
        tempPath = ( Path ) enum.nextElement();

        pStream.print ( "define " + "path " + tempPath.getName() + "
{<" );

```

```

        // The names of the nodes in the path will be placed in
tempPathNodeVector
        Vector tempStringVector = new Vector (1);

        Enumeration enumString = ( tempPath.getPathNodesVector()
).elements();
        while ( enumString.hasMoreElements() ) {
            Node tempNode = new Node ();
            tempNode = ( Node ) enumString.nextElement();
            tempStringVector.addElement ( tempNode.toString() );
        }

        Enumeration enumPath = tempStringVector.elements();
        for ( int i = 0; enumPath.hasMoreElements (); i++) {
            String tempString;
            tempString = ( String ) enumPath.nextElement();
            if ( i >=1 ) {
                pStream.print ( ", " );
            } // End of if
            pStream.print ( tempString );
        } // End of for

        pStream.println ( ">};");

        Vector parametersVector = new Vector (1);

        if ( tempPath.getHasBwParameter() ) {
            String s = "BW := " + tempPath.getBwValue() + " MBPS";
            parametersVector.addElement (s);
        }
        if ( tempPath.getHasDelayParameter() ) {
            parametersVector.addElement ( "delay ( )" );
        }
        if ( tempPath.getHasLossRateParameter() ) {
            parametersVector.addElement ( "loss_rate ( )" );
        }
        if ( tempPath.getHasJitterParameter() ) {
            parametersVector.addElement ( "jitter ( )" );
        }
        if ( tempPath.getHasUsedBwParameter() ) {
            parametersVector.addElement ( "used_bw ( )" );
        }

        if ( !( parametersVector.isEmpty() ) ) {
            pStream.print ( "define " + "path_param " +
tempPath.toString() + " { " );
        }

        Enumeration enumForParametersVector =
parametersVector.elements();
        for ( int i = 0; enumForParametersVector.hasMoreElements ();
i++) {
            String tempString;
            tempString = ( String ) enumForParametersVector.nextElement
());
            if ( i >= 1 ) {

```

```

        pStream.print ( ", " );
    }

    pStream.print ( tempString );

} // End of for

if ( !( parametersVector.isEmpty() ) ) {
    pStream.print ( " };");
    pStream.println ();
    pStream.println ();
}
else {
    pStream.println ();
}

} // End of while

} // End of method writePathsToFileMethod

/**
 * Method      : writeClassesToFileMethod
 * Purpose     : This method writes the classes to file in PPL
 * format.
 * Parameters  : None
 */

private void writeClassesToFileMethod () {
    if ( !( classVector.isEmpty() ) ) {
        pStream.println ("/*");
        pStream.println ("* User defined classes which can be used in
the target element of policy rules");
        pStream.println ("*/");
    } // End of if

    Enumeration enum = classVector.elements();

    while ( enum.hasMoreElements() ) {
        Class tempClass = new Class ();
        tempClass = ( Class ) enum.nextElement();

        pStream.print ( "define " + "class " + tempClass.getName() + "
{" );

        // Class member names will be placed in tempStringVector
        Vector tempStringVector = new Vector (1);

        Enumeration enumString = ( tempClass.getClassMembersVector()
).elements();

        for ( int i = 0; enumString.hasMoreElements (); i++) {
            String tempString;
            tempString = ( String ) enumString.nextElement();
            if ( i >=1 ) {
                pStream.print ( ", " );
            }
        }
    }
}

```

```

        } // End of if

        pStream.print ( tempString );
    } // End of for

    pStream.println ( "};");
} // End of while

pStream.println();

} // End of method writeClassesToFileMethod

/**
 * Method      : writeTypesToFileMethod
 * Purpose     : This method writes the types to file in PPL format.
 * Parameters  : None
 */

private void writeTypesToFileMethod () {
    if ( !( typeVector.isEmpty() ) ) {
        pStream.println ("/*");
        pStream.println ("* User defined types which can be used in
the conditional element of policy rules");
        pStream.println ("*/");
    } // End of if

    Enumeration enum = typeVector.elements();

    while ( enum.hasMoreElements() ) {
        Type tempType = new Type ();

        tempType = ( Type ) enum.nextElement();

        pStream.print ( "define " + "type " + tempType.getName() + "
{" " );

        // Type member names will be placed in tempStringVector
        Vector tempStringVector = new Vector (1);

        Enumeration enumString = ( tempType.getTypeMembersVector()
).elements();

        for ( int i = 0; enumString.hasMoreElements (); i++) {
            String tempString;
            tempString = ( String ) enumString.nextElement();
            if ( i >=1 ) {
                pStream.print ( ", " );
            } // End of if

            pStream.print ( tempString );
        } // End of for

        pStream.println ( "};");
    } // End of while
}

```

```

        pStream.println();

    } // End of method writeTypesToFileMethod


/**
 * Method      : writePolicyMakerToFileMethod
 * Purpose     : This method writes the policy makers to file in PPL
 * format.
 * Parameters  : None
 */

private void writePolicyMakerToFileMethod () {

    // If the login ID of the policy maker is not "administrator",
    then write only the login ID
    // of the user
    if ( !(policyMakerID.equalsIgnoreCase("administrator")) ) {
        pStream.println ("/*");
        pStream.println ("* Definition of policy maker currently
logged in");
        pStream.println ("*/");

        int currentPolicyMakerPriority = 0;
        Enumeration enum = policyMakerVector.elements();
        while ( enum.hasMoreElements() ) {
            PolicyMaker tempPolicyMaker = new PolicyMaker ("", "", 1);
            tempPolicyMaker = ( PolicyMaker ) enum.nextElement();
            if ( ( tempPolicyMaker.getLoginID() ).equalsIgnoreCase (
policyMakerID ) ) {
                currentPolicyMakerPriority =
tempPolicyMaker.getPolicyMakerPriority();
                break;
            }
        }

        pStream.print ( "define " + "policy_maker " + policyMakerID +
"(" + currentPolicyMakerPriority + ");" );
        pStream.println();
    } // End of if
    // If the login ID of the policy maker is "administrator", then
write the login ID
    // of all users to compile the policies together

    else {
        pStream.println ("/*");
        pStream.println ("* Definition of policy makers");
    }
}

```

```

        pStream.println ("*/");

        Enumeration enum = policyMakerVector.elements();
        while ( enum.hasMoreElements() ) {
            PolicyMaker tempPolicyMaker = new PolicyMaker ("", "", 1);
            tempPolicyMaker = ( PolicyMaker ) enum.nextElement();
            pStream.println("define " + "policy_maker " +
tempPolicyMaker.getLoginID() + "(" +
tempPolicyMaker.getPolicyMakerPriority() + ");" );
        }
        pStream.println();
    } // End of else

} // End of method writePolicyMakerToFileMethod
/**
 * Method      : writePolicyMakerToPolicyFileMethod
 * Purpose      : This method writes the login ID of policy maker to
 * the file which includes only policies in PPL format.
 * Parameters   : None
 */

private void writePolicyMakerToPolicyFileMethod () {
    poStream.println ("/*");
    poStream.println ("* Definition of policy maker currently logged
in");
    poStream.println ("*/");

    int currentPolicyMakerPriority = 0;
    Enumeration enum = policyMakerVector.elements();
    while ( enum.hasMoreElements() ) {
        PolicyMaker tempPolicyMaker = new PolicyMaker ("", "", 1);
        tempPolicyMaker = ( PolicyMaker ) enum.nextElement();
        if ( ( tempPolicyMaker.getLoginID() ).equalsIgnoreCase (
policyMakerID ) ) {
            currentPolicyMakerPriority =
tempPolicyMaker.getPolicyMakerPriority();
            break;
        }
    }

    poStream.print ( "define " + "policy_maker " + policyMakerID +
 "(" + currentPolicyMakerPriority + ");" );
    poStream.println();
} // End of method writePolicyMakerToPolicyFileMethod

/**
 * Method      : writePoliciesToFileMethod
 * Purpose      : This method writes the policies to file in PPL
 * format (Save File menu item)
 * Parameters   : None
 */

private void writePoliciesToFileMethod () {
    if ( !( policyVector.isEmpty() ) ) {
        pStream.println ("/*");

```

```

        pStream.println ("* Format of Policy Term: PolicyID UserID @
{paths} {targets} {conditions} {action_items};");
        pStream.println ("*/");
    } // End of if

    Enumeration enumForPolicies = policyVector.elements();
    while ( enumForPolicies.hasMoreElements() ) {

        Vector nodeLinkPathNames = new Vector (1); // Node, link ve
path names will be placed in it as Strings.
        Vector conditionElementVector = new Vector (1); // Condition
elements will be placed in it as Strings

        Policy tempPolicy = new Policy ();
        tempPolicy = ( Policy ) enumForPolicies.nextElement();

        pStream.print ( tempPolicy.getPolicyID() + "    " +
tempPolicy.getPolicyMakerID() + "    @    " + "{ " );

        Enumeration enumForPolicyNodeVector = (
tempPolicy.getNodesInPolicyPathVector() ).elements();
        while ( enumForPolicyNodeVector.hasMoreElements() ) {
            Node tempNode = new Node ();
            tempNode = ( Node ) enumForPolicyNodeVector.nextElement ();
            nodeLinkPathNames.addElement( tempNode.toString() );
        }

        Enumeration enumForPolicyLinkVector = (
tempPolicy.getLinksInPolicyPathVector() ).elements();
        while ( enumForPolicyLinkVector.hasMoreElements() ) {
            Link tempLink = new Link ();
            tempLink = ( Link ) enumForPolicyLinkVector.nextElement ();
            nodeLinkPathNames.addElement( tempLink.toString() );
        }

        Enumeration enumForPolicyPathVector = (
tempPolicy.getPathsInPolicyPathVector() ).elements();
        while ( enumForPolicyPathVector.hasMoreElements() ) {
            Path tempPath = new Path ();
            tempPath = ( Path ) enumForPolicyPathVector.nextElement ();
            nodeLinkPathNames.addElement( tempPath.toString() );
        }

        Enumeration enumForNodeLinkPathNamesVector =
nodeLinkPathNames.elements();
        for ( int i = 0;
enumForNodeLinkPathNamesVector.hasMoreElements (); i++) {
            String tempString;
            tempString = ( String )
enumForNodeLinkPathNamesVector.nextElement ();
            if ( i >= 1 ) {
                pStream.print ( ", " );
            }
        }
    }

```

```

        pStream.print ( tempString ); // Write the nodes, links and
paths of the policy

    } // End of for
    pStream.print ( " }    { " );

    // write target element to file

    // If this policy has a wild card character in its target
element.
    if ( ( tempPolicy.getPolicyTarget() ).getTargetAllProperty() )
    {
        pStream.print ( "* }    " );
    } // End of if
    else {
        Enumeration enumForTargetVector = ( (
tempPolicy.getPolicyTarget() ).getPolicyTargets() ).elements();
        for ( int i = 0; enumForTargetVector.hasMoreElements ();
i++) {
            String tempString;
            tempString = ( ( OneTarget )
enumForTargetVector.nextElement () ).outputMethod();
            if ( i >=1 ) {
                pStream.print ( " || " );
            } // End of if

            pStream.print ( tempString ); // Write the target
elements of the policy

        } // End of for

        pStream.print ( " }    " );

    } // End of else

    // write condition element to file

    pStream.print ( "{ " );

    // If this policy has a wild card character in its condition
element.
    if ( ( tempPolicy.getPolicyCondition()
).getNoConditionProperty() ) {
        pStream.print ( "* }    " );
    } // End of if

    // collect all the condition names in the
conditionElementVector
    else{
        Enumeration enumForPriorityConditionVector = ( (
tempPolicy.getPolicyCondition() ).getPriorityConditionVector()
).elements();
        while ( enumForPriorityConditionVector.hasMoreElements() )
        {
            OnePriority tempOnePriority = new OnePriority ();

```



```

        tempOnePriority = ( OnePriority )
enumForPriorityConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOnePriority.toString() );
    }

    Enumeration enumForHopCountConditionVector = ( (
tempPolicy.getPolicyCondition() ).getHopCountConditionVector()
).elements();
    while ( enumForHopCountConditionVector.hasMoreElements() )
    {
        OneHopCount tempOneHopCount = new OneHopCount ();
        tempOneHopCount = ( OneHopCount )
enumForHopCountConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneHopCount.toString() );
    }

    Enumeration enumForTimeConditionVector = ( (
tempPolicy.getPolicyCondition() ).getTimeConditionVector()
).elements();
    while ( enumForTimeConditionVector.hasMoreElements() ) {
        OneTime tempOneTime = new OneTime ();
        tempOneTime = ( OneTime )
enumForTimeConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneTime.toString() );
    }

    Enumeration enumForSrcIPAddressConditionVector = ( (
tempPolicy.getPolicyCondition() ).getSrcIPAddressConditionVector()
).elements();
    while (
enumForSrcIPAddressConditionVector.hasMoreElements() ) {
        OneSrcIPAddress tempOneSrcIPAddress = new
OneSrcIPAddress ();
        tempOneSrcIPAddress = ( OneSrcIPAddress )
enumForSrcIPAddressConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneSrcIPAddress.toString() );
    }

    Enumeration enumForBWConditionVector = ( (
tempPolicy.getPolicyCondition() ).getBWConditionVector() ).elements();
    while ( enumForBWConditionVector.hasMoreElements() ) {
        OneBW tempOneBW = new OneBW ();
        tempOneBW = ( OneBW )
enumForBWConditionVector.nextElement ();
        conditionElementVector.addElement( tempOneBW.toString()
);
    }

    Enumeration enumForUserIDConditionVector = ( (
tempPolicy.getPolicyCondition() ).getUserIDConditionVector()
).elements();
    while ( enumForUserIDConditionVector.hasMoreElements() ) {

```

```

        OneUserID tempOneUserID = new OneUserID ();
        tempOneUserID = ( OneUserID )
enumForUserIDConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneUserID.toString() );
    }

    Enumeration enumForTypeConditionVector = ( (
tempPolicy.getPolicyCondition() ).getTypeConditionVector()
).elements();
    while ( enumForTypeConditionVector.hasMoreElements() ) {
        OneType tempOneType = new OneType ();
        tempOneType = ( OneType )
enumForTypeConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneType.toString() );
    }

    Enumeration enumForParameterConditionVector = ( (
tempPolicy.getPolicyCondition() ).getParameterConditionVector()
).elements();
    while ( enumForParameterConditionVector.hasMoreElements() )
    {
        OneParameter tempOneParameter = new OneParameter ();
        tempOneParameter = ( OneParameter )
enumForParameterConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneParameter.toString() );
    }

    // All 8 types of conditions are placed in the
conditionElementVector and they are written to file by the foolowing
Enumeration
    Enumeration enumForConditionElementVector =
conditionElementVector.elements();
    for ( int i = 0;
enumForConditionElementVector.hasMoreElements (); i++) {
        String tempString;
        tempString = ( String )
enumForConditionElementVector.nextElement ();
        if ( i >= 1 ) {
            pStream.print ( " && " );
        }
        pStream.print ( tempString );
    } // End of for

    pStream.print ( " }    " );

} // End of else (condition part does not have a wild card
character)

    pStream.print ( "{ " );

    if ( ( tempPolicy.getPolicyAction() ).getHasDenyAction() ) {
        pStream.println ("deny };");
    }

```

```

else {
    Vector actionElementVector = new Vector (1);
    if ( ( tempPolicy.getPolicyAction() ).getHasPermitAction()
) {
        actionElementVector.addElement("permit");
    }
    if ( ( tempPolicy.getPolicyAction() ).getPriorityValue() !=
0 ) {
        actionElementVector.addElement("priority" + " := " + (
tempPolicy.getPolicyAction() ).getPriorityValue() );
    }
    if ( ( tempPolicy.getPolicyAction() ).getHopCountValue() !=
0 ) {
        actionElementVector.addElement("hopCount" + " := " + (
tempPolicy.getPolicyAction() ).getHopCountValue() );
    }
    if ( ( tempPolicy.getPolicyAction() ).getAllocatedBWValue()
!= 0 ) {
        actionElementVector.addElement("allocated_bw" + " := " +
( tempPolicy.getPolicyAction() ).getAllocatedBWValue() + " MBPS");
    }
    if ( ( tempPolicy.getPolicyAction() ).getMaxLossRateValue()
!= 0 ) {
        actionElementVector.addElement("maxLossRate" + " := " +
( tempPolicy.getPolicyAction() ).getMaxLossRateValue() + " %");
    }
    if ( ( tempPolicy.getPolicyAction() ).getDelayBoundValue()
!= 0 ) {
        actionElementVector.addElement("delayBound" + " := " + (
tempPolicy.getPolicyAction() ).getDelayBoundValue() + " msec");
    }
    if ( ( tempPolicy.getPolicyAction()
).getSecurityLevelValue() != 0 ) {
        actionElementVector.addElement("securityLevel" + " := "
+ ( tempPolicy.getPolicyAction() ).getSecurityLevelValue() );
    }
}

Enumeration enumForActionElementVector =
actionElementVector.elements();
for ( int i = 0; enumForActionElementVector.hasMoreElements
()); i++) {
    String tempString;
    tempString = ( String )
enumForActionElementVector.nextElement ();
    if ( i >= 1 ) {
        pStream.print ( ", " );
    }
    pStream.print ( tempString );
} // End of for
pStream.println ( " };" );
} // End of else
pStream.println();
} // End of while ( enumForPolicies.hasMoreElements() )

} // End of method writePoliciesToFileMethod

```

```

/**
 * Method      : writePoliciesToPolicyFileMethod
 * Purpose     : This method writes the policies to file which
 * contains only policies in PPL format (Save Policies menu item)
 * Parameters  : None
 */

private void writePoliciesToPolicyFileMethod () {
    if ( !( policyVector.isEmpty() ) ) {
        poStream.println ("/*");
        poStream.println ("* Format of Policy Term: PolicyID UserID @
{paths} {targets} {conditions} {action_items};");
        poStream.println ("*/");
    } // End of if

    Enumeration enumForPolicies = policyVector.elements();
    while ( enumForPolicies.hasMoreElements() ) {

        Vector nodeLinkPathNames = new Vector (1); // Node, link ve
path names will be placed in it as Strings.
        Vector conditionElementVector = new Vector (1); // Condition
elements will be placed in it as Strings

        Policy tempPolicy = new Policy ();
        tempPolicy = ( Policy ) enumForPolicies.nextElement();

        poStream.print ( tempPolicy.getPolicyID() + "    " +
tempPolicy.getPolicyMakerID() + "    @    " + "{ " );

        Enumeration enumForPolicyNodeVector = (
tempPolicy.getNodesInPolicyPathVector() ).elements();
        while ( enumForPolicyNodeVector.hasMoreElements() ) {
            Node tempNode = new Node ();
            tempNode = ( Node ) enumForPolicyNodeVector.nextElement ();
            nodeLinkPathNames.addElement( tempNode.toString() );
        }

        Enumeration enumForPolicyLinkVector = (
tempPolicy.getLinksInPolicyPathVector() ).elements();
        while ( enumForPolicyLinkVector.hasMoreElements() ) {
            Link tempLink = new Link ();
            tempLink = ( Link ) enumForPolicyLinkVector.nextElement ();
            nodeLinkPathNames.addElement( tempLink.toString() );
        }

        Enumeration enumForPolicyPathVector = (
tempPolicy.getPathsInPolicyPathVector() ).elements();
        while ( enumForPolicyPathVector.hasMoreElements() ) {
            Path tempPath = new Path ();
            tempPath = ( Path ) enumForPolicyPathVector.nextElement ();
            nodeLinkPathNames.addElement( tempPath.toString() );
        }
    }
}

```

```

        Enumeration enumForNodeLinkPathNamesVector =
nodeLinkPathNames.elements();
        for ( int i = 0;
enumForNodeLinkPathNamesVector.hasMoreElements (); i++) {
            String tempString;
            tempString = ( String )
enumForNodeLinkPathNamesVector.nextElement ();
            if ( i >= 1 ) {
                poStream.print ( ", " );
            }

            poStream.print ( tempString ); // Write the nodes, links
and paths of the policy

        } // End of for
        poStream.print ( " }    { " );

        // write target element to file

        // If this policy has a wild card character in its target
element.
        if ( ( tempPolicy.getPolicyTarget() ).getTargetAllProperty() )
        {
            poStream.print ( "*" }    " );
        } // End of if
        else {
            Enumeration enumForTargetVector = ( (
tempPolicy.getPolicyTarget() ).getPolicyTargets() ).elements();
            for ( int i = 0; enumForTargetVector.hasMoreElements ();
i++) {
                String tempString;
                tempString = ( ( OneTarget )
enumForTargetVector.nextElement () ).outputMethod();
                if ( i >=1 ) {
                    poStream.print ( " || " );
                } // End of if

                poStream.print ( tempString ); // Write the target
elements of the policy

            } // End of for

            poStream.print ( " }    " );

        } // End of else

        // write condition element to file

        poStream.print ( "{ " );

        // If this policy has a wild card character in its condition
element.
        if ( ( tempPolicy.getPolicyCondition()
).getNoConditionProperty() ) {
            poStream.print ( "*" }    " );

```

```

        } // End of if

        // collect all the condition names in the
        conditionElementVector
        else{
            Enumeration enumForPriorityConditionVector = ( (
tempPolicy.getPolicyCondition() ).getPriorityConditionVector()
).elements();
            while ( enumForPriorityConditionVector.hasMoreElements() )
            {
                OnePriority tempOnePriority = new OnePriority ();
                tempOnePriority = ( OnePriority )
enumForPriorityConditionVector.nextElement ();
                conditionElementVector.addElement(
tempOnePriority.toString() );
            }

            Enumeration enumForHopCountConditionVector = ( (
tempPolicy.getPolicyCondition() ).getHopCountConditionVector()
).elements();
            while ( enumForHopCountConditionVector.hasMoreElements() )
            {
                OneHopCount tempOneHopCount = new OneHopCount ();
                tempOneHopCount = ( OneHopCount )
enumForHopCountConditionVector.nextElement ();
                conditionElementVector.addElement(
tempOneHopCount.toString() );
            }

            Enumeration enumForTimeConditionVector = ( (
tempPolicy.getPolicyCondition() ).getTimeConditionVector()
).elements();
            while ( enumForTimeConditionVector.hasMoreElements() ) {
                OneTime tempOneTime = new OneTime ();
                tempOneTime = ( OneTime )
enumForTimeConditionVector.nextElement ();
                conditionElementVector.addElement(
tempOneTime.toString() );
            }

            Enumeration enumForSrcIPAddressConditionVector = ( (
tempPolicy.getPolicyCondition() ).getSrcIPAddressConditionVector()
).elements();
            while (
enumForSrcIPAddressConditionVector.hasMoreElements() ) {
                OneSrcIPAddress tempOneSrcIPAddress = new
OneSrcIPAddress ();
                tempOneSrcIPAddress = ( OneSrcIPAddress )
enumForSrcIPAddressConditionVector.nextElement ();
                conditionElementVector.addElement(
tempOneSrcIPAddress.toString() );
            }

            Enumeration enumForBWConditionVector = ( (
tempPolicy.getPolicyCondition() ).getBWConditionVector() ).elements();
            while ( enumForBWConditionVector.hasMoreElements() ) {

```

```

        OneBW tempOneBW = new OneBW ();
        tempOneBW = ( OneBW )
enumForBWConditionVector.nextElement ();
        conditionElementVector.addElement( tempOneBW.toString()
);
    }

    Enumeration enumForUserIDConditionVector = ( (
tempPolicy.getPolicyCondition() ).getUserIDConditionVector()
).elements();
    while ( enumForUserIDConditionVector.hasMoreElements() ) {
        OneUserID tempOneUserID = new OneUserID ();
        tempOneUserID = ( OneUserID )
enumForUserIDConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneUserID.toString() );
    }

    Enumeration enumForTypeConditionVector = ( (
tempPolicy.getPolicyCondition() ).getTypeConditionVector()
).elements();
    while ( enumForTypeConditionVector.hasMoreElements() ) {
        OneType tempOneType = new OneType ();
        tempOneType = ( OneType )
enumForTypeConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneType.toString() );
    }

    Enumeration enumForParameterConditionVector = ( (
tempPolicy.getPolicyCondition() ).getParameterConditionVector()
).elements();
    while ( enumForParameterConditionVector.hasMoreElements() )
    {
        OneParameter tempOneParameter = new OneParameter ();
        tempOneParameter = ( OneParameter )
enumForParameterConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneParameter.toString() );
    }

    // All 8 types of conditions are placed in the
conditionElementVector and they are written to file by the foolowing
Enumeration
    Enumeration enumForConditionElementVector =
conditionElementVector.elements();
    for ( int i = 0;
enumForConditionElementVector.hasMoreElements (); i++) {
        String tempString;
        tempString = ( String )
enumForConditionElementVector.nextElement ();
        if ( i >= 1 ) {
            poStream.print ( " && " );
        }
        poStream.print ( tempString );
    } // End of for

```

```

        poStream.print ( " }    " );

    } // End of else (condition part does not have a wild card
character)

    poStream.print ( "{ " );

    if ( ( tempPolicy.getPolicyAction() ).getHasDenyAction() ) {
        poStream.println ("deny };");
    }
    else {
        Vector actionElementVector = new Vector (1);
        if ( ( tempPolicy.getPolicyAction() ).getHasPermitAction()
) {
            actionElementVector.addElement("permit");
        }
        if ( ( tempPolicy.getPolicyAction() ).getPriorityValue() !=
0 ) {
            actionElementVector.addElement("priority" + " := " + (
tempPolicy.getPolicyAction() ).getPriorityValue() );
        }
        if ( ( tempPolicy.getPolicyAction() ).getHopCountValue() !=
0 ) {
            actionElementVector.addElement("hopCount" + " := " + (
tempPolicy.getPolicyAction() ).getHopCountValue() );
        }
        if ( ( tempPolicy.getPolicyAction() ).getAllocatedBWValue()
!= 0 ) {
            actionElementVector.addElement("allocated_bw" + " := " +
( tempPolicy.getPolicyAction() ).getAllocatedBWValue() + " MBPS");
        }
        if ( ( tempPolicy.getPolicyAction() ).getMaxLossRateValue()
!= 0 ) {
            actionElementVector.addElement("maxLossRate" + " := " +
( tempPolicy.getPolicyAction() ).getMaxLossRateValue() + " %");
        }
        if ( ( tempPolicy.getPolicyAction() ).getDelayBoundValue()
!= 0 ) {
            actionElementVector.addElement("delayBound" + " := " + (
tempPolicy.getPolicyAction() ).getDelayBoundValue() + " msec");
        }
        if ( ( tempPolicy.getPolicyAction()
).getSecurityLevelValue() != 0 ) {
            actionElementVector.addElement("securityLevel" + " := "
+ ( tempPolicy.getPolicyAction() ).getSecurityLevelValue() );
        }

        Enumeration enumForActionElementVector =
actionElementVector.elements();
        for ( int i = 0; enumForActionElementVector.hasMoreElements
()); i++) {
            String tempString;
            tempString = ( String )
enumForActionElementVector.nextElement ();

```



```

        if ( i >= 1 ) {
            poStream.print ( ", " );
        }
        poStream.print ( tempString );
    } // End of for
    poStream.println ( " }; " );
} // End of else
poStream.println();
} // End of while ( enumForPolicies.hasMoreElements() )

} // End of method writePoliciesToPolicyFileMethod

/**
 * Method      : openFromFileMethod
 * Purpose     : This method imports a file into PPL Manager (Open
 * File menu item)
 * Parameters  : None
 */

public void openFromFileMethod () {
    // The following pops up a file chooser in the users home
directory
    JFileChooser fileChooser = new JFileChooser();

    fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
    // A file filter which allows only directories and .txt files is
instantiated and added to the JFileChooser
    txtFilter filter = new txtFilter ();
    fileChooser.addChoosableFileFilter(filter);

    int result = fileChooser.showDialog(null, "Open File");
    // If the user clicked Cancel button on dialog then do nothing
    if ( result == JFileChooser.CANCEL_OPTION ) {

    }
    else {
        File inputFile = fileChooser.getSelectedFile();
        if ( inputFile == null || (inputFile.getName()).equals("") ) {
            // Then do nothing
        }
        else {
            try {
                inputForNetwork = new ObjectInputStream ( new
FileInputStream (inputFile) );

                Vector tempCastVector = new Vector (1);
                tempCastVector = (Vector) inputForNetwork.readObject();

                Enumeration enum = tempCastVector.elements();

                // If the user tries to open a policy file instead of a
complete file with network
                // definition and policies in it then ClassCastException
is thrown
                Node tempNode = new Node ();

```

```

        tempNode = (Node) enum.nextElement();

        nodeVector = tempCastVector;
        linkVector = (Vector) inputForNetwork.readObject();
        pathVector = (Vector) inputForNetwork.readObject();
        classVector = (Vector) inputForNetwork.readObject();
        typeVector = (Vector) inputForNetwork.readObject();
        policyVector = (Vector) inputForNetwork.readObject();
        closeNetworkFile ();
    } // End of try

    catch ( FileNotFoundException fnfe ) {
        JOptionPane.showMessageDialog(null, "File not found
error", "Error", JOptionPane.ERROR_MESSAGE);
    }

    catch ( EOFException eof) {
        closeNetworkFile ();
    }

    catch ( IOException e ) {
        JOptionPane.showMessageDialog(null, "The file you
attempted to open is not in the correct PPL format", "Error",
JOptionPane.ERROR_MESSAGE);
    } // End of catch

    catch ( ClassNotFoundException cnfex ) {
        JOptionPane.showMessageDialog (null, "Unable to create
object", "Class not found", JOptionPane.ERROR_MESSAGE);
    }

    catch ( ClassCastException cce) {
        JOptionPane.showMessageDialog (null, "The file you
attempted to open is not in the correct PPL format", "Incorrect file
format", JOptionPane.ERROR_MESSAGE);
    }

    } // End of else

    } // End of else

} // End of method openFromFileMethod

/**
 * Method      : closeNetworkFile
 * Purpose     : This method closes inputForNetwork input stream
 * Parameters  : None
 */

private void closeNetworkFile () {
    try {
        inputForNetwork.close();
    }
    catch ( IOException ioe ){

```

```

        JOptionPane.showMessageDialog (null, "Error closing file",
"Error", JOptionPane.ERROR_MESSAGE);
    }
}

```

```

/**
 * Method      : openPoliciesFromFileMethod
 * Purpose     : This method import policies from a file into PPL
 * Manager (Import Policy menu item)
 * Parameters  : None
 */

public void openPoliciesFromFileMethod () {
    // The following pops up a file chooser in the users home
directory
    JFileChooser fileChooser = new JFileChooser();

    fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
    // A file filter which allows only directories and .txt files is
instantiated and added to the JFileChooser
    txtFilter filter = new txtFilter ();
    fileChooser.addChoosableFileFilter(filter);

    /* int result = fileChooser.showOpenDialog(null); */
    int result = fileChooser.showDialog(null, "Import Policies");
    // If the user clicked Cancel button on dialog then do nothing
    if ( result == JFileChooser.CANCEL_OPTION ) {

    }
    else {
        File inputFile = fileChooser.getSelectedFile();
        if ( inputFile == null || (inputFile.getName()).equals("") ) {
            // Then do nothing
        }
        else {
            try {
                inputForPolicy = new ObjectInputStream ( new
FileInputStream (inputFile) );

                Vector tempPolicyVector = new Vector (1);
                tempPolicyVector = (Vector) inputForPolicy.readObject();
                Enumeration enum = tempPolicyVector.elements();
                while ( enum.hasMoreElements() ) {
                    Policy tempPolicy = new Policy ();
                    tempPolicy = ( Policy ) enum.nextElement();
                    policyVector.addElement ( tempPolicy );
                }

                closePolicyFile ();
            }

```

```

        } // End of try

        catch ( FileNotFoundException fnfe ) {
            JOptionPane.showMessageDialog(null, "File not found
error", "Error", JOptionPane.ERROR_MESSAGE);
        }

        catch ( EOFException eof) {
            closeNetworkFile ();
        }

        catch ( IOException e ) {
            JOptionPane.showMessageDialog(null, "The file you
attempted to import is not a PPL policy file", "Incorrect File",
JOptionPane.ERROR_MESSAGE);
        } // End of catch

        catch ( ClassNotFoundException cnfex ) {
            JOptionPane.showMessageDialog (null, "Unable to create
object", "Class not found", JOptionPane.ERROR_MESSAGE);
        }

        catch ( ClassCastException cce) {
            JOptionPane.showMessageDialog (null, "The file you
attempted to import is not a PPL policy file", "Incorrect file format",
JOptionPane.ERROR_MESSAGE);
        }

    } // End of else

} // End of else

} // End of method openPoliciesFromFileMethod

/**
 * Method      : closePolicyFile
 * Purpose     : This method closes inputForPolicy input stream
 * Parameters  : None
 */

private void closePolicyFile () {
    try {
        inputForPolicy.close();
    }
    catch ( IOException ioe ){
        JOptionPane.showMessageDialog (null, "Error closing file",
"Error", JOptionPane.ERROR_MESSAGE);
    }
} // End of method closePolicyFile

/**
 * Method      : savePolicyMakersToFileMethod
 * Purpose     : This method saves policy makers to file
 * Parameters  : None

```

```

*/

public void savePolicyMakersToFileMethod () {

    try {
        output = new ObjectOutputStream ( new FileOutputStream (
userFile ) );

        Enumeration enum = policyMakerVector.elements();
        while ( enum.hasMoreElements() ) {
            PolicyMaker tempPolicyMaker = new PolicyMaker ("", "", 1);
            tempPolicyMaker = ( PolicyMaker ) enum.nextElement();
            if ( !( ( tempPolicyMaker.getLoginID()
).equalsIgnoreCase("administrator") ) ) {
                output.writeObject ( tempPolicyMaker );
            }
        }
        output.flush(); // Any data stored in memory is written to
file
        output.close(); // Close the output stream
    } // End of try

    catch (IOException e) {
        JOptionPane.showMessageDialog (null, "Error opening file",
"Error", JOptionPane.ERROR_MESSAGE);
    } // End of catch

} // End of method savePolicyMakersMethod

/**
 * Method      : readPolicyMakersFromFileMethod
 * Purpose      : This method reads policy makers from file
 * Parameters   : None
 */

private void readPolicyMakersFromFileMethod () {

    try {
        input = new ObjectInputStream ( new FileInputStream (userFile)
);
        while ( true ) {
            PolicyMaker tempPolicyMaker = new PolicyMaker ("", "", 1);
            tempPolicyMaker = ( PolicyMaker ) input.readObject();
            addPolicyMaker (tempPolicyMaker);
        }
    } // End of try

    catch ( FileNotFoundException fnfe ) {
        JOptionPane.showMessageDialog(null, "File not found error",
"Error", JOptionPane.ERROR_MESSAGE);
    }

    catch ( EOFException eof) {
        closeFile ();
    }
}

```

```

        catch ( IOException e ) {
            JOptionPane.showMessageDialog(null, "Error opening file",
"Error", JOptionPane.ERROR_MESSAGE);
        } // End of catch

        catch ( ClassNotFoundException cnfex ) {
            JOptionPane.showMessageDialog (null, "Unable to create
object", "Class not found", JOptionPane.ERROR_MESSAGE);
        }

    } // End of method readUsersFromFile method

/**
 * Method      : closeFile
 * Purpose     : This method closes the input stream which is used
 * to read policy makers from file
 * Parameters  : None
 */

private void closeFile () {
    try {
        input.close();
    }
    catch ( IOException ioe ){
        JOptionPane.showMessageDialog (null, "Error closing file",
"Error", JOptionPane.ERROR_MESSAGE);
    }
}

/**
 * Method      : launchGuiMenu
 * Purpose     : If the password and login ID of the policy creator
 * are correct, then this method is called from PasswordGui class
 * Parameters  : None
 */

public void launchGuiMenu () {
    myGui = new GuiMenu (this);
} // End of launchGuiMenu method

/**
 * Method      : setPolicyMakerID
 * Purpose     : This method sets the id of the policy creator after
 * the login process which will be used when the user creates a
 * policy and to decide whether to let the policy maker to create
 * new policy makers or not. Administrator will only be able to
 * create new policy makers.
 * Parameters  : String s
 */

public void setPolicyMakerID (String s) {
    policyMakerID = s;
} // End of method setPolicyMakerID

```

```

/**
 * Method      : getPolicyMakerID
 * Purpose     : This method returns the policy maker ID after the
 * login process
 * Parameters  : None
 */

public String getPolicyMakerID () {
    return policyMakerID;
}

/**
 * Method      : main
 * Purpose     : main method for DirectorClass class
 * Parameters  : String args[]
 */

public static void main( String args[] ) {
    DirectorClass pplDirector = new DirectorClass ();
} // End of main method

/**
 * Method      : getNodeVector
 * Purpose     : This method returns the current node vector.
 * Parameters  : None
 */

public Vector getNodeVector () {
    return nodeVector;
}

/**
 * Method      : getLinkVector
 * Purpose     : This method returns the current link vector.
 * Parameters  : None
 */

public Vector getLinkVector () {
    return linkVector;
}

/**
 * Method      : getPathVector
 * Purpose     : This method returns the current path vector.
 * Parameters  : None
 */

public Vector getPathVector () {
    return pathVector;
}

/**
 * Method      : getClassVector
 * Purpose     : This method returns the current class vector.
 * Parameters  : None

```

```

*/

public Vector getClassVector () {
    return classVector;
}


/**
 * Method      : getTypeVector
 * Purpose     : This method returns the current type vector.
 * Parameters  : None
 */

public Vector getTypeVector () {
    return typeVector;
}


/**
 * Method      : getPolicyVector
 * Purpose     : This method returns the current policy vector.
 * Parameters  : None
 */

public Vector getPolicyVector () {
    return policyVector;
}


/**
 * Method      : getPolicyMakerVector
 * Purpose     : This method returns the current policy maker
 *               vector.
 * Parameters  : None
 */

public Vector getPolicyMakerVector () {
    return policyMakerVector;
}


/**
 * Method      : addPolicyMaker
 * Purpose     : Method addPolicyMaker adds a new policy maker to
 *               the policyMakerVector. Validity check is made in
 *               policyMakerRedefinitionCheck method. The first policy maker
 *               "administrator" with the password "administrator" and the
 *               priority of "1" was created by the programmer. And only
 *               administrator will be able to add and delete policy makers.
 * Parameters  : PolicyMaker n
 */

public void addPolicyMaker ( PolicyMaker n ) {

```



```

        policyMakerVector.addElement(n);
    } // End addPolicyMaker method


/**
 * Method      : policyMakerRedefinitionCheck
 * Purpose     : policyMakerRedefinitionCheck method checks if the
 * login ID of a newly defined policy maker is the same as one of
 * the previously defined ones (redefinition check). If it is so,
 * this method returns false. If the newly defined policy maker is
 * valid, then this method returns true.
 * Parameters  : String o
 */

    public boolean policyMakerRedefinitionCheck ( String o ) {
        redefinedPolicyMaker = false; // Each time this method is called,
redefinedPolicyMaker flag is set to false.
        Enumeration enum = policyMakerVector.elements();

        while ( enum.hasMoreElements() ) {
            PolicyMaker tempPolicyMaker = new PolicyMaker ("", "", 1);
            tempPolicyMaker = ( PolicyMaker ) enum.nextElement();

            if ( ( tempPolicyMaker.getLoginID() ).equalsIgnoreCase( o ) )
            {
                redefinedPolicyMaker = true; // meaning that a policy maker
redefinition
                break; // After I detect a redefinition, it is not needed
to traverse the rest of the policy maker vector
            } // End of if

        } // End of while

        if ( redefinedPolicyMaker == true ) {
            return false; // meaning a policy maker redefinition
        }
        else {
            return true; // meaning a legal policy maker that can be
defined
        }

    } // End of policyMakerRedefinitionCheck method


/**
 * Method      : removePolicyMaker
 * Purpose     : Method removePolicyMaker removes the policy maker
 * sent as a parameter from the policy maker Vector. There is no

```

```

    * need to check the policy maker, because the Administrator will
    * select it from the combo box.
    * Parameters   : PolicyMaker n
    */

public void removePolicyMaker (PolicyMaker n) {
    policyMakerVector.removeElement ( n );
} // End removePolicyMaker Method

/**
 * Method       : addNode
 * Purpose      : Method addNode adds the new node to the end of
 * nodeVector. Validity check is made in nodeRedefinitionCheck
 * method.
 * Parameters   : Node n
 */

public void addNode ( Node n) {
    nodeVector.addElement(n);
} // End addNode method

/**
 * Method       : nodeRedefinitionCheck
 * Purpose      : nodeRedefinitionCheck method checks if the domain
 * and name of a newly defined node is the same as one of the
 * previously defined ones (redefinition check). If it is so, this
 * method returns false. If the newly defined node is valid, then
 * this method returns true.
 * Parameters   : Node o
 */

public boolean nodeRedefinitionCheck ( Node o ) {
    redefinedNode = false; // Each time this method is called,
    redefinedNode flag is set to false.
    Enumeration enum = nodeVector.elements();

    while ( enum.hasMoreElements() ) {
        Node tempNode = new Node ();
        tempNode = ( Node ) enum.nextElement();

        if ( ( ( tempNode.getDomain() ).equals( o.getDomain() ) ) &&
            ( ( tempNode.getName() ).equals( o.getName() ) ) ) {
            redefinedNode = true; // meaning that a node redefinition
            break; // After I detect a redefinition, it is not needed
            to traverse the rest of the node vector
        } // End of if

    } // End of while

    if ( redefinedNode == true ) {
        return false; // meaning a node redefinition
    }
}

```

```

        else {
            return true; // meaning a legal node that can be defined
        }
    } // End of nodeRedefinitionCheck method

/**
 * Method      : removeNode
 * Purpose     : Method removeNode removes the node sent as a
 * parameter from the node Vector. There is no need to check the
 * node, because the user selects it from the combo box.
 * Parameters  : Node n
 */

public void removeNode (Node n) {
    nodeVector.removeElement ( n );
} // End removeNode Method

/**
 * Method      : addLink
 * Purpose     : Method addLink adds the new link to the end of
 * linkVector. Validity check is made in linkRedefinitionCheck
 * method.
 * Parameters  : Link n
 */

public void addLink ( Link n ) {
    linkVector.addElement( n );
} // End of addLink method

/**
 * Method      : linkRedefinitionCheck
 * Purpose     : linkRedefinitionCheck method checks if the name of
 * a newly defined link is the same as one of the previously defined
 * ones (redefinition check). If it is so, this method returns
 * false. If the newly defined link is valid, then this method
 * returns true.
 * Parameters  : Link o
 */

public boolean linkRedefinitionCheck ( Link o ) {
    redefinedLink = false; // Each time this method is called,
    redefinedLink flag is set to false.
    Enumeration enum = linkVector.elements();

    while ( enum.hasMoreElements() ) {
        Node a = new Node ();
        Node b = new Node ();

```

```

        Link tempLink = new Link ("", a, b, 0, false, false, false,
false, false);
        tempLink = ( Link ) enum.nextElement();
        if ( ( tempLink.getName() ).equals( o.getName() ) ) {
            redefinedLink = true; // meaning that a link redefinition
            break; // After I detect a redefinition, it is not needed
to traverse the rest of the link vector
        } // End of if

    } // End of while

    if ( redefinedLink == true ) {
        return false; // meaning a link redefinition
    }
    else {
        return true; // meaning a legal link that can be defined
    }

} // End of linkRedefinitionCheck method

/**
 * Method      : removeLink
 * Purpose      : Method removeLink removes the link sent as a
 * parameter from the link Vector. There is no need to check the
 * link, because the user selects it from the combo box.
 * Parameters   : Link n
 */

public void removeLink (Link n) {
    linkVector.removeElement ( n );
} // End removeLink Method

/**
 * Method      : addPath
 * Purpose      : Method addPath adds the new path to the end of
 * pathVector. Validity check is made in pathRedefinitionCheck
 * method.
 * Parameters   : Path n
 */

public void addPath ( Path n ) {
    pathVector.addElement( n );
} // End of addPath method

/**
 * Method      : pathRedefinitionCheck
 * Purpose      : pathRedefinitionCheck method checks if the name of
 * a newly defined path is the same as one of the previously defined
 * ones (redefinition check). If it is so, this method returns
 * false. If the newly defined path is valid, then this method
 * returns true.
 * Parameters   : Path o
 */

```

```

    public boolean pathRedefinitionCheck ( Path o ) {
        redefinedPath = false; // Each time this method is called,
redefinedPath flag is set to false.
        Enumeration enum = pathVector.elements();

        while ( enum.hasMoreElements() ) {
            Vector d = new Vector ();
            Path tempPath = new Path ("", d, 0, false, false, false,
false, false);
            tempPath = ( Path ) enum.nextElement();

            if ( ( tempPath.getName() ).equals( o.getName() ) ) {
                redefinedPath = true; // meaning that a path redefinition
                break; // After I detect a redefinition, it is not needed
to traverse the rest of the path vector
            } // End of if

        } // End of while

        if ( redefinedPath == true ) {
            return false; // meaning a path redefinition
        }
        else {
            return true; // meaning a legal path that can be defined
        }

    } // End of pathRedefinitionCheck method

/**
 * Method      : removePath
 * Purpose      : Method removePath removes the path sent as a
 * parameter from the path Vector. There is no need to check the
 * path, because the user selects it from the combo box.
 * Parameters   : Path n
 */

public void removePath (Path n) {
    pathVector.removeElement ( n );
} // End removePath Method

/**
 * Method      : addClass
 * Purpose      : Method addClass adds the new class to the end of
 * classVector. Validity check is made in classRedefinitionCheck
 * method.
 * Parameters   : Class n
 */

public void addClass ( Class n ) {
    classVector.addElement( n );
} // End of addClass method

```

```

/**
 * Method      : classRedefinitionCheck
 * Purpose     : classRedefinitionCheck method checks if the name of
 * a newly defined class is the same as one of the previously
 * defined ones (redefinition check). If it is so, this method
 * returns false. If the newly defined class is valid, then this
 * method returns true.
 * Parameters  : Class o
 */

public boolean classRedefinitionCheck ( Class o ) {
    redefinedClass = false; // Each time this method is called,
    redefinedClass flag is set to false.
    Enumeration enum = classVector.elements();

    while ( enum.hasMoreElements() ) {
        Vector d = new Vector ();
        Class tempClass = new Class ("", d );
        tempClass = ( Class ) enum.nextElement();

        if ( ( tempClass.getName() ).equals( o.getName() ) ) {
            redefinedClass = true; // meaning that a class redefinition
            break; // After I detect a redefinition, it is not needed
to traverse the rest of the class vector
        } // End of if

    } // End of while

    if ( redefinedClass == true ) {
        return false; // meaning a class redefinition
    }
    else {
        return true; // meaning a legal class that can be defined
    }

} // End of classRedefinitionCheck method

/**
 * Method      : removeClass
 * Purpose     : Method removeClass removes the class sent as a
 * parameter from the class Vector. There is no need to check the
 * class, because the user selects it from the combo box.
 * Parameters  : Class n
 */

public void removeClass (Class n) {
    classVector.removeElement ( n );
} // End removeClass Method

/**
 * Method      : addType
 * Purpose     : Method addType adds the new type to the end of
 * typeVector. Validity check is made in typeRedefinitionCheck
 * method.

```

```

* Parameters   : Type n
*/

public void addType ( Type n ) {
    typeVector.addElement( n );
} // End of addType method


/**
* Method       : typeRedefinitionCheck
* Purpose      : typeRedefinitionCheck method checks if the name of
* a newly defined type is the same as one of the previously defined
* ones (redefinition check). If it is so, this method returns
* false. If the newly defined type is valid, then this method
* returns true.
* Parameters   : Type o
*/

public boolean typeRedefinitionCheck ( Type o ) {
    redefinedType = false; // Each time this method is called,
redefinedType flag is set to false.
    Enumeration enum = typeVector.elements();

    while ( enum.hasMoreElements() ) {
        Vector d = new Vector ();
        Type tempType = new Type ("", d );
        tempType = ( Type ) enum.nextElement();

        if ( ( tempType.getName() ).equals( o.getName() ) ) {
            redefinedType = true; // meaning that a type redefinition
            break; // After I detect a redefinition, it is not needed
to traverse the rest of the type vector
        } // End of if

    } // End of while

    if ( redefinedType == true ) {
        return false; // meaning a type redefinition
    }
    else {
        return true; // meaning a legal type that can be defined
    }

} // End of typeRedefinitionCheck method


/**
* Method       : removeType
* Purpose      : Method removeType removes the class sent as a
* parameter from the type Vector. There is no need to check the
* type, because the user selects it from the combo box.
* Parameters   : Type n
*/

```

```

public void removeType (Type n) {
    typeVector.removeElement ( n );
} // End removeType Method


/**
 * Method      : addPolicy
 * Purpose     : Method addPolicy adds the new policy to the end of
 * policyVector. Validity check is made in policyRedefinitionCheck
 * method.
 * Parameters  : Policy n
 */

public void addPolicy ( Policy n ) {
    policyVector.addElement( n );
} // End of addPolicy method


/**
 * Method      : policyRedefinitionCheck
 * Purpose     : policyRedefinitionCheck method checks if the name
 * of a newly defined policy is the same as one of the previously
 * defined ones (redefinition check). If it is so, this method
 * returns false. If the newly defined policy is valid, then this
 * method returns true.
 * Parameters  : String o
 */

public boolean policyRedefinitionCheck ( String o ) {
    redefinedPolicy = false; // Each time this method is called,
    redefinedPolicy flag is set to false.
    Enumeration enum = policyVector.elements();

    while ( enum.hasMoreElements() ) {
        Policy tempPolicy = new Policy ();
        tempPolicy = ( Policy ) enum.nextElement();

        if ( ( tempPolicy.getPolicyID() ).equals( o ) ) {
            redefinedPolicy = true; // meaning that a policy
            redefinition
            break; // After I detect a redefinition, it is not needed
            to traverse the rest of the policy vector
        } // End of if

    } // End of while

    if ( redefinedPolicy == true ) {
        return false; // meaning a policy redefinition
    }
    else {

```



```

        return true; // meaning a legal policy that can be defined
    }

} // End of policyRedefinitionCheck method


/**
 * Method      : removePolicy
 * Purpose     : Method removePolicy removes the policy sent as a
 * parameter from the policy Vector. There is no need to check the
 * policy, because the user selects it from the combo box.
 * Parameters  : Policy n
 */

public void removePolicy (Policy n) {
    policyVector.removeElement ( n );
} // End removePolicy Method

} // End of DirectorClass

```

```

/*****
File: GuiMenu.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/11/2001
Description: In this file, I will design the menu of the graphical user
interface tool kit for path based network policy language.
*****/

```

```

import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;
import java.io.*;

```

```

public class GuiMenu extends JFrame {
    private DirectorClass owner;
    private CreateNode createNodeObject;
    private DeleteNode deleteNodeObject;
    private CreateLink createLinkObject;
    private DeleteLink deleteLinkObject;
    private CreatePath createPathObject;
    private DeletePath deletePathObject;
    private CreateClass createClassObject;
    private DeleteClass deleteClassObject;
    private CreateType createTypeObject;
    private DeleteType deleteTypeObject;
    private CreatePolicyFirst createPolicyFirstObject;
    private CreatePolicy createPolicyObject;
    private DeletePolicy deletePolicyObject;
    private ModifyPolicy modifyPolicyObject;
    private CreatePolicyMaker createPolicyMakerObject;
    private DeletePolicyMaker deletePolicyMakerObject;
    private About aboutObject;
    private AboutFile aboutFileObject;

    // A JTextArea for displaying network elements, classes, types and
    policies
    private JTextArea viewJTextArea;
    private JTable viewTable;
    private JMenuBar bar;
    private JMenu fileMenu, policyMakerMenu, viewMenu, nodeMenu,
    linkMenu, pathMenu, classMenu, typeMenu, policyMenu, helpMenu;
    private JMenuItem newFileMenuItem, saveAsMenuItem, openFileMenuItem,
    savePoliciesMenuItem, openPoliciesMenuItem, compileFileMenuItem,
    exitMenuItem;
    private JMenuItem createPolicyMakerMenuItem,
    deletePolicyMakerMenuItem, savePolicyMakerMenuItem;
    private JMenuItem viewNodesMenuItem, viewLinksMenuItem,
    viewPathsMenuItem;
    private JMenuItem viewClassesMenuItem, viewTypesMenuItem,
    viewPoliciesMenuItem, viewPolicyMakersMenuItem;
    private JMenuItem createNodeMenuItem, deleteNodeMenuItem;
    private JMenuItem createLinkMenuItem, deleteLinkMenuItem;
    private JMenuItem createPathMenuItem, deletePathMenuItem;
    private JMenuItem createClassMenuItem, deleteClassMenuItem;

```

```

    private JMenuItem createTypeMenuItem, deleteTypeMenuItem;
    private JMenuItem createPolicyMenuItem, deletePolicyMenuItem,
modifyPolicyMenuItem, aboutMenuItem;
    private JMenuItem aboutFileMenuItem;

    private JRadioButtonMenuItem compileModeItems[];
    private ButtonGroup compileModeGroup;
    private int compileModeNumber = 0; // 0 = no argument, 1 = no wild
character, 2 = no implicit deny
    private Vector compilerOutputVector;

    private MenuItemHandler myMenuItemHandler; // inner class object
reference for event handling

/**
 * Method      : GuiMenu
 * Purpose     : Constructor for GuiMenu class
 * Parameters  : DirectorClass myDirector
 */

public GuiMenu ( DirectorClass myDirector ) {
    super( "PPL Manager" );
    this.owner = myDirector;
    Listener window = new Listener ();
    this.addWindowListener(window);
    setJMenuBar( menuBarBuilderMethod () ); // call the method that
will build the menu bar for the main GUI
    Container c = getContentPane ();

    viewJTextArea = new JTextArea (); // A JTextArea for displaying
network elements, classes, types and policies
    viewJTextArea.setEditable(false);
    viewJTextArea.setBackground(Color.lightGray);
    c.add ( new JScrollPane (viewJTextArea) ); // add a JScrollPane
containing the view JTextArea so that the user will scroll if the
contents in the JTextArea do not fit.

    setSize( 800, 700 );
//    setResizable (false);
    show();
} // End of constructor

/**
 * Method      : viewNodesMethod
 * Purpose     : This method displays the nodes and node parameters
 * created by the user in the text area of the main GUI.
 * Parameters  : None
 */

private void viewNodesMethod() {
    viewJTextArea.setText("");
    viewJTextArea.append("NODE AND NODE PARAMETER LIST:" + "\n\n");

    Enumeration enum = ( owner.getNodeVector() ).elements();

```

```

while ( enum.hasMoreElements() ) {
    Node tempNode = new Node ();
    tempNode = ( Node ) enum.nextElement();
    viewJTextArea.append ( "define " + "node " +
tempNode.toString() + ";" + "\n" );

    Vector parametersVector = new Vector (1);

    if ( tempNode.getHasBwParameter() ) {
        String s = "BW := " + tempNode.getBwValue() + " MBPS";
        parametersVector.addElement (s);
    }
    if ( tempNode.getHasDelayParameter() ) {
        parametersVector.addElement ( "delay ( )" );
    }
    if ( tempNode.getHasLossRateParameter() ) {
        parametersVector.addElement ( "loss_rate ( )" );
    }
    if ( tempNode.getHasJitterParameter() ) {
        parametersVector.addElement ( "jitter ( )" );
    }
    if ( tempNode.getHasUsedBwParameter() ) {
        parametersVector.addElement ( "used_bw ( )" );
    }

    if ( !( parametersVector.isEmpty() ) ) {
        viewJTextArea.append ( "define " + "node_param " +
tempNode.toString() + " { " );
    }

    Enumeration enumForParametersVector =
parametersVector.elements();
    for ( int i = 0; enumForParametersVector.hasMoreElements ();
i++) {
        String tempString;
        tempString = ( String ) enumForParametersVector.nextElement
();
        if ( i >= 1 ) {
            viewJTextArea.append ( ", " );
        }

        viewJTextArea.append ( tempString );

    } // End of for

    if ( !( parametersVector.isEmpty() ) ) {
        viewJTextArea.append( " };" + "\n\n");
    }
    else {
        viewJTextArea.append ( "\n" );
    }
} // End of while

} // End of viewNodesMethod
/**
 * Method          : viewLinksMethod

```

```

* Purpose      : This method displays the links and link parameters
* created by the user in the text area of the main GUI.
* Parameters   : None
*/

private void viewLinksMethod() {
    viewJTextArea.setText("");
    viewJTextArea.append("LINK AND LINK PARAMETER LIST:" + "\n\n");

    Enumeration enum = ( owner.getLinkVector() ).elements();
    while ( enum.hasMoreElements() ) {
        Node a = new Node ();
        Node b = new Node ();
        Link tempLink = new Link ("", a, b, 0, false, false, false,
false, false);
        tempLink = ( Link ) enum.nextElement();

        viewJTextArea.append ( "define " + "link " +
tempLink.getName() + " <" + ( tempLink.getNode1() ).toString() + ", " +
( tempLink.getNode2() ).toString() + ">;" + "\n");

        Vector parametersVector = new Vector (1);

        if ( tempLink.getHasBwParameter() ) {
            String s = "BW := " + tempLink.getBwValue() + " MBPS";
            parametersVector.addElement (s);
        }
        if ( tempLink.getHasDelayParameter() ) {
            parametersVector.addElement ( "delay ( )" );
        }
        if ( tempLink.getHasLossRateParameter() ) {
            parametersVector.addElement ( "loss_rate ( )" );
        }
        if ( tempLink.getHasJitterParameter() ) {
            parametersVector.addElement ( "jitter ( )" );
        }
        if ( tempLink.getHasUsedBwParameter() ) {
            parametersVector.addElement ( "used_bw ( )" );
        }

        if ( !( parametersVector.isEmpty() ) ) {
            viewJTextArea.append ( "define " + "link_param " +
tempLink.toString() + " { " );
        }

        Enumeration enumForParametersVector =
parametersVector.elements();
        for ( int i = 0; enumForParametersVector.hasMoreElements ();
i++) {
            String tempString;
            tempString = ( String ) enumForParametersVector.nextElement
();
            if ( i >= 1 ) {
                viewJTextArea.append ( ", " );
            }
        }
    }
}

```

```

        viewJTextArea.append ( tempString );

    } // End of for

    if ( !( parametersVector.isEmpty() ) ) {
        viewJTextArea.append( " }; " + "\n\n");
    }
    else {
        viewJTextArea.append ( "\n" );
    }

    } // End of while

} // End of viewLinksMethod

/**
 * Method      : viewPathsMethod
 * Purpose     : This method displays the paths and paths parameters
 * created by the user in the text area of the main GUI.
 * Parameters  : None
 */

private void viewPathsMethod() {
    viewJTextArea.setText("");
    viewJTextArea.append ( "PATH AND PATH PARAMETER LIST:" + "\n\n"
);

    Enumeration enum = ( owner.getPathVector() ).elements();
    while ( enum.hasMoreElements() ) {
        Path tempPath = new Path ();
        tempPath = ( Path ) enum.nextElement();

        viewJTextArea.append ( "define " + "path " +
tempPath.getName() + " {<" );

        // The names of the nodes in the path will be placed in
tempPathNodeVector
        Vector tempStringVector = new Vector (1);

        Enumeration enumString = ( tempPath.getPathNodesVector()
).elements();
        while ( enumString.hasMoreElements() ) {
            Node tempNode = new Node ();
            tempNode = ( Node ) enumString.nextElement();
            tempStringVector.addElement ( tempNode.toString() );
        }

        Enumeration enumPath = tempStringVector.elements();
        for ( int i = 0; enumPath.hasMoreElements (); i++) {
            String tempString;
            tempString = ( String ) enumPath.nextElement();
            if ( i >=1 ) {
                viewJTextArea.append ( ", " );
            } // End of if
            viewJTextArea.append ( tempString );

```

```

    } // End of for

    viewJTextArea.append ( ">;" + "\n");

    Vector parametersVector = new Vector (1);

    if ( tempPath.getHasBwParameter() ) {
        String s = "BW := " + tempPath.getBwValue() + " MBPS";
        parametersVector.addElement (s);
    }
    if ( tempPath.getHasDelayParameter() ) {
        parametersVector.addElement ( "delay ( )" );
    }
    if ( tempPath.getHasLossRateParameter() ) {
        parametersVector.addElement ( "loss_rate ( )" );
    }
    if ( tempPath.getHasJitterParameter() ) {
        parametersVector.addElement ( "jitter ( )" );
    }
    if ( tempPath.getHasUsedBwParameter() ) {
        parametersVector.addElement ( "used_bw ( )" );
    }

    if ( !( parametersVector.isEmpty() ) ) {
        viewJTextArea.append ( "define " + "path_param " +
tempPath.toString() + " { " );
    }

    Enumeration enumForParametersVector =
parametersVector.elements();
    for ( int i = 0; enumForParametersVector.hasMoreElements ();
i++) {
        String tempString;
        tempString = ( String ) enumForParametersVector.nextElement
();
        if ( i >= 1 ) {
            viewJTextArea.append ( ", " );
        }

        viewJTextArea.append ( tempString );

    } // End of for

    if ( !( parametersVector.isEmpty() ) ) {
        viewJTextArea.append( " };" + "\n\n");
    }
    else {
        viewJTextArea.append ( "\n" );
    }

    } // End of while

} // End of viewPathsMethod

/**
 * Method          : viewClassesMethod

```

```

* Purpose      : This method displays the classes created by the
* user in the text area of the main GUI.
* Parameters   : None
*/

private void viewClassesMethod() {
    viewJTextArea.setText("");
    viewJTextArea.append( "CLASS LIST:" + "\n\n" );

    Enumeration enum = ( owner.getClassVector() ).elements();
    while ( enum.hasMoreElements() ) {
        Class tempClass = new Class ();
        tempClass = ( Class ) enum.nextElement();

        viewJTextArea.append ( "define " + "class " +
tempClass.getName() + " {" );

        // Class member names will be placed in tempStringVector
        Vector tempStringVector = new Vector (1);

        Enumeration enumString = ( tempClass.getClassMembersVector()
).elements();

        for ( int i = 0; enumString.hasMoreElements (); i++) {
            String tempString;
            tempString = ( String ) enumString.nextElement();
            if ( i >=1 ) {
                viewJTextArea.append ( ", " );
            } // End of if

            viewJTextArea.append ( tempString );
        } // End of for

        viewJTextArea.append ( "};" + "\n" + "\n");

    } // End of while
} // End of viewClassesMethod

/**
* Method      : viewTypesMethod
* Purpose     : This method displays the types created by the user
* in the text area of the main GUI.
* Parameters  : None
*/

private void viewTypesMethod() {
    viewJTextArea.setText("");
    viewJTextArea.append( "TYPE LIST:" + "\n\n" );
    Enumeration enum = ( owner.getTypeVector() ).elements();
    while ( enum.hasMoreElements() ) {

        Type tempType = new Type ();
        tempType = ( Type ) enum.nextElement();

```



```

        viewJTextArea.append ( "define " + "type " +
tempType.getName() + " { " );

        // Type member names will be placed in tempStringVector
        Vector tempStringVector = new Vector (1);

        Enumeration enumString = ( tempType.getTypeMembersVector()
).elements();

        for ( int i = 0; enumString.hasMoreElements (); i++) {
            String tempString;
            tempString = ( String ) enumString.nextElement();
            if ( i >=1 ) {
                viewJTextArea.append ( ", " );
            } // End of if

            viewJTextArea.append ( tempString );
        } // End of for

        viewJTextArea.append ( "};" + "\n\n" );

    } // End of while

} // End of viewTypesMethod

/**
 * Method      : viewPolicyMakersMethod
 * Purpose     : This method displays the policy makers created by
 * the administrator.
 * Parameters  : None
 */

private void viewPolicyMakersMethod() {
    viewJTextArea.setText("");
    viewJTextArea.append("POLICY MAKER LIST:" + "\n\n");

    Enumeration enum = ( owner.getPolicyMakerVector() ).elements();
    while ( enum.hasMoreElements() ) {
        PolicyMaker tempPolicyMaker = new PolicyMaker ( "", "", 1);
        tempPolicyMaker = ( PolicyMaker ) enum.nextElement();
        viewJTextArea.append ( "define " + "policy_maker " +
tempPolicyMaker.getLoginID() + " (" +
tempPolicyMaker.getPolicyMakerPriority() + ");" + "\n\n");
    } // End of while

} // End of viewPolicyMakersMethod

/**

```

```

* Method      : viewPoliciesMethod
* Purpose     : This method displays the policies created by the
* policy maker.
* Parameters  : None
*/

private void viewPoliciesMethod() {

    viewJTextArea.setText("");
    viewJTextArea.append("POLICY LIST:" + "\n\n");

    Enumeration enumForPolicies = ( owner.getPolicyVector()
).elements();
    while ( enumForPolicies.hasMoreElements() ) {

        Vector nodeLinkPathNames = new Vector (1); // Node, link ve
path names will be placed in it as Strings.

        Vector conditionElementVector = new Vector (1); // Condition
elements will be placed in it as Strings

        Policy tempPolicy = new Policy ();
        tempPolicy = ( Policy ) enumForPolicies.nextElement();

        viewJTextArea.append ( tempPolicy.getPolicyID() + "    " +
tempPolicy.getPolicyMakerID() + "    @    " + "{ " );

        Enumeration enumForPolicyNodeVector = (
tempPolicy.getNodesInPolicyPathVector() ).elements();
        while ( enumForPolicyNodeVector.hasMoreElements() ) {
            Node tempNode = new Node ();
            tempNode = ( Node ) enumForPolicyNodeVector.nextElement ();
            nodeLinkPathNames.addElement( tempNode.toString() );
        }

        Enumeration enumForPolicyLinkVector = (
tempPolicy.getLinksInPolicyPathVector() ).elements();
        while ( enumForPolicyLinkVector.hasMoreElements() ) {
            Link tempLink = new Link ();
            tempLink = ( Link ) enumForPolicyLinkVector.nextElement ();
            nodeLinkPathNames.addElement( tempLink.toString() );
        }

        Enumeration enumForPolicyPathVector = (
tempPolicy.getPathsInPolicyPathVector() ).elements();
        while ( enumForPolicyPathVector.hasMoreElements() ) {
            Path tempPath = new Path ();
            tempPath = ( Path ) enumForPolicyPathVector.nextElement ();
            nodeLinkPathNames.addElement( tempPath.toString() );
        }

        Enumeration enumForNodeLinkPathNamesVector =
nodeLinkPathNames.elements();

```

```

        for ( int i = 0;
enumForNodeLinkPathNamesVector.hasMoreElements (); i++) {
            String tempString;
            tempString = ( String )
enumForNodeLinkPathNamesVector.nextElement ();
            if ( i >= 1 ) {
                viewJTextArea.append ( ", " );
            }

            viewJTextArea.append ( tempString );

        } // End of for
        viewJTextArea.append ( " }    { " );

        // write target element to JTextArea

        // If this policy has a wild card character in its target
element.
        if ( ( tempPolicy.getPolicyTarget() ).getTargetAllProperty() )
        {
            viewJTextArea.append ( "*" }    " );
        } // End of if
        else {
            Enumeration enumForTargetVector = ( (
tempPolicy.getPolicyTarget() ).getPolicyTargets() ).elements();
            for ( int i = 0; enumForTargetVector.hasMoreElements ();
i++) {
                String tempString;
                tempString = ( ( OneTarget )
enumForTargetVector.nextElement () ).outputMethod();
                if ( i >=1 ) {
                    viewJTextArea.append ( " || " );
                } // End of if

                viewJTextArea.append ( tempString );

            } // End of for

            viewJTextArea.append ( "}"    " );

        } // End of else

        // write condition element to JTextArea

        viewJTextArea.append ( "{ " );

        // If this policy has a wild card character in its condition
element.
        if ( ( tempPolicy.getPolicyCondition()
).getNoConditionProperty() ) {
            viewJTextArea.append ( "*" }    " );
        } // End of if

```

```

        else{
            Enumeration enumForPriorityConditionVector = ( (
tempPolicy.getPolicyCondition() ).getPriorityConditionVector()
).elements();
            while ( enumForPriorityConditionVector.hasMoreElements() )
            {
                OnePriority tempOnePriority = new OnePriority ();
                tempOnePriority = ( OnePriority )
enumForPriorityConditionVector.nextElement ();
                conditionElementVector.addElement(
tempOnePriority.toString() );
            }

            Enumeration enumForHopCountConditionVector = ( (
tempPolicy.getPolicyCondition() ).getHopCountConditionVector()
).elements();
            while ( enumForHopCountConditionVector.hasMoreElements() )
            {
                OneHopCount tempOneHopCount = new OneHopCount ();
                tempOneHopCount = ( OneHopCount )
enumForHopCountConditionVector.nextElement ();
                conditionElementVector.addElement(
tempOneHopCount.toString() );
            }

            Enumeration enumForTimeConditionVector = ( (
tempPolicy.getPolicyCondition() ).getTimeConditionVector()
).elements();
            while ( enumForTimeConditionVector.hasMoreElements() ) {
                OneTime tempOneTime = new OneTime ();
                tempOneTime = ( OneTime )
enumForTimeConditionVector.nextElement ();
                conditionElementVector.addElement(
tempOneTime.toString() );
            }

            Enumeration enumForSrcIPAddressConditionVector = ( (
tempPolicy.getPolicyCondition() ).getSrcIPAddressConditionVector()
).elements();
            while (
enumForSrcIPAddressConditionVector.hasMoreElements() ) {
                OneSrcIPAddress tempOneSrcIPAddress = new
OneSrcIPAddress ();
                tempOneSrcIPAddress = ( OneSrcIPAddress )
enumForSrcIPAddressConditionVector.nextElement ();
                conditionElementVector.addElement(
tempOneSrcIPAddress.toString() );
            }

            Enumeration enumForBWConditionVector = ( (
tempPolicy.getPolicyCondition() ).getBWConditionVector() ).elements();

            while ( enumForBWConditionVector.hasMoreElements() ) {
                OneBW tempOneBW = new OneBW ();
                tempOneBW = ( OneBW )
enumForBWConditionVector.nextElement ();

```

```

        conditionElementVector.addElement( tempOneBW.toString()
);
    }

    Enumeration enumForUserIDConditionVector = ( (
tempPolicy.getPolicyCondition() ).getUserIDConditionVector()
).elements();

    while ( enumForUserIDConditionVector.hasMoreElements() ) {
        OneUserID tempOneUserID = new OneUserID ();
        tempOneUserID = ( OneUserID )
enumForUserIDConditionVector.nextElement ();
        String temp = tempOneUserID.toString();
        conditionElementVector.addElement(
tempOneUserID.toString() );
    }

    Enumeration enumForTypeConditionVector = ( (
tempPolicy.getPolicyCondition() ).getTypeConditionVector()
).elements();
    while ( enumForTypeConditionVector.hasMoreElements() ) {
        OneType tempOneType = new OneType ();
        tempOneType = ( OneType )
enumForTypeConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneType.toString() );
    }

    Enumeration enumForParameterConditionVector = ( (
tempPolicy.getPolicyCondition() ).getParameterConditionVector()
).elements();
    while ( enumForParameterConditionVector.hasMoreElements() )
    {
        OneParameter tempOneParameter = new OneParameter ();
        tempOneParameter = ( OneParameter )
enumForParameterConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneParameter.toString() );
    }

    // All 8 types of conditions are placed in the
conditionElementVector and they are displayed in the JTextArea by the
foolowing Enumeration
    Enumeration enumForConditionElementVector =
conditionElementVector.elements();
    for ( int i = 0;
enumForConditionElementVector.hasMoreElements (); i++) {
        String tempString;
        tempString = ( String )
enumForConditionElementVector.nextElement ();
        if ( i >= 1 ) {
            viewJTextArea.append ( " && " );
        }
        viewJTextArea.append ( tempString );
    } // End of for

```

```

        viewJTextArea.append ( " }    " );

    } // End of else (condition part does not have a wild card
character)

    // write action element of policy to JTextArea
    viewJTextArea.append ( "{ " );

    if ( ( tempPolicy.getPolicyAction() ).getHasDenyAction() ) {
        viewJTextArea.append("deny "; " + "\n");
    }
    else {
        Vector actionElementVector = new Vector (1);
        if ( ( tempPolicy.getPolicyAction() ).getHasPermitAction()
) {
            actionElementVector.addElement("permit");
        }
        if ( ( tempPolicy.getPolicyAction() ).getPriorityValue() !=
0 ) {
            actionElementVector.addElement("priority" + " := " + (
tempPolicy.getPolicyAction() ).getPriorityValue() );
        }
        if ( ( tempPolicy.getPolicyAction() ).getHopCountValue() !=
0 ) {
            actionElementVector.addElement("hopCount" + " := " + (
tempPolicy.getPolicyAction() ).getHopCountValue() );
        }
        if ( ( tempPolicy.getPolicyAction() ).getAllocatedBWValue()
!= 0 ) {
            actionElementVector.addElement("allocated_bw" + " := " +
( tempPolicy.getPolicyAction() ).getAllocatedBWValue() + " MBPS");
        }
        if ( ( tempPolicy.getPolicyAction() ).getMaxLossRateValue()
!= 0 ) {
            actionElementVector.addElement("maxLossRate" + " := " +
( tempPolicy.getPolicyAction() ).getMaxLossRateValue() + " %");
        }
        if ( ( tempPolicy.getPolicyAction() ).getDelayBoundValue()
!= 0 ) {
            actionElementVector.addElement("delayBound" + " := " + (
tempPolicy.getPolicyAction() ).getDelayBoundValue() + " msec");
        }
        if ( ( tempPolicy.getPolicyAction()
).getSecurityLevelValue() != 0 ) {
            actionElementVector.addElement("securityLevel" + " := "
+ ( tempPolicy.getPolicyAction() ).getSecurityLevelValue() );
        }
    }

    Enumeration enumForActionElementVector =
actionElementVector.elements();
    for ( int i = 0; enumForActionElementVector.hasMoreElements
()); i++) {
        String tempString;

```

```

        tempString = ( String )
enumForActionElementVector.nextElement ();
        if ( i >= 1 ) {
            viewJTextArea.append ( ", " );
        }
        viewJTextArea.append ( tempString );
    } // End of for
    viewJTextArea.append ( " };" + "\n" );
} // End of else
    viewJTextArea.append ("\n");
} // End of while ( enumForPolicies.hasMoreElements() )

} // End of viewPoliciesMethod

/**
 * Method      : viewCompilerOutputMethod
 * Purpose      : This method displays the output of the PPL compiler
 * (BASH Shell) in the text area of the main gui.
 * Parameters   : None
 */

private void viewCompilerOutputMethod () {
    viewJTextArea.setText("");

    Enumeration enum = compilerOutputVector.elements();
    while ( enum.hasMoreElements() ) {
        String tempString;
        tempString = ( String ) enum.nextElement ();
        viewJTextArea.append(tempString);
        viewJTextArea.append("\n");
    } // End of while
    compilerOutputVector.removeAllElements();
} // End of method viewCompilerOutputMethod

/**
 * Class      : Listener
 * Purpose     : This inner class inherits empty definitions of all
 * methods that WindowListener contains. So, I can use any method I
 * want (windowClosing in this case) without implementing all
 * methods.
 */

class Listener extends WindowAdapter {
    public void windowClosing (WindowEvent e) {
        // Current list of policy makers is saved to file in case the
user forgets to save it
        owner.savePolicyMakersToFileMethod();
        System.exit(0);
    }
} // End of class Listener

```

```

/**
 * Method      : menuBarBuilderMethod
 * Purpose     : This method builds a JMenuBar for the GUI and
 * returns that menu bar.
 * Parameters  : None
 */

private JMenuBar menuBarBuilderMethod () {
    bar = new JMenuBar(); // create menubar

    fileMenu = new JMenu( "File" ); // create "File" menu
    fileMenu.setMnemonic( 'F' );
    bar.add(fileMenu); //Add "File" menu to the "bar" menu bar

    myMenuItemHandler = new MenuItemHandler (); // inner class
    object creation for event handling

    newFileMenuItem = new JMenuItem( "New..." );
    newFileMenuItem.setMnemonic( 'N' );
    newFileMenuItem.addActionListener ( myMenuItemHandler );
    fileMenu.add( newFileMenuItem );
    fileMenu.addSeparator();

    openFileMenuItem = new JMenuItem( "Open File..." );
    openFileMenuItem.setMnemonic( 'O' );
    openFileMenuItem.addActionListener ( myMenuItemHandler );
    fileMenu.add( openFileMenuItem );
    fileMenu.addSeparator();

    saveAsMenuItem = new JMenuItem ( "Save File...");
    saveAsMenuItem.setMnemonic( 'S' );
    saveAsMenuItem.addActionListener ( myMenuItemHandler );
    fileMenu.add( saveAsMenuItem );
    fileMenu.addSeparator();

    openPoliciesMenuItem = new JMenuItem ( "Import Policy...");
    openPoliciesMenuItem.setMnemonic( 'P' );
    openPoliciesMenuItem.addActionListener ( myMenuItemHandler );

    fileMenu.add ( openPoliciesMenuItem );
    fileMenu.addSeparator();

    savePoliciesMenuItem = new JMenuItem ( "Save Policy...");
    savePoliciesMenuItem.setMnemonic( 'e' );
    savePoliciesMenuItem.addActionListener ( myMenuItemHandler );
    fileMenu.add ( savePoliciesMenuItem );
    fileMenu.addSeparator();

    // create "Set Compile Mode" submenu
    String compileModes[] = { "No argument", "No wild card
character", "No implicit deny" };
    JMenu compileModeMenu = new JMenu( "Set Compile Mode" );
    compileModeMenu.setMnemonic( 'M' );

```



```

        compileModeItems = new JRadioButtonMenuItem [ compileModes.length
];

    // To ensure that only one of the menu items in the "Set Compile
Mode" submenu is selected at a time
    compileModeGroup = new ButtonGroup();

    // used to respond to selections from "Set Compile Mode" submenu
    ItemHandler itemHandler = new ItemHandler();

    for ( int i = 0; i < compileModes.length; i++ ) {
        //Create 3 JRadioButtonMenuItems
        compileModeItems[ i ] = new JRadioButtonMenuItem(
compileModes[ i ] );
        // Add 3 JRadioButtonMenuItems to "Set Compile Mode" submenu
        compileModeMenu.add( compileModeItems[ i ] );
        // Add 3 JRadioButtonMenuItems to button group
        compileModeGroup.add( compileModeItems[ i ] );
        // Registers the ActionListener for each JRadioButtonMenuItems
        compileModeItems[ i ].addActionListener( itemHandler );
    }

    compileModeItems[ 0 ].setSelected( true );
    fileMenu.add( compileModeMenu ); //Add "Set Compile Mode" submenu
to "Compile" menu
    fileMenu.addSeparator();

    compileFileMenuItem = new JMenuItem( "Compile" ); // create
"Compile" menu item
    compileFileMenuItem.setMnemonic( 'C' );
    compileFileMenuItem.addActionListener ( myMenuItemHandler );
    fileMenu.add( compileFileMenuItem );// Add "Compile" menu item to
"File" menu
    fileMenu.addSeparator();

    //create "Exit" menu item
    exitMenuItem = new JMenuItem( "Exit" );
    exitMenuItem.setMnemonic( 'x' );
    exitMenuItem.addActionListener( myMenuItemHandler );
    fileMenu.add( exitMenuItem ); //Add "Exit" menu item to "File"
menu

    // create policy maker menu
    policyMakerMenu = new JMenu ( "User" );
    policyMakerMenu.setMnemonic( 'U');
    policyMakerMenu.addActionListener ( myMenuItemHandler );

    // If the current policy maker is not administrator, then he will
not have the right to create, delete or
    // save policy makers.
    if ( !( ( owner.getPolicyMakerID() ).equalsIgnoreCase(
"administrator" ) ) ) {
        policyMakerMenu.setEnabled(false);
    }

```

```

        policyMakerMenu.setToolTipText("Only administrator can create,
delete or save policy makers (users)");
    }

    // create "Create Policy Maker" menu item
    createPolicyMakerMenuItem = new JMenuItem ("Create User...");
    createPolicyMakerMenuItem.setMnemonic('C');
    createPolicyMakerMenuItem.addActionListener(myMenuItemHandler);
    policyMakerMenu.add(createPolicyMakerMenuItem);
    policyMakerMenu.addSeparator();
    // create "Delete Policy Maker" menu item
    deletePolicyMakerMenuItem = new JMenuItem ("Delete User...");
    deletePolicyMakerMenuItem.setMnemonic('D');
    deletePolicyMakerMenuItem.addActionListener(myMenuItemHandler);
    policyMakerMenu.add(deletePolicyMakerMenuItem);
    policyMakerMenu.addSeparator();
    // create "Save Policy Maker" menu item
    savePolicyMakerMenuItem = new JMenuItem ("Save current users");
    savePolicyMakerMenuItem.setMnemonic('S');
    savePolicyMakerMenuItem.addActionListener(myMenuItemHandler);
    policyMakerMenu.add(savePolicyMakerMenuItem);

    bar.add ( policyMakerMenu );

    // create "View" menu
    viewMenu = new JMenu( "View" );
    viewMenu.setMnemonic( 'V' );

    //create "ViewPolicyMakers" menu item
    viewPolicyMakersMenuItem = new JMenuItem( "View Users" );
    viewPolicyMakersMenuItem.setMnemonic( 'U' );
    viewPolicyMakersMenuItem.addActionListener(myMenuItemHandler);
    // If the current policy maker is not administrator, then he will
not have the right to view policy makers.
    if ( !( ( owner.getPolicyMakerID() ).equalsIgnoreCase(
"administrator" ) ) ) {
        viewPolicyMakersMenuItem.setEnabled(false);
        viewPolicyMakersMenuItem.setToolTipText("Only administrator
can view policy makers (users)");
    }
    viewMenu.add( viewPolicyMakersMenuItem ); //Add "View policy
makers" menu item to "View" menu
    viewMenu.addSeparator();

    //create "ViewNodes" menu item
    viewNodesMenuItem = new JMenuItem( "View Nodes" );
    viewNodesMenuItem.setMnemonic( 'N' );
    viewNodesMenuItem.addActionListener(myMenuItemHandler);
    viewMenu.add( viewNodesMenuItem ); //Add "View nodes" menu item
to "View" menu
    viewMenu.addSeparator();

    //create "ViewLinks" menu item
    viewLinksMenuItem = new JMenuItem( "View Links" );
    viewLinksMenuItem.setMnemonic( 'L' );

```

```

        viewLinksMenuItem.addActionListener(myMenuItemHandler);
        viewMenu.add( viewLinksMenuItem ); //Add "View links" menu item
to "View" menu
        viewMenu.addSeparator();

        //create "ViewPaths" menu item
        viewPathsMenuItem = new JMenuItem( "View Paths" );
        viewPathsMenuItem.setMnemonic( 'P' );
        viewPathsMenuItem.addActionListener(myMenuItemHandler);
        viewMenu.add( viewPathsMenuItem ); //Add "View paths" menu item
to "View" menu
        viewMenu.addSeparator();

        //create "ViewClasses" menu item
        viewClassesMenuItem = new JMenuItem( "View Classes" );
        viewClassesMenuItem.setMnemonic( 'C' );
        viewClassesMenuItem.addActionListener(myMenuItemHandler);
        viewMenu.add( viewClassesMenuItem ); //Add "View classes" menu
item to "View" menu
        viewMenu.addSeparator();

        //create "ViewTypes" menu item
        viewTypesMenuItem = new JMenuItem( "View Types" );
        viewTypesMenuItem.setMnemonic( 'T' );
        viewTypesMenuItem.addActionListener(myMenuItemHandler);
        viewMenu.add( viewTypesMenuItem ); //Add "View types" menu item
to "View" menu
        viewMenu.addSeparator();

        //create "ViewPolicies" menu item
        viewPoliciesMenuItem = new JMenuItem( "View Policies" );
        viewPoliciesMenuItem.setMnemonic( 'o' );
        viewPoliciesMenuItem.addActionListener(myMenuItemHandler);
        viewMenu.add( viewPoliciesMenuItem ); //Add "View policies" menu
item to "View" menu

        bar.add(viewMenu); //Add "View" menu to "bar" menu bar

        // create "Node" menu
        nodeMenu = new JMenu( "Node" );
        nodeMenu.setMnemonic( 'N' );

        //create "Create node" menu item
        createNodeMenuItem = new JMenuItem( "Create node..." );
        createNodeMenuItem.setMnemonic( 'C' );
        createNodeMenuItem.addActionListener(myMenuItemHandler);
        nodeMenu.add( createNodeMenuItem ); //Add "Create node" menu item
to "Node" menu
        nodeMenu.addSeparator();

        //create "Delete node" menu item
        deleteNodeMenuItem = new JMenuItem( "Delete node..." );
        deleteNodeMenuItem.setMnemonic( 'D' );
        deleteNodeMenuItem.addActionListener(myMenuItemHandler);

```

```

        nodeMenu.add( deleteNodeMenuItem ); //Add "Delete node" menu item
to "Node" menu

        bar.add(nodeMenu); //Add "Node" menu to "bar" menu bar

        // create "Link" menu
        linkMenu = new JMenu( "Link" );
        linkMenu.setMnemonic( 'L' );

        //create "Create link" menu item
        createLinkMenuItem = new JMenuItem( "Create link..." );
        createLinkMenuItem.setMnemonic( 'C' );
        createLinkMenuItem.addActionListener(myMenuItemHandler);
        linkMenu.add( createLinkMenuItem ); //Add "Create link" menu item
to "Link" menu
        linkMenu.addSeparator();

        //create "Delete link" menu item
        deleteLinkMenuItem = new JMenuItem( "Delete link..." );
        deleteLinkMenuItem.setMnemonic( 'D' );
        deleteLinkMenuItem.addActionListener(myMenuItemHandler);
        linkMenu.add( deleteLinkMenuItem ); //Add "Delete link" menu item
to "Link" menu

        bar.add(linkMenu); //Add "Link" menu to "bar" menu bar

        // create "Path" menu
        pathMenu = new JMenu( "Path" );
        pathMenu.setMnemonic( 'P' );

        //create "Create path" menu item
        createPathMenuItem = new JMenuItem( "Create path..." );
        createPathMenuItem.setMnemonic( 'C' );
        createPathMenuItem.addActionListener(myMenuItemHandler);
        pathMenu.add( createPathMenuItem ); //Add "Create path" menu item
to "Path" menu
        pathMenu.addSeparator();

        //create "Delete path" menu item
        deletePathMenuItem = new JMenuItem( "Delete path..." );
        deletePathMenuItem.setMnemonic( 'D' );
        deletePathMenuItem.addActionListener(myMenuItemHandler);
        pathMenu.add( deletePathMenuItem ); //Add "Delete path" menu item
to "Path" menu

        bar.add(pathMenu); //Add "Path" menu to "bar" menu bar

        // create "Class" menu
        classMenu = new JMenu( "Class" );
        classMenu.setMnemonic( 'C' );

        //create "Create class" menu item
        createClassMenuItem = new JMenuItem( "Create class..." );

```

```

        createClassMenuItem.setMnemonic( 'C' );
        createClassMenuItem.addActionListener(myMenuItemHandler);
        classMenu.add( createClassMenuItem ); //Add "Create class" menu
item to "Class" menu
        classMenu.addSeparator();

        //create "Delete class" menu item
        deleteClassMenuItem = new JMenuItem( "Delete class..." );
        deleteClassMenuItem.setMnemonic( 'D' );
        deleteClassMenuItem.addActionListener(myMenuItemHandler);
        classMenu.add( deleteClassMenuItem ); //Add "Delete class" menu
item to "Class" menu

        bar.add(classMenu); //Add "Class" menu to "bar" menu bar

        // create "Type" menu
        typeMenu = new JMenu( "Type" );
        typeMenu.setMnemonic( 'T' );

        //create "Create type" menu item
        createTypeMenuItem = new JMenuItem( "Create type..." );
        createTypeMenuItem.setMnemonic( 'C' );
        createTypeMenuItem.addActionListener(myMenuItemHandler);
        typeMenu.add( createTypeMenuItem ); //Add "Create type" menu item
to "Type" menu
        typeMenu.addSeparator();

        //create "Delete type" menu item
        deleteTypeMenuItem = new JMenuItem( "Delete type..." );
        deleteTypeMenuItem.setMnemonic( 'D' );
        deleteTypeMenuItem.addActionListener(myMenuItemHandler);
        typeMenu.add( deleteTypeMenuItem ); //Add "Delete type" menu item
to "Type" menu

        bar.add (typeMenu); //Add "Type" menu to "bar" menu bar

        // create "Policy" menu
        policyMenu = new JMenu( "Policy" );
        policyMenu.setMnemonic( 'o' );

        //create "Create policy" menu item
        createPolicyMenuItem = new JMenuItem( "Create policy..." );
        createPolicyMenuItem.setMnemonic( 'C' );
        createPolicyMenuItem.addActionListener(myMenuItemHandler);
        policyMenu.add( createPolicyMenuItem ); //Add "Create policy"
menu item to "Policy" menu
        policyMenu.addSeparator();

        //create "Delete policy" menu item
        deletePolicyMenuItem = new JMenuItem( "Delete policy..." );
        deletePolicyMenuItem.setMnemonic( 'D' );
        deletePolicyMenuItem.addActionListener(myMenuItemHandler);
        policyMenu.add( deletePolicyMenuItem ); //Add "Delete policy"
menu item to "Policy" menu

```

```

    policyMenu.addSeparator();

    //create "Modify policy" menu item
    modifyPolicyMenuItem = new JMenuItem( "Modify policy..." );
    modifyPolicyMenuItem.setMnemonic( 'M' );
    modifyPolicyMenuItem.addActionListener(myMenuItemHandler);
    policyMenu.add( modifyPolicyMenuItem ); // Add "Modify policy"
menu item to "Policy" menu

    bar.add(policyMenu); //Add "Policy" menu to "bar" menu bar

    // create "Help" menu
    helpMenu = new JMenu( "Help" );
    helpMenu.setMnemonic( 'H' );

    //create "About" menu item
    aboutMenuItem = new JMenuItem( "About PPL Manager..." );
    aboutMenuItem.setMnemonic( 'A' );
    aboutMenuItem.addActionListener(myMenuItemHandler);
    helpMenu.add( aboutMenuItem ); //Add "About" menu item to "Help"
menu
    helpMenu.addSeparator();

    // create "About File Menu" menu item
    aboutFileMenuItem = new JMenuItem( "About File Menu..." );
    aboutFileMenuItem.setMnemonic( 'b' );
    aboutFileMenuItem.addActionListener(myMenuItemHandler);
    helpMenu.add(aboutFileMenuItem); // Add "About File Menu" menu
item to "Help" menu

    bar.add(helpMenu);

    return bar;

} //End of menuBarBuilderMethod

```

```

/**
 * Class      : ItemHandler
 * Purpose    : Inner class for set compile mode event handling. If
 * the user selects no argument mode, integer 0, if he selects no
 * wild character mode, integer 1, if he selects no implicit deny
 * mode, integer 2 will be passed to the compileFile method of the
 * DirectorClass.
 */

```

```

class ItemHandler implements ActionListener {
    public void actionPerformed( ActionEvent e ) {
        if ( compileModeItems[ 0 ].isSelected() ) {
            compileModeNumber = 0;
        } // End of if
        if ( compileModeItems[ 1 ].isSelected() ) {
            compileModeNumber = 1;
        } // End of if
    }
}

```

```

        if ( compileModeItems[ 2 ].isSelected() ) {
            compileModeNumber = 2;
        } // End of if

    } // End of public void actionPerformed((ActionEvent e) method

} // End of class ItemHandler implements ActionListener

/**
 * Method      : createNewFileMethod
 * Purpose     : This method creates a new PPL policy file ("New"
 * menu item under "File" menu.
 * Parameters  : None
 */

private void createNewFileMethod () {
    int answer = 0;
    answer = JOptionPane.showConfirmDialog ( GuiMenu.this, "Do you
want to save current network definiton and policies to file?", "Save
Confirmation", JOptionPane.YES_NO_CANCEL_OPTION );
    if ( answer == JOptionPane.CANCEL_OPTION ) {
        // Then do nothing
    } // End of if
    if (answer == JOptionPane.NO_OPTION ) {
        clearTextAreaAndVectorsMethod ();
    } // End of if
    if (answer == JOptionPane.YES_OPTION ) {
        owner.saveToFileMethod ();
        if ( owner.cancelOptionSelectedInSaveFile == true ) {
            /*("New" menu item - Yes option - Cancel option) then the text area of
the GUI will not be cleared and the network element and topology
vectors will not be emptied. */
        } // End of if
        else {
            /*("New" menu item - Yes option - Cancel option) then clear the text
area and empty the vectors after the user saves them to file. */
            clearTextAreaAndVectorsMethod ();
        } // End of else
    } // End of if
} // End of method createNewFileMethod

/**
 * Method      : clearTextAreaAndVectorsMethod
 * Purpose     : This method clears the vectors holding the network
 * definition and policies and the text area where all network
 * elements and policies are presented to the user, when new menu
 * item is confirmed.
 * Parameters  : None
 */

private void clearTextAreaAndVectorsMethod () {
    viewJTextArea.setText("");
    ( owner.getClassVector() ).removeAllElements();
    ( owner.getLinkVector() ).removeAllElements();
    ( owner.getNodeVector() ).removeAllElements();

```

```

        ( owner.getPathVector() ).removeAllElements();
        ( owner.getPolicyVector() ).removeAllElements();
        ( owner.getTypeVector() ).removeAllElements();
    } // End of method clearTextAreaAndVectorsMethod

```

```

/**
 * Class      : MenuItemHandler
 * Purpose    : Inner class for for menu item event handling.
 */

```

```

class MenuItemHandler implements ActionListener {
    public void actionPerformed (ActionEvent e ) {
        if ( e.getSource() == viewNodesMenuItem ) {
            viewNodesMethod ();
        }
        if ( e.getSource() == viewLinksMenuItem ) {
            viewLinksMethod ();
        }
        if ( e.getSource() == viewPathsMenuItem ) {
            viewPathsMethod ();
        }
        if ( e.getSource() == viewClassesMenuItem ) {
            viewClassesMethod ();
        }
        if ( e.getSource() == viewTypesMenuItem ) {
            viewTypesMethod ();
        }
        if ( e.getSource() == viewPoliciesMenuItem ) {
            viewPoliciesMethod ();
        }
        if ( e.getSource() == viewPolicyMakersMenuItem ) {
            viewPolicyMakersMethod ();
        }
        if ( e.getSource() == exitMenuItem ) {
            // Current list of policy makers is saved to file in case
the user forgets to save it
            owner.savePolicyMakersToFileMethod();
            System.exit(0);
        }
        if ( e.getSource() == createNodeMenuItem ) {
            createNodeObject = new CreateNode ( GuiMenu.this, owner);
        }
        if ( e.getSource() == deleteNodeMenuItem ) {
            deleteNodeObject = new DeleteNode ( GuiMenu.this, owner);
        }
        if ( e.getSource() == createLinkMenuItem ) {
            createLinkObject = new CreateLink (GuiMenu.this, owner);
        }
        if ( e.getSource() == deleteLinkMenuItem ) {
            deleteLinkObject = new DeleteLink ( GuiMenu.this, owner);
        }
        if ( e.getSource() == createPathMenuItem ) {
            createPathObject = new CreatePath (GuiMenu.this, owner);
        }
    }
}

```



```

        if ( e.getSource() == deletePathMenuItem ) {
            deletePathObject = new DeletePath (GuiMenu.this, owner);
        }
        if ( e.getSource() == createClassMenuItem ) {
            createClassObject = new CreateClass (GuiMenu.this, owner);
        }
        if ( e.getSource() == deleteClassMenuItem ) {
            deleteClassObject = new DeleteClass (GuiMenu.this, owner);
        }
        if ( e.getSource() == createTypeMenuItem ) {
            createTypeObject = new CreateType (GuiMenu.this, owner);
        }
        if ( e.getSource() == deleteTypeMenuItem ) {
            deleteTypeObject = new DeleteType (GuiMenu.this, owner);
        }
        if ( e.getSource() == createPolicyMenuItem ) {
            createPolicyObject = new CreatePolicy (GuiMenu.this,
owner);
        }
        if ( e.getSource() == modifyPolicyMenuItem ) {
            modifyPolicyObject = new ModifyPolicy (GuiMenu.this,
owner);
        }
        if ( e.getSource() == deletePolicyMenuItem ) {
            deletePolicyObject = new DeletePolicy (GuiMenu.this,
owner);
        }
        if ( e.getSource() == createPolicyMakerMenuItem ) {
            createPolicyMakerObject = new CreatePolicyMaker
(GuiMenu.this, owner);
        }
        if ( e.getSource() == deletePolicyMakerMenuItem ) {
            deletePolicyMakerObject = new DeletePolicyMaker
(GuiMenu.this, owner);
        }
        if ( e.getSource() == savePolicyMakerMenuItem ) {
            owner.savePolicyMakersToFileMethod ();
        }
        if ( e.getSource() == newFileMenuItem ) {
            createNewFileMethod ();
        }
        if ( e.getSource() == openFileMenuItem ) {
            owner.openFromFileMethod ();
        }
        if ( e.getSource() == saveAsMenuItem ) {
            owner.saveToFileMethod ();
        }
        if ( e.getSource() == openPoliciesMenuItem ) {
            owner.openPoliciesFromFileMethod ();
        }
        if ( e.getSource() == savePoliciesMenuItem ) {
            owner.savePoliciesToFileMethod ();
        }
        if ( e.getSource() == compileFileMenuItem ) {
            owner.compileFileMethod (compileModeNumber);

```

```

        compilerOutputVector = owner.displayCompilerOutputMethod
    );
        viewCompilerOutputMethod ();
    }
    if ( e.getSource() == aboutMenuItem ) {
        aboutObject = new About (GuiMenu.this);
    }
    if ( e.getSource() == aboutFileMenuItem ) {
        aboutFileObject = new AboutFile (GuiMenu.this);
    }

    } // End public void actionPerformed (ActionEvent e )

    } // End class MenuItemHandler implements ActionListener

} // End of public class GuiMenu extends JFrame

```

```

/*****
File: Link.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/22/2001
Description: In this file, network element link will be defined.
*****/

```

```

import java.io.Serializable;

```

```

public class Link extends Object implements Serializable {

```

```

    private String name;
    private Node node1;
    private Node node2;
    private float bwValue;
    private boolean hasBwParameter;
    private boolean hasDelayParameter;
    private boolean hasLossRateParameter;
    private boolean hasJitterParameter;
    private boolean hasUsedBwParameter;

```

```

    /**
     * Method      : Link
     * Purpose      : Default constructor for Link class
     * Parameters   : None
     */

```

```

    public Link () {
    } // End of default constructor

```

```

    /**
     * Method      : Link
     * Purpose      : Constructor for Link class
     * Parameters   : String n, Node no1, Node no2, float b, boolean bwp,
     *               : boolean dp, boolean lrp, boolean jp, boolean ubwp
     */

```

```

    public Link (String n, Node no1, Node no2, float b, boolean bwp,
boolean dp, boolean lrp, boolean jp, boolean ubwp) {
        setName ( n );
        setNode1 ( no1 );
        setNode2 ( no2 );
        setBwValue ( b );
        setHasBwParameter ( bwp );
        setHasDelayParameter ( dp );
        setHasLossRateParameter ( lrp );
        setHasJitterParameter ( jp );
        setHasUsedBwParameter ( ubwp );
    } // End of constructor

```

```

    /**
     * Method      : toString

```

```

    * Purpose      : to override toString () method of Object class
    * Parameters   : none
    */

public String toString () {
    return name;
}

/**
 * Method        : set and get methods of Link class
 * Purpose       : To get and set the data members of Link class
 * Parameters    : Each set method sets one data member of Link class.
 * get methods do not have parameters.
 */
public void setName (String n) {
    name = n;
}

public String getName () {
    return name;
}

public void setNode1 (Node no1) {
    node1 = no1;
}

public Node getNode1 () {
    return node1;
}

public void setNode2 (Node no2) {
    node2 = no2;
}

public Node getNode2 () {
    return node2;
}

public void setBwValue (float b) {
    bwValue = b;
}

public float getBwValue () {
    return bwValue;
}

public void setHasBwParameter (boolean bwp) {
    hasBwParameter = bwp;
}

public boolean getHasBwParameter () {
    return hasBwParameter;
}

public void setHasDelayParameter (boolean dp) {
    hasDelayParameter = dp;
}

```

```

    }

    public boolean getHasDelayParameter () {
        return hasDelayParameter;
    }

    public void setHasLossRateParameter (boolean lrp) {
        hasLossRateParameter = lrp;
    }

    public boolean getHasLossRateParameter () {
        return hasLossRateParameter;
    }

    public void setHasJitterParameter (boolean jp) {
        hasJitterParameter = jp;
    }

    public boolean getHasJitterParameter () {
        return hasJitterParameter;
    }

    public void setHasUsedBwParameter (boolean ubwp) {
        hasUsedBwParameter = ubwp;
    }

    public boolean getHasUsedBwParameter () {
        return hasUsedBwParameter;
    }

} // End class Link

```

```

/*****
File: ModifyPolicy.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/25/2001
Description: By means of this class, the policy maker will be able to
modify a policy.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class ModifyPolicy extends JDialog {
    private DirectorClass boss;
    private GuiMenu guiMenuForModifyPolicy;
    private Policy networkPolicyToBeModified = new Policy ();

    private JLabel modifyPolicyLabel, lineupLabel;
    private JComboBox policiesComboBox;
    private JPanel modifyPanel, buttonPanel;
    private JButton okButton, cancelButton;

    /**
     * Method      : ModifyPolicy
     * Purpose     : Constructor for ModifyPolicy class
     * Parameters  : GuiMenu gui, DirectorClass x
     */

    public ModifyPolicy ( GuiMenu gui, DirectorClass x ) {
        super ( gui, "Modify Policy", true ); // parent Frame is main
        GUI, and this JDialog is modal
        setLocation (115, 115);
        this.boss = x;
        this.guiMenuForModifyPolicy = gui;
        modifyPolicyGuiBuilderMethod (); // A method call for building
the GUI for "modify policy" JDialog
        setResizable (false);
        setSize(455,370);
        show();
    } // End of constructor

    /**
     * Method      : modifyPolicyGuiBuilderMethod
     * Purpose     : This method places the components of modify policy
     * window.
     * Parameters  : None
     */

    private void modifyPolicyGuiBuilderMethod () {
        Container c = getContentPane ();

        modifyPanel = new JPanel ();

        modifyPolicyLabel = new JLabel ( "                Select the policy
you want to modify:                ");

```

```

        modifyPanel.add ( modifyPolicyLabel );

        policiesComboBox = new JComboBox ( ( boss.getPolicyVector() ) );
        policiesComboBox.setMaximumRowCount(7);
        modifyPanel.add ( new JScrollPane (policiesComboBox) ); //
Provide a scroll bar if there are more than 7 policies in the combo
box.
        Dimension d = new Dimension (175, 25); // dimension object
(width, height) to be used as the dimension of policy combo box
        policiesComboBox.setPreferredSize(d); // set the size of the node
policy box so that it will not be too big or too small
        modifyPanel.add ( policiesComboBox );

        ButtonHandler handlerBut = new ButtonHandler ();

        c.add (modifyPanel, BorderLayout.CENTER);

        buttonPanel = new JPanel ();

        okButton = new JButton ("Modify");
        buttonPanel.add(okButton);
        okButton.addActionListener(handlerBut);
        if ( ( boss.getPolicyVector() ).isEmpty() ) {
            okButton.setEnabled(false);
            okButton.setToolTipText("No policy is defined for this network
yet.");
        }

        lineupLabel = new JLabel ("          ");
        buttonPanel.add(lineupLabel);

        cancelButton = new JButton ("Cancel");
        buttonPanel.add(cancelButton);
        cancelButton.addActionListener(handlerBut);

        c.add ( buttonPanel, BorderLayout.SOUTH );

    } // End of deletePolicyGuiBuilderMethod

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for event handling of OK and Cancel
 * buttons. The purpose is to call the window by which the policy
 * maker will modify the selected policy and pass the selected
 * policy to that window as a parameter and then destroy the
 * deletion window.
 */

private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == okButton ) {
            // Selected policy is assigned to a Policy object for
modification purpose
            networkPolicyToBeModified = ( Policy )
policiesComboBox.getSelectedItem();

```

```

        ModifyPolicy.this.setVisible(false);
        ModifyPolicySecond modifyPolicySecondObject = new
ModifyPolicySecond ( guiMenuForModifyPolicy, boss,
networkPolicyToBeModified );
    } // End of if

    if ( e.getSource() == cancelButton ) {
        ModifyPolicy.this.dispose();
    } // End of if

    } // End of method actionPerformed

} // End of class ButtonHandler

} // End of class ModifyPolicy

```



```

/*****
File: ModifyPolicy.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/25/2001
Description: In this file, I will design the main GUI which will be
used to modify a policy. The elements of the policy that was chosen
from the first modify policy window will automatically be defined in
the policy fields. When the policy maker changes elements of the policy
and clicks on OK button old policy that he wanted to modify will be
removed and the new version will be added to the policy vector in the
DirectorClass.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class ModifyPolicySecond extends JDialog {

    private DirectorClass boss;
    private GuiMenu guiMenuForPolicyModification;
    private Policy oldPolicy;
    private Policy newPolicy;

    private JButton policyIDButton, policyPathButton,
policyTargetButton, policyConditionButton;
    private JButton policyActionButton, okButton, cancelButton;
    private JLabel policyCreatorLabel, atLabel, lineupLabel,
lineupLabel2, lineupLabel3, lineupLabel4, infoLabel;
    private JTextArea viewJTextArea;

    /**
     * Method      : ModifyPolicySecond
     * Purpose      : Constructor for ModifyPolicySecond class
     * Parameters   : GuiMenu gui, DirectorClass x, Policy p
     */

    public ModifyPolicySecond ( GuiMenu gui, DirectorClass x, Policy p )
    {
        super ( gui, "Modify Policy", true ); // Parent Frame is main
        GUI, and this JDialog is modal
        setLocation (115, 115);
        this.guiMenuForPolicyModification = gui;
        this.boss = x;
        this.oldPolicy = p;
        Policy newPolicyTemp = new Policy ();
        newPolicyTemp.copy (p);
        this.newPolicy = newPolicyTemp;
        modifyPolicyGuiBuilderMethod ( );
        setResizable (false);
        setSize(780, 300);
        show();
    } // End of constructor

    /**
     * Method      : seven set methods

```

```

* Purpose      : Following seven set methods are used to set the
* Policy ID, Policy Path, Target Traffic, Path Conditions and
* Action Items elements of a policy.
* Parameters   : Appropriate parameter for setting each of the
* elements.
*/

public void setNetworkPolicyID ( String s ) {
    newPolicy.setPolicyID ( s );
}

public void setNetworkPolicyNodeVector ( Vector v ) {
    newPolicy.setNodesInPolicyPathVector ( v );
}

public void setNetworkPolicyLinkVector ( Vector v ) {
    newPolicy.setLinksInPolicyPathVector ( v );
}

public void setNetworkPolicyPathVector ( Vector v ) {
    newPolicy.setPathsInPolicyPathVector ( v );
}

public void setNetworkPolicyTarget ( Target t ) {
    newPolicy.setPolicyTarget ( t );
}

public void setNetworkPolicyCondition ( Condition c ) {
    newPolicy.setPolicyCondition ( c );
}

public void setNetworkPolicyAction ( Action a ) {
    newPolicy.setPolicyAction ( a );
}

/**
* Method      : modifyPolicyGuiBuilderMethod
* Purpose     : This method builds the GUI for the policy
* modification process.
* Parameters  : None
*/

private void modifyPolicyGuiBuilderMethod () {
    Container c = getContentPane ();
    c.setLayout( new FlowLayout () );

    infoLabel = new JLabel ("Move the mouse on any element of the
policy you wanted to modify to see information about that element.");
    c.add(infoLabel);

    lineupLabel = new JLabel ("          ");
    c.add(lineupLabel);

    ButtonHandler handlerBut = new ButtonHandler ();

```

```

MouseListener handlerMouse = new MouseHandler ();

policyIDButton = new JButton ("Policy ID");
policyIDButton.addActionListener(handlerBut);
policyIDButton.addMouseListener(handlerMouse);
policyIDButton.setBackground(Color.pink);
policyIDButton.setBorderPainted(true);
c.add ( policyIDButton );

policyCreatorLabel = new JLabel ( boss.getPolicyMakerID() );
policyCreatorLabel.addMouseListener(handlerMouse);
c.add ( policyCreatorLabel );

atLabel = new JLabel ( " @ " );
c.add(atLabel);

policyPathButton = new JButton ("{Policy Path}");
policyPathButton.addActionListener(handlerBut);
policyPathButton.addMouseListener(handlerMouse);
policyPathButton.setBackground(Color.cyan);
policyPathButton.setBorderPainted(true);
c.add ( policyPathButton );

policyTargetButton = new JButton ("{Policy Target}");
policyTargetButton.addActionListener(handlerBut);
policyTargetButton.addMouseListener(handlerMouse);
policyTargetButton.setBackground(Color.green);
policyTargetButton.setBorderPainted(true);
c.add ( policyTargetButton );

policyConditionButton = new JButton ("{Policy Condition}");
policyConditionButton.addActionListener(handlerBut);
policyConditionButton.addMouseListener(handlerMouse);
policyConditionButton.setBackground(Color.red);
policyConditionButton.setBorderPainted(true);
c.add ( policyConditionButton );

policyActionButton = new JButton ("{Policy Action}");
policyActionButton.addActionListener(handlerBut);
policyActionButton.addMouseListener(handlerMouse);
policyActionButton.setBackground(Color.orange);
policyActionButton.setBorderPainted(true);
c.add ( policyActionButton );

lineupLabel2 = new JLabel ( " " );
c.add(lineupLabel2);

viewJTextArea = new JTextArea (5, 62);
viewJTextArea.setBackground(Color.lightGray);
viewJTextArea.setEditable(false);
viewJTextArea.setLineWrap(true);
c.add ( new JScrollPane (viewJTextArea) );

lineupLabel3 = new JLabel ( " " );
c.add(lineupLabel3);

```

```

        okButton = new JButton ( "   OK   " );
        okButton.addActionListener(handlerBut);
        c.add ( okButton );

        lineupLabel4 = new JLabel ( "                                " );
        c.add(lineupLabel4);

        cancelButton = new JButton ("Cancel");
        cancelButton.addActionListener(handlerBut);
        c.add ( cancelButton );

    } // End of method modifyPolicyGuiBuilderMethod

/**
 * Class          : MouseHandler
 * Purpose        : When the mouse is moved on a policy element,
 *                  information about that element will be displayed to the user by
 *                  means of this class.
 */

private class MouseHandler extends MouseAdapter {

    public void mouseEntered ( MouseEvent e ) {
        if ( e.getSource() == policyIDButton ) {
            viewJTextArea.setText ( "Policy ID is:  " +
newPolicy.getPolicyID() );
        } // End of if ( e.getSource() == policyIDButton )
        if ( e.getSource() == policyCreatorLabel ) {
            viewJTextArea.setText ( "This is login name of policy
creator. It is automatically assigned after login process." );
        }
        if ( e.getSource() == policyPathButton ) {
            pathDisplayMethod ();
        }
        if ( e.getSource() == policyTargetButton ) {
            targetDisplayMethod ();
        }
        if ( e.getSource() == policyConditionButton ) {
            conditionDisplayMethod ();
        }
        if ( e.getSource() == policyActionButton ) {
            actionDisplayMethod ();
        }
    }

    } // End of method mouseEntered

    public void mouseExited ( MouseEvent e ) {
        viewJTextArea.setText("");
    } // End of method mouseExited

    private void pathDisplayMethod () {

```

```

viewJTextArea.append ( "Policy path is: " + "{ " );
Vector nodeLinkPathNames = new Vector (1);

Enumeration enumForPolicyNodeVector = (
newPolicy.getNodesInPolicyPathVector() ).elements();
while ( enumForPolicyNodeVector.hasMoreElements() ) {
    Node tempNode = new Node ();
    tempNode = ( Node ) enumForPolicyNodeVector.nextElement ();
    nodeLinkPathNames.addElement( tempNode.toString() );
}

Enumeration enumForPolicyLinkVector = (
newPolicy.getLinksInPolicyPathVector() ).elements();
while ( enumForPolicyLinkVector.hasMoreElements() ) {
    Link tempLink = new Link ();
    tempLink = ( Link ) enumForPolicyLinkVector.nextElement ();
    nodeLinkPathNames.addElement( tempLink.toString() );
}

Enumeration enumForPolicyPathVector = (
newPolicy.getPathsInPolicyPathVector() ).elements();
while ( enumForPolicyPathVector.hasMoreElements() ) {
    Path tempPath = new Path ();
    tempPath = ( Path ) enumForPolicyPathVector.nextElement ();
    nodeLinkPathNames.addElement( tempPath.toString() );
}

Enumeration enumForNodeLinkPathNamesVector =
nodeLinkPathNames.elements();
for ( int i = 0;
enumForNodeLinkPathNamesVector.hasMoreElements (); i++) {
    String tempString;
    tempString = ( String )
enumForNodeLinkPathNamesVector.nextElement ();
    if ( i >= 1 ) {
        viewJTextArea.append ( ", " );
    }

    viewJTextArea.append ( tempString );

} // End of for
viewJTextArea.append ( " }" );

} // End of method pathDisplayMethod

private void targetDisplayMethod () {
    viewJTextArea.append ( "Policy target is: " + "{ " );
    // If this policy has a wild card character in its target
element.
    if ( ( newPolicy.getPolicyTarget() ).getTargetAllProperty() )
{
        viewJTextArea.append ( "*" );
    } // End of if
    else {

```

```

        Enumeration enumForTargetVector = ( (
newPolicy.getPolicyTarget() ).getPolicyTargets() ).elements();
        for ( int i = 0; enumForTargetVector.hasMoreElements ();
i++) {
            String tempString;
            tempString = ( ( OneTarget )
enumForTargetVector.nextElement () ).outputMethod();
            if ( i >=1 ) {
                viewJTextArea.append ( " || " );
            } // End of if

            viewJTextArea.append ( tempString );

        } // End of for

        viewJTextArea.append ( " }" );

    } // End of else

} // End of method targetDisplayMethod

private void conditionDisplayMethod () {
    Vector conditionElementVector = new Vector (1); // Condition
elements will be placed in it as Strings
    viewJTextArea.append ( "Policy condition is: " + "{ " );
    // If this policy has a wild card character in its condition
element.
    if ( ( newPolicy.getPolicyCondition()
).getNoConditionProperty() ) {
        viewJTextArea.append ( "*" );
    } // End of if

    else{
        Enumeration enumForPriorityConditionVector = ( (
newPolicy.getPolicyCondition() ).getPriorityConditionVector()
).elements();
        while ( enumForPriorityConditionVector.hasMoreElements() )
        {
            OnePriority tempOnePriority = new OnePriority ();
            tempOnePriority = ( OnePriority )
enumForPriorityConditionVector.nextElement ();
            conditionElementVector.addElement(
tempOnePriority.toString() );
        }

        Enumeration enumForHopCountConditionVector = ( (
newPolicy.getPolicyCondition() ).getHopCountConditionVector()
).elements();
        while ( enumForHopCountConditionVector.hasMoreElements() )
        {
            OneHopCount tempOneHopCount = new OneHopCount ();
            tempOneHopCount = ( OneHopCount )
enumForHopCountConditionVector.nextElement ();
            conditionElementVector.addElement(
tempOneHopCount.toString() );

```

```

    }

    Enumeration enumForTimeConditionVector = ( (
newPolicy.getPolicyCondition() ).getTimeConditionVector() ).elements();
    while ( enumForTimeConditionVector.hasMoreElements() ) {
        OneTime tempOneTime = new OneTime ();
        tempOneTime = ( OneTime )
enumForTimeConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneTime.toString() );
    }

    Enumeration enumForSrcIPAddressConditionVector = ( (
newPolicy.getPolicyCondition() ).getSrcIPAddressConditionVector()
).elements();
    while (
enumForSrcIPAddressConditionVector.hasMoreElements() ) {
        OneSrcIPAddress tempOneSrcIPAddress = new
OneSrcIPAddress ();
        tempOneSrcIPAddress = ( OneSrcIPAddress )
enumForSrcIPAddressConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneSrcIPAddress.toString() );
    }

    Enumeration enumForBWConditionVector = ( (
newPolicy.getPolicyCondition() ).getBWConditionVector() ).elements();

    while ( enumForBWConditionVector.hasMoreElements() ) {
        OneBW tempOneBW = new OneBW ();
        tempOneBW = ( OneBW )
enumForBWConditionVector.nextElement ();
        conditionElementVector.addElement( tempOneBW.toString()
);
    }

    Enumeration enumForUserIDConditionVector = ( (
newPolicy.getPolicyCondition() ).getUserIDConditionVector()
).elements();

    while ( enumForUserIDConditionVector.hasMoreElements() ) {
        OneUserID tempOneUserID = new OneUserID ();
        tempOneUserID = ( OneUserID )
enumForUserIDConditionVector.nextElement ();
        String temp = tempOneUserID.toString();
        conditionElementVector.addElement(
tempOneUserID.toString() );
    }

    Enumeration enumForTypeConditionVector = ( (
newPolicy.getPolicyCondition() ).getTypeConditionVector() ).elements();
    while ( enumForTypeConditionVector.hasMoreElements() ) {
        OneType tempOneType = new OneType ();
        tempOneType = ( OneType )
enumForTypeConditionVector.nextElement ();

```

```

        conditionElementVector.addElement(
tempOneType.toString() );
    }

    Enumeration enumForParameterConditionVector = ( (
newPolicy.getPolicyCondition() ).getParameterConditionVector()
).elements();
    while ( enumForParameterConditionVector.hasMoreElements() )
    {
        OneParameter tempOneParameter = new OneParameter ();
        tempOneParameter = ( OneParameter )
enumForParameterConditionVector.nextElement ();
        conditionElementVector.addElement(
tempOneParameter.toString() );
    }

    // All 8 types of conditions are placed in the
conditionElementVector and they are displayed in the JTextArea by the
foolowing Enumeration
    Enumeration enumForConditionElementVector =
conditionElementVector.elements();
    for ( int i = 0;
enumForConditionElementVector.hasMoreElements (); i++) {
        String tempString;
        tempString = ( String )
enumForConditionElementVector.nextElement ();
        if ( i >= 1 ) {
            viewJTextArea.append ( " && " );
        }
        viewJTextArea.append ( tempString );
    } // End of for

    viewJTextArea.append ( " }" );

    } // End of else (condition part does not have a wild card
character)

    } // End of method conditionDisplayMethod

private void actionDisplayMethod () {
    viewJTextArea.append ( "Policy action is: " + "{ " );

    if ( ( newPolicy.getPolicyAction() ).getHasDenyAction() ) {
        viewJTextArea.append("deny }; " + "\n");
    }
    else {
        Vector actionElementVector = new Vector (1);
        if ( ( newPolicy.getPolicyAction() ).getHasPermitAction() )
        {
            actionElementVector.addElement("permit");
        }
        if ( ( newPolicy.getPolicyAction() ).getPriorityValue() !=
0 ) {
            actionElementVector.addElement("priority" + " := " + (
newPolicy.getPolicyAction() ).getPriorityValue() );

```



```

        }
        if ( ( newPolicy.getPolicyAction() ).getHopCountValue() !=
0 ) {
            actionElementVector.addElement("hopCount" + " := " + (
newPolicy.getPolicyAction() ).getHopCountValue() );
        }
        if ( ( newPolicy.getPolicyAction() ).getAllocatedBWValue()
!= 0 ) {
            actionElementVector.addElement("allocated_bw" + " := " +
( newPolicy.getPolicyAction() ).getAllocatedBWValue() + " MBPS");
        }
        if ( ( newPolicy.getPolicyAction() ).getMaxLossRateValue()
!= 0 ) {
            actionElementVector.addElement("maxLossRate" + " := " +
( newPolicy.getPolicyAction() ).getMaxLossRateValue() + " %");
        }
        if ( ( newPolicy.getPolicyAction() ).getDelayBoundValue()
!= 0 ) {
            actionElementVector.addElement("delayBound" + " := " + (
newPolicy.getPolicyAction() ).getDelayBoundValue() + " msec");
        }
        if ( ( newPolicy.getPolicyAction()
).getSecurityLevelValue() != 0 ) {
            actionElementVector.addElement("securityLevel" + " := "
+ ( newPolicy.getPolicyAction() ).getSecurityLevelValue() );
        }
    }

    Enumeration enumForActionElementVector =
actionElementVector.elements();
    for ( int i = 0; enumForActionElementVector.hasMoreElements
()); i++) {
        String tempString;
        tempString = ( String )
enumForActionElementVector.nextElement ();
        if ( i >= 1 ) {
            viewJTextArea.append ( ", " );
        }
        viewJTextArea.append ( tempString );
    } // End of for
    viewJTextArea.append ( " };" );
} // End of else

} // End of method actionDisplayMethod

} // End of class MouseHandler

```

```

/**
 * Class      : ButtonHandler

```

```

* Purpose      : This class creates different windows for the
* modification of each of the element of a policy when the button,
* on which the name of the element is written, is clicked. Policy
* ID, Policy Path, Target Traffic, Path Conditions and Action Items
* are 5 elements of a policy in PPL.
*/
private class ButtonHandler implements ActionListener {
    public void actionPerformed ( ActionEvent e) {
        if ( e.getSource() == policyIDButton ) {
            CreatePolicyID createPolicyIDObject = new CreatePolicyID (
boss, ModifyPolicySecond.this);
        }
        if ( e.getSource() == policyPathButton ) {
            CreatePolicyFirst createPolicyFirstObject = new
CreatePolicyFirst ( boss, ModifyPolicySecond.this);
        }
        if ( e.getSource() == policyTargetButton ) {
            CreatePolicySecond createPolicySecondObject = new
CreatePolicySecond ( boss, ModifyPolicySecond.this);
        }
        if ( e.getSource() == policyConditionButton ) {
            CreatePolicyThird createPolicyThirdObject = new
CreatePolicyThird ( guiMenuForPolicyModification, boss,
ModifyPolicySecond.this);
        }
        if ( e.getSource() == policyActionButton ) {
            CreatePolicyFourth createPolicyFourthObject = new
CreatePolicyFourth ( boss, ModifyPolicySecond.this);
        }
        if ( e.getSource() == okButton ) {
            boss.removePolicy(oldPolicy);
            boss.addPolicy (newPolicy);
            ModifyPolicySecond.this.dispose();
        } // End of if ( e.getSource() == okButton )
        if ( e.getSource() == cancelButton ) {
            ModifyPolicySecond.this.dispose();
        } // End of if

    } // End of method actionPerformed

} // End of class ButtonHandler

} // End of class ModifyPolicySecond

```

```

/*****
File: Node.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/12/2001
Description: In this file, network element node will be defined.
*****/

```

```
import java.io.Serializable;
```

```

public class Node extends Object implements Serializable {
    private String domain;
    private String name;
    private float bwValue;
    private boolean hasBwParameter;
    private boolean hasDelayParameter;
    private boolean hasLossRateParameter;
    private boolean hasJitterParameter;
    private boolean hasUsedBwParameter;

    /**
     * Method      : Node
     * Purpose      : default costructor for Node class, calls the
     * costructor with argument
     * Parameters    : None
     */

    public Node () {
        this ( "", "", 0, false, false, false, false, false );
    }

    /**
     * Method      : Node
     * Purpose      : costructor with arguements for Node class
     * Parameters    : String d, String n, float b, boolean bwp, boolean
     * dp, boolean lrp, boolean jp, boolean ubwp
     */

    public Node (String d, String n, float b, boolean bwp, boolean dp,
boolean lrp, boolean jp, boolean ubwp) {
        setDomain ( d );
        setName ( n );
        setBwValue ( b );
        setHasBwParameter ( bwp );
        setHasDelayParameter ( dp );
        setHasLossRateParameter ( lrp );
        setHasJitterParameter ( jp );
        setHasUsedBwParameter ( ubwp );
    }

    /**
     * Method      : toString

```

```

    * Purpose      : to override toString () method of Object class
    * Parameters   : none
    */

public String toString () {
    if ( name.equals("") ) {
        return "";
    }
    else {
        return name + "_" + domain;
    }
}

/**
 * Method        : equals
 * Purpose       : equals method of object class is overridden to
 * compare Node objects
 * Parameters    : Object o
 */

public boolean equals ( Object o ) {
    Node op = ( Node ) o;
    if ( name.equals ( op.name ) && domain.equals ( op.domain ) ) {
        return true;
    }
    else {
        return false;
    }
} // End of method equals

/**
 * Method        : set and get methods of Node class
 * Purpose       : To get and set the data members of Node class
 * Parameters    : Each set method sets one data member of Node class.
 * get methods do not have parameters.
 */

public void setDomain (String d) {
    domain = d;
}

public String getDomain () {
    return domain;
}

public void setName (String n) {
    name = n;
}

public String getName () {
    return name;
}

public void setBwValue (float b) {
    bwValue = b;
}

```

```

    }

    public float getBwValue () {
        return bwValue;
    }

    public void setHasBwParameter (boolean bwp) {
        hasBwParameter = bwp;
    }

    public boolean getHasBwParameter () {
        return hasBwParameter;
    }

    public void setHasDelayParameter (boolean dp) {
        hasDelayParameter = dp;
    }

    public boolean getHasDelayParameter () {
        return hasDelayParameter;
    }

    public void setHasLossRateParameter (boolean lrp) {
        hasLossRateParameter = lrp;
    }

    public boolean getHasLossRateParameter () {
        return hasLossRateParameter;
    }

    public void setHasJitterParameter (boolean jp) {
        hasJitterParameter = jp;
    }

    public boolean getHasJitterParameter () {
        return hasJitterParameter;
    }

    public void setHasUsedBwParameter (boolean ubwp) {
        hasUsedBwParameter = ubwp;
    }

    public boolean getHasUsedBwParameter () {
        return hasUsedBwParameter;
    }

} //End of node class

```

```

/*****
File: OneBW.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/14/2001
Description: In this file, BW values will be defined to be used in the
condition element of a policy. (BW <= 100 MBPS)
*****/
import java.io.Serializable;

public class OneBW implements Serializable {
    private boolean isGreaterThanOrEqualTo;
    private boolean isSmallerThanOrEqualTo;
    private float BWValue;

    /**
     * Method      : equals
     * Purpose      : equals method of object class is overridden to
     * compare OneBW objects
     * Parameters   : Object o
     */

    public boolean equals ( Object o ) {
        OneBW op = ( OneBW ) o;
        if ( (isGreaterThanOrEqualTo == op.isGreaterThanOrEqualTo) &&
            (isSmallerThanOrEqualTo==op.isSmallerThanOrEqualTo) &&
            (BWValue == op.BWValue) ) {
            return true;
        }
        else {
            return false;
        }
    } // End of method equals

    /**
     * Method      : toString
     * Purpose      : to override toString () method of Object class
     * Parameters   : none
     */

    public String toString () {
        if (isGreaterThanOrEqualTo == true) {
            return "BW" + " >= " + BWValue + " MBPS";
        }
        else {
            return "BW" + " <= " + BWValue + " MBPS";
        }
    } // End of method toString

    /**
     * Method      : copy

```

```

* Purpose      : This method creates a copy of the parameter OneBW
* object and puts it into the OneBW object by which this method is
* called.
* Parameters   : OneBW o
*/

public void copy (OneBW o) {
    BWValue = o.getBWValue ();
    isGreaterThanOrEqualTo = o.getIsGreaterThanOrEqualTo ();
    isSmallerThanOrEqualTo = o.getIsSmallerThanOrEqualTo ();
} // End of method copy

/**
* Method       : set and get methods of OneBW class
* Purpose      : To get and set the data members of OneBW class
* Parameters   : Each set method sets one data member of OneBW
* class. get methods do not have parameters.
*/

public void setBWValue (float s) {
    BWValue = s;
}

public float getBWValue () {
    return BWValue;
}

public void setIsGreaterThanOrEqualTo (boolean igtoe) {
    isGreaterThanOrEqualTo = igtoe;
}

public boolean getIsGreaterThanOrEqualTo () {
    return isGreaterThanOrEqualTo;
}

public void setIsSmallerThanOrEqualTo (boolean istoe) {
    isSmallerThanOrEqualTo = istoe;
}

public boolean getIsSmallerThanOrEqualTo () {
    return isSmallerThanOrEqualTo;
}

} // End of class OneBW

```

```

/*****
File: OneHopCount.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/12/2001
Description: In this file, hop count values will be defined to be used
in the condition element of a policy. (hopCount <= 20)
*****/
import java.io.Serializable;

public class OneHopCount implements Serializable {

    private boolean isGreaterThanOrEqualTo;
    private boolean isSmallerThanOrEqualTo;
    private int hopCountValue;

    /**
     * Method      : equals
     * Purpose      : equals method of object class is overridden to
     * compare OneHopCount objects
     * Parameters   : Object o
     */

    public boolean equals ( Object o ) {
        OneHopCount op = ( OneHopCount) o;
        if ( (isGreaterThanOrEqualTo == op.isGreaterThanOrEqualTo) &&
(isSmallerThanOrEqualTo==op.isSmallerThanOrEqualTo) && (hopCountValue ==
op.hopCountValue) ) {
            return true;
        }
        else {
            return false;
        }
    } // End of method equals

    /**
     * Method      : toString
     * Purpose      : to override toString () method of Object class
     * Parameters   : none
     */

    public String toString () {
        if (isGreaterThanOrEqualTo == true) {
            return "hopCount" + " >= " + hopCountValue;
        }
        else {
            return "hopCount" + " <= " + hopCountValue;
        }
    } // End of method toString

    /**
     * Method      : copy

```



```

* Purpose      : This method creates a copy of the parameter
* OneHopCount object and puts it into the OneHopCount object by
* which this method is called.
* Parameters   : OneHopCount o
*/

public void copy (OneHopCount o) {
    hopCountValue = o.getHopCountValue ();
    isGreaterThanOrEqualTo = o.getIsGreaterThanOrEqualTo ();
    isSmallerThanOrEqualTo = o.getIsSmallerThanOrEqualTo ();
} // End of method copy

/**
* Method       : set and get methods of OneHopCount class
* Purpose      : To get and set the data members of OneHopCount
* class Parameters : Each set method sets one data member of
* OneHopCount class. get methods do not have parameters.
*/

public void setHopCountValue (int s) {
    hopCountValue = s;
}

public int getHopCountValue () {
    return hopCountValue;
}

public void setIsGreaterThanOrEqualTo (boolean igtoe) {
    isGreaterThanOrEqualTo = igtoe;
}

public boolean getIsGreaterThanOrEqualTo () {
    return isGreaterThanOrEqualTo;
}

public void setIsSmallerThanOrEqualTo (boolean istoe) {
    isSmallerThanOrEqualTo = istoe;
}

public boolean getIsSmallerThanOrEqualTo () {
    return isSmallerThanOrEqualTo;
}

} // End of class OneHopCount

```

```

/*****
File: OneParameter.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/14/2001
Description: In this file, parameter values will be defined to be used
in the condition element of a policy. (delay () <= 10 msec)
*****/
import java.io.Serializable;

public class OneParameter implements Serializable {
    private String parameterName;
    private float parameterValue;
    private boolean isEqual;
    private boolean isNotEqual;
    private boolean isGreaterThanOrEqualTo;
    private boolean isSmallerThanOrEqualTo;
    private boolean isGreater;
    private boolean isSmaller;

    /**
     * Method      : equals
     * Purpose      : equals method of object class is overridden to
     * compare OneParameter objects
     * Parameters   : Object o
     */

    public boolean equals ( Object o ) {
        OneParameter op = ( OneParameter ) o;
        if ( (isEqual == op.isEqual) && (isNotEqual==op.isNotEqual) &&
            (isGreaterThanOrEqualTo == op.isGreaterThanOrEqualTo) &&
            (isSmallerThanOrEqualTo == op.isSmallerThanOrEqualTo) &&
            (isGreater == op.isGreater) && (isSmaller == op.isSmaller)
            &&
            (parameterName.equals(op.parameterName)) && (parameterValue
            == op.parameterValue) ) {
            return true;
        }
        else {
            return false;
        }
    } // End of method equals

    /**
     * Method      : toString
     * Purpose      : to override toString () method of Object class
     * Parameters   : none
     */

    public String toString () {
        if ( ( parameterName.equals("delay ()") ) && ( isEqual == true) )
        {
            return "delay ()" + " == " + parameterValue + " msec";
        }
    }
}

```

```

        else if ( ( parameterName.equals("delay ()") ) && ( isNotEqual ==
true ) ) {
            return "delay ()" + " != " + parameterValue + " msec";
        }
        else if ( ( parameterName.equals("delay ()") ) && (
isGreaterThanOrEqual == true ) ) {
            return "delay ()" + " >= " + parameterValue + " msec";
        }
        else if ( ( parameterName.equals("delay ()") ) && (
isSmallerThanOrEqual == true ) ) {
            return "delay ()" + " <= " + parameterValue + " msec";
        }
        else if ( ( parameterName.equals("delay ()") ) && ( isGreater ==
true ) ) {
            return "delay ()" + " > " + parameterValue + " msec";
        }
        else if ( ( parameterName.equals("delay ()") ) && ( isSmaller ==
true ) ) {
            return "delay ()" + " < " + parameterValue + " msec";
        }

        else if ( parameterName.equals("loss rate ()") && isEqual ) {
            return "loss_rate ()" + " == " + parameterValue + " %";
        }
        else if ( parameterName.equals("loss rate ()") && isNotEqual ) {
            return "loss_rate ()" + " != " + parameterValue + " %";
        }
        else if ( parameterName.equals("loss rate ()") &&
isGreaterThanOrEqual ) {
            return "loss_rate ()" + " >= " + parameterValue + " %";
        }
        else if ( parameterName.equals("loss rate ()") &&
isSmallerThanOrEqual ) {
            return "loss_rate ()" + " <= " + parameterValue + " %";
        }
        else if ( parameterName.equals("loss rate ()") && isGreater ) {
            return "loss_rate ()" + " > " + parameterValue + " %";
        }
        else if ( parameterName.equals("loss rate ()") && isSmaller ) {
            return "loss_rate ()" + " < " + parameterValue + " %";
        }

        else if ( parameterName.equals("jitter ()") && isEqual ) {
            return "jitter ()" + " == " + parameterValue + " msec";
        }
        else if ( parameterName.equals("jitter ()") && isNotEqual ) {
            return "jitter ()" + " != " + parameterValue + " msec";
        }
        else if ( parameterName.equals("jitter ()") &&
isGreaterThanOrEqual ) {
            return "jitter ()" + " >= " + parameterValue + " msec";
        }
        else if ( parameterName.equals("jitter ()") &&
isSmallerThanOrEqual ) {
            return "jitter ()" + " <= " + parameterValue + " msec";
        }
    }

```

```

        else if ( parameterName.equals("jitter ()") && isGreater ) {
            return "jitter ()" + " > " + parameterValue + " msec";
        }
        else if ( parameterName.equals("jitter ()") && isSmaller ) {
            return "jitter ()" + " < " + parameterValue + " msec";
        }

        else if ( parameterName.equals("used bw ()") && isEqual ) {
            return "used_bw ()" + " == " + parameterValue + " MBPS";
        }
        else if ( parameterName.equals("used bw ()") && isNotEqual ) {
            return "used_bw ()" + " != " + parameterValue + " MBPS";
        }
        else if ( parameterName.equals("used bw ()") &&
isGreaterThanOrEqualTo ) {
            return "used_bw ()" + " >= " + parameterValue + " MBPS";
        }
        else if ( parameterName.equals("used bw ()") &&
isSmallerThanOrEqualTo ) {
            return "used_bw ()" + " <= " + parameterValue + " MBPS";
        }
        else if ( parameterName.equals("used bw ()") && isGreater ) {
            return "used_bw ()" + " > " + parameterValue + " MBPS";
        }
        else {
            return "used_bw ()" + " < " + parameterValue + " MBPS";
        }
    } // End of method toString

/**
 * Method      : copy
 * Purpose      : This method creates a copy of the parameter
 * OneParameter object and puts it into the OneParameter object by
 * which this method is called.
 * Parameters   : OneParameter o
 */

public void copy (OneParameter o) {
    parameterName = o.getParameterName();
    parameterValue = o.getParameterValue();
    isGreaterThanOrEqualTo = o.getIsGreaterThanOrEqualTo ();
    isSmallerThanOrEqualTo = o.getIsSmallerThanOrEqualTo ();
    isGreater = o.getIsGreater();
    isSmaller = o.getIsSmaller();
    isEqual = o.getIsEqual();
    isNotEqual = o.getIsNotEqual();
} // End of method copy

```

/**

```

* Method      : set and get methods of OneParameter class
* Purpose     : To get and set the data members of OneParameter
* class
* Parameters  : Each set method sets one data member of
* OneParameter class. get methods do not have parameters.
*/

public void setParameterName (String s) {
    parameterName = s;
}

public String getParameterName () {
    return parameterName;
}

public void setParameterValue (float f) {
    parameterValue = f;
}

public float getParameterValue () {
    return parameterValue;
}

public void setIsEqual (boolean ie) {
    isEqual = ie;
}

public boolean getIsEqual () {
    return isEqual;
}

public void setIsNotEqual (boolean ine) {
    isNotEqual = ine;
}

public boolean getIsNotEqual () {
    return isNotEqual;
}

public void setIsGreaterThanOrEqual (boolean igtoe) {
    isGreaterThanOrEqual = igtoe;
}

public boolean getIsGreaterThanOrEqual () {
    return isGreaterThanOrEqual;
}

public void setIsSmallerThanOrEqual (boolean istoe) {
    isSmallerThanOrEqual = istoe;
}

public boolean getIsSmallerThanOrEqual () {
    return isSmallerThanOrEqual;
}

```

```
public void setIsGreater (boolean ig) {
    isGreater = ig;
}

public boolean getIsGreater () {
    return isGreater;
}

public void setIsSmaller (boolean is) {
    isSmaller = is;
}

public boolean getIsSmaller () {
    return isSmaller;
}

} // End of class OneParameter
```

```

/*****
File: OnePriority.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/12/2001
Description: In this file, priority values will be defined to be used
in the condition element of a policy. (priority <= 4)
*****/
import java.io.Serializable;

public class OnePriority implements Serializable {
    private boolean isGreaterThanOrEqualTo;
    private boolean isSmallerThanOrEqualTo;
    private int priorityValue;

    /**
     * Method      : toString
     * Purpose      : to override toString () method of Object class
     * Parameters   : none
     */

    public String toString () {
        if (isGreaterThanOrEqualTo == true) {
            return "priority" + " >= " + priorityValue;
        }
        else {
            return "priority" + " <= " + priorityValue;
        }
    } // End of method toString

    /**
     * Method      : equals
     * Purpose      : equals method of object class is overridden to
     * compare OnePriority objects
     * Parameters   : Object o
     */

    public boolean equals ( Object o ) {
        OnePriority op = ( OnePriority) o;
        if ( (isGreaterThanOrEqualTo == op.isGreaterThanOrEqualTo) &&
(isSmallerThanOrEqualTo==op.isSmallerThanOrEqualTo) && (priorityValue ==
op.priorityValue) ) {
            return true;
        }
        else {
            return false;
        }
    } // End of method equals

    /**
     * Method      : copy

```

```

* Purpose      : This method creates a copy of the parameter
* OnePriority object and puts it into the OnePriority object by
* which this method is called.
* Parameters   : OnePriority o
*/

public void copy (OnePriority o) {
    priorityValue = o.getPriorityValue ();
    isGreaterThanOrEqual = o.getIsGreaterThanOrEqual ();
    isSmallerThanOrEqual = o.getIsSmallerThanOrEqual ();
} // End of method copy

/**
* Method       : set and get methods of OnePriority class
* Purpose      : To get and set the data members of OnePriority
* class
* Parameters   : Each set method sets one data member of OnePriority
* class. get methods do not have parameters.
*/

public void setPriorityValue (int s) {
    priorityValue = s;
}

public int getPriorityValue () {
    return priorityValue;
}

public void setIsGreaterThanOrEqual (boolean igtoe) {
    isGreaterThanOrEqual = igtoe;
}

public boolean getIsGreaterThanOrEqual () {
    return isGreaterThanOrEqual;
}

public void setIsSmallerThanOrEqual (boolean istoe) {
    isSmallerThanOrEqual = istoe;
}

public boolean getIsSmallerThanOrEqual () {
    return isSmallerThanOrEqual;
}

} // End of class OnePriority

```



```

/*****
File: OneSrcIPAddress.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/13/2001
Description: In this file, source IP address values will be defined to
be used in the condition element of a policy. (srcIPAddress ==
131.120.*.* && srcIPAddress == 204.156.254.210)
*****/
import java.io.Serializable;

public class OneSrcIPAddress implements Serializable {

    private boolean isEqual;
    private boolean isNotEqual;

    private String quadDot1;
    private String quadDot2;
    private String quadDot3;
    private String quadDot4;

    /**
     * Method      : equals
     * Purpose      : equals method of object class is overridden to
     * compare OneSrcIPAddress objects
     * Parameters   : Object o
     */

    public boolean equals ( Object o ) {
        OneSrcIPAddress op = ( OneSrcIPAddress ) o;
        if ( (isEqual == op.isEqual) && (isNotEqual == op.isNotEqual) &&
(quadDot1.equals(op.quadDot1)) &&
        (quadDot2.equals(op.quadDot2)) &&
(quadDot3.equals(op.quadDot3)) && (quadDot4.equals(op.quadDot4)) ) {
            return true;
        }
        else {
            return false;
        }
    } // End of method equals

    /**
     * Method      : toString
     * Purpose      : to override toString () method of Object class
     * Parameters   : none
     */

    public String toString () {
        if (isEqual == true) {
            return "srcIPAddress" + " == " + quadDot1 + "." + quadDot2 +
"." + quadDot3 + "." + quadDot4;
        }
        else {
            return "srcIPAddress" + " != " + quadDot1 + "." + quadDot2 +
"." + quadDot3 + "." + quadDot4;
        }
    }
}

```

```

} // End of method toString

/**
 * Method      : copy
 * Purpose     : This method creates a copy of the parameter
 * OneSrcIPAddress object and puts it into the OneSrcIP object by
 * which this method is called.
 * Parameters  : OneSrcIPAddress o
 */

public void copy (OneSrcIPAddress o) {
    quadDot1 = o.getQuadDot1 ();
    quadDot2 = o.getQuadDot2 ();
    quadDot3 = o.getQuadDot3 ();
    quadDot4 = o.getQuadDot4 ();
    isEqual = o.getIsEqual ();
    isNotEqual = o.getIsNotEqual ();
} // End of method copy

/**
 * Method      : set and get methods of OneSrcIPAddress class
 * Purpose     : To get and set the data members of OneSrcIPAddress
 * class
 * Parameters  : Each set method sets one data member of
 * OneSrcIPAddress class. get methods do not have parameters.
 */

public void setQuadDot1 ( String s) {
    quadDot1 = s;
}

public String getQuadDot1 () {
    return quadDot1;
}

public void setQuadDot2 ( String s) {
    quadDot2 = s;
}

public String getQuadDot2 () {
    return quadDot2;
}

public void setQuadDot3 ( String s) {
    quadDot3 = s;
}

public String getQuadDot3 () {
    return quadDot3;
}

public void setQuadDot4 ( String s) {
    quadDot4 = s;
}

public String getQuadDot4 () {

```

```

        return quadDot4;
    }

    public void setIsEqual (boolean ie) {
        isEqual = ie;
    }

    public boolean getIsEqual () {
        return isEqual;
    }

    public void setIsNotEqual (boolean ine) {
        isNotEqual = ine;
    }

    public boolean getIsNotEqual () {
        return isNotEqual;
    }

} // End of class OneSrcIPAddress

```

```

/*****
File: OneTarget.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/06/2001
Description: In this file, target statements that will be put in the
target element of a policy will be defined.( traffic_type == {research}
)
*****/
import java.util.*;
import java.io.Serializable;

public class OneTarget implements Serializable {

    private String targetClassName;
    private String targetClassMemberName;
    private boolean isEqual;
    private boolean isNotEqual;

    /**
     * Method      : equals
     * Purpose      : equals method of object class is overridden to
     * compare OneTarget objects
     * Parameters   : Object o
     */

    public boolean equals ( Object o ) {
        OneTarget op = ( OneTarget) o;
        if ( (isEqual == op.isEqual) && (isNotEqual==op.isNotEqual) && (
            ( targetClassName.equals(op.targetClassName) ) && (
targetClassMemberName.equals(op.targetClassMemberName) ) ) ) {
            return true;
        }
        else {
            return false;
        }
    } // End of method equals

    /**
     * Method      : toString
     * Purpose      : to override toString () method of Object class
     * Parameters   : none
     */

    public String toString () {
        if ( isEqual == true ) {
            return targetClassName + " == " + targetClassMemberName;
        }
        else {
            return targetClassName + " != " + targetClassMemberName;
        }
    } // End of method toString

    /**
     * Method      : outputMethod

```

```

    * Purpose      : This method writes each Policy Target item to file
    * in the format that PPL compiler accepts.
    * Parameters   : None
    */

    public String outputMethod () {
        if ( isEqual == true ) {
            return targetClassName + " == " + "<" + targetClassMemberName
+ ">";
        }
        else {
            return targetClassName + " != " + "<" + targetClassMemberName
+ ">";
        }
    } // End of method outputMethod

/**
 * Method      : copy
 * Purpose     : This method creates a copy of the parameter
 * OneTarget object and puts it into the OneTarget object by which
 * this method is called.
 * Parameters  : OneTarget o
 */

    public void copy (OneTarget o) {
        targetClassName = o.getTargetClassName();
        targetClassMemberName = o.getTargetClassMemberName();
        isEqual = o.getIsEqual();
        isNotEqual = o.getIsNotEqual();
    }

/**
 * Method      : set and get methods of OneTarget class
 * Purpose     : To get and set the data members of OneTarget class
 * Parameters  : Each set method sets one data member of OneTarget
 * class. get methods do not have parameters.
 */

    public void setTargetClassName (String s) {
        targetClassName = s;
    }

    public String getTargetClassName () {
        return targetClassName;
    }

    public void setTargetClassMemberName (String s) {
        targetClassMemberName = s;
    }

    public String getTargetClassMemberName () {
        return targetClassMemberName;
    }

```

```
public void setIsEqual (boolean ie) {
    isEqual = ie;
}

public boolean getIsEqual () {
    return isEqual;
}

public void setIsNotEqual (boolean ine) {
    isNotEqual = ine;
}

public boolean getIsNotEqual () {
    return isNotEqual;
}

} // End of class OneTarget
```

```

/*****
File: OneTime.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/12/2001
Description: In this file, time values will be defined to be used in
the condition element of a policy. (time <= 1345)
*****/
import java.io.Serializable;

public class OneTime implements Serializable {

    private boolean isGreaterThanOrEqualTo;
    private boolean isSmallerThanOrEqualTo;
    private int timeValue;

    /**
     * Method      : equals
     * Purpose      : equals method of object class is overridden to
     * compare OneTime objects
     * Parameters   : Object o
     */

    public boolean equals ( Object o ) {
        OneTime op = ( OneTime) o;
        if ( (isGreaterThanOrEqualTo == op.isGreaterThanOrEqualTo) &&
(isSmallerThanOrEqualTo==op.isSmallerThanOrEqualTo) && (timeValue ==
op.timeValue) ) {
            return true;
        }
        else {
            return false;
        }
    } // End of method equals

    /**
     * Method      : toString
     * Purpose      : to override toString () method of Object class
     * Parameters   : none
     */

    public String toString () {
        if (isGreaterThanOrEqualTo == true) {
            return "time" + " >= " + timeValue;
        }
        else {
            return "time" + " <= " + timeValue;
        }
    } // End of method toString

    /**
     * Method      : copy

```

```

* Purpose      : This method creates a copy of the parameter OneTime
* object and puts it into the OneTime object by which this method
* is called.
* Parameters   : OneTime o
*/

public void copy (OneTime o) {
    timeValue = o.getTimeValue ();
    isGreaterThanOrEqualTo = o.getIsGreaterThanOrEqualTo ();
    isSmallerThanOrEqualTo = o.getIsSmallerThanOrEqualTo ();
} // End of method copy

/**
* Method       : set and get methods of OneTime class
* Purpose      : To get and set the data members of OneTime class
* Parameters   : Each set method sets one data member of OneTime
* class. get methods do not have parameters.
*/

public void setTimeValue (int s) {
    timeValue = s;
}

public int getTimeValue () {
    return timeValue;
}

public void setIsGreaterThanOrEqualTo (boolean igtoe) {
    isGreaterThanOrEqualTo = igtoe;
}

public boolean getIsGreaterThanOrEqualTo () {
    return isGreaterThanOrEqualTo;
}

public void setIsSmallerThanOrEqualTo (boolean istoe) {
    isSmallerThanOrEqualTo = istoe;
}

public boolean getIsSmallerThanOrEqualTo () {
    return isSmallerThanOrEqualTo;
}

} // End class OneTime

```



```

/*****
File: OneType.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/08/2001
Description: In this file, user defined types will be formatted to be
used in the condition element of
a policy. (day >= Monday)
*****/
import java.io.Serializable;

public class OneType implements Serializable {
    private String conditionTypeName;
    private String conditionTypeMemberName;
    private boolean isEqual;
    private boolean isNotEqual;
    private boolean isGreaterThanOrEqual;
    private boolean isSmallerThanOrEqual;

    /**
     * Method      : equals
     * Purpose      : equals method of object class is overridden to
     * compare OneType objects
     * Parameters   : Object o
     */

    public boolean equals ( Object o ) {
        OneType op = ( OneType ) o;
        if ( (isEqual == op.isEqual) && (isNotEqual==op.isNotEqual) &&
            (isGreaterThanOrEqual == op.isGreaterThanOrEqual) &&
(isSmallerThanOrEqual == op.isSmallerThanOrEqual) &&
            (conditionTypeName.equals(op.conditionTypeName)) &&
(conditionTypeMemberName.equals(op.conditionTypeMemberName)) ) {
            return true;
        }
        else {
            return false;
        }
    } // End of method equals

    /**
     * Method      : toString
     * Purpose      : to override toString () method of Object class
     * Parameters   : none
     */

    public String toString () {
        if ( isEqual == true ) {
            return conditionTypeName + " == " + conditionTypeMemberName;
        }
        else if (isNotEqual == true) {
            return conditionTypeName + " != " + conditionTypeMemberName;
        }
        else if (isGreaterThanOrEqual == true) {

```

```

        return conditionTypeName + " >= " + conditionTypeMemberName;
    }
    else {
        return conditionTypeName + " <= " + conditionTypeMemberName;
    }
} // End of method toString

/**
 * Method      : copy
 * Purpose     : This method creates a copy of the parameter OneType
 * object and puts it into the OneType object by which this method
 * is called.
 * Parameters  : OneType o
 */

public void copy (OneType o) {
    conditionTypeName = o.getConditionTypeName();
    conditionTypeMemberName = o.getConditionTypeMemberName();
    isEqual = o.getIsEqual();
    isNotEqual = o.getIsNotEqual();
    isGreaterThanOrEqual = o.getIsGreaterThanOrEqual ();
    isSmallerThanOrEqual = o.getIsSmallerThanOrEqual ();
} // End of method copy

/**
 * Method      : set and get methods of OneType class
 * Purpose     : To get and set the data members of OneType class
 * Parameters  : Each set method sets one data member of OneType
 * class. get methods do not have parameters.
 */

public void setConditionTypeName (String s) {
    conditionTypeName = s;
}

public String getConditionTypeName () {
    return conditionTypeName;
}

public void setConditionTypeMemberName (String s) {
    conditionTypeMemberName = s;
}

public String getConditionTypeMemberName () {
    return conditionTypeMemberName;
}

public void setIsEqual (boolean ie) {
    isEqual = ie;
}

public boolean getIsEqual () {
    return isEqual;
}

```

```

    }

    public void setIsNotEqual (boolean ine) {
        isNotEqual = ine;
    }

    public boolean getIsNotEqual () {
        return isNotEqual;
    }

    public void setIsGreaterThanOrEqual (boolean igtoe) {
        isGreaterThanOrEqual = igtoe;
    }

    public boolean getIsGreaterThanOrEqual () {
        return isGreaterThanOrEqual;
    }

    public void setIsSmallerThanOrEqual (boolean istoe) {
        isSmallerThanOrEqual = istoe;
    }

    public boolean getIsSmallerThanOrEqual () {
        return isSmallerThanOrEqual;
    }

} // End of class OneType

```

```

/*****
File: OneUserID.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/17/2001
Description: In this file, user ID values will be defined to be used in
the condition element of a policy. (user_ID == tekin)
*****/
import java.io.Serializable;

public class OneUserID implements Serializable {
    private boolean isEqual;
    private boolean isNotEqual;
    private String userIDValue;

    /**
     * Method      : equals
     * Purpose      : equals method of object class is overridden to
     * compare OneUserID objects
     * Parameters   : Object o
     */

    public boolean equals ( Object o ) {
        OneUserID op = ( OneUserID ) o;
        if ( (isEqual == op.isEqual) && (isNotEqual==op.isNotEqual) &&
            (userIDValue.equals(op.userIDValue)) ) {
            return true;
        }
        else {
            return false;
        }
    } // End of method equals

    /**
     * Method      : toString
     * Purpose      : to override toString () method of Object class
     * Parameters   : none
     */

    public String toString () {
        if (isEqual == true) {
            return "userID" + " == " + userIDValue;
        }
        else {
            return "userID" + " != " + userIDValue;
        }
    } // End of method toString

    /**
     * Method      : copy

```

```

    * Purpose      : This method creates a copy of the parameter
    * OneUserID object and puts it into the OneUserID object by which
    * this method is called.
    * Parameters   : OneUserID o
    */

public void copy (OneUserID o) {
    userIDValue = o.getUserIDValue ();
    isEqual = o.getIsEqual ();
    isNotEqual = o.getIsNotEqual ();
} // End of method copy

/**
 * Method        : set and get methods of OneUserID class
 * Purpose       : To get and set the data members of OneUserID class
 * Parameters    : Each set method sets one data member of OneUserID
 * class. get methods do not have parameters.
 */

public void setUserIDValue (String s) {
    userIDValue = s;
}

public String getUserIDValue () {
    return userIDValue;
}

public void setIsEqual (boolean b) {
    isEqual = b;
}

public boolean getIsEqual () {
    return isEqual;
}

public void setIsNotEqual (boolean b) {
    isNotEqual = b;
}

public boolean getIsNotEqual () {
    return isNotEqual;
}

} // End of class OneUserID

```

```

/*****
File: PasswordGui.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/15/2001
Description: In this file, I will design the password GUI of the
graphical user interface tool kit for path based network policy
language.
*****/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;

public class PasswordGui extends JFrame {

    private DirectorClass boss;
    private boolean passwordIsCorrect = false;
    private boolean loginIDAndPasswordAreCorrect = false;

    private JLabel loginIDLabel, passwordLabel, lineupLabel,
lineupLabel2, lineupLabel3, lineupLabel4;
    private JTextField loginIDTextField;
    private JPasswordField passwordField;
    private JButton loginButton;

    /**
     * Method      : PasswordGui
     * Purpose     : Constructor for PasswordGui class
     * Parameters  : DirectorClass x
     */

    public PasswordGui ( DirectorClass x ) {
        super ("Login Process PPL Manager");
        setLocation (115, 115);
        this.boss = x;
        createPasswordGuiBuilderMethod ( );
        setResizable (false);
        setSize(400, 300);
        show();
    } // End of constructor

    /**
     * Method      : createPasswordGuiBuilderMethod
     * Purpose     : This method builds the GUI for login process.
     * Parameters  : None
     */

    private void createPasswordGuiBuilderMethod ( ) {
        Container c = getContentPane ( );
        c.setLayout( new FlowLayout ( ) );

        loginIDLabel = new JLabel ("Login ID          ");
        c.add ( loginIDLabel );

        loginIDTextField = new JTextField (10);

```

```

c.add(loginIDTextField);

lineupLabel = new JLabel ("");
c.add(lineupLabel);

lineupLabel2 = new JLabel ("");
c.add(lineupLabel2);

passwordLabel = new JLabel ("Password");
c.add(passwordLabel);

passwordField = new JPasswordField (10);
c.add(passwordField);

lineupLabel3 = new JLabel ("");
c.add(lineupLabel3);

lineupLabel4 = new JLabel ("");
c.add(lineupLabel4);

ButtonHandler handlerBut = new ButtonHandler ();

loginButton = new JButton ("Login");
c.add(loginButton);
loginButton.addActionListener(handlerBut);

} // End of method createPasswordGuiBuilderMethod

/**
 * Class      : ButtonHandler
 * Purpose    : Inner class for event handling of OK button in login
 * window. The purpose is check if the login ID and the password of
 * the user is correct or not. If they are correct Main GUI for
 * creating the network and policies is launched. Otherwise, it is not
 * launched.
 */

private class ButtonHandler implements ActionListener {

    // This actionPerformed method checks if the loginID and password
    entered by the user are correct. If they are correct
    // Main GUI for creating the network and policies is launched.
    Otherwise, it is not launched.
    public void actionPerformed ( ActionEvent e) {

        Enumeration enum = ( boss.getPolicyMakerVector() ).elements();

        while ( enum.hasMoreElements() ) {
            PolicyMaker tempPolicyMaker = new PolicyMaker ("", "", 1);
            tempPolicyMaker = ( PolicyMaker ) enum.nextElement();

            char passwordFieldCharArray [] =
passwordField.getPassword();
            char policyMakerPasswordCharArray [] = (
tempPolicyMaker.getPasswordOfPolicyMaker() ).toCharArray();

```

```

        if ( passwordFieldCharArray.length ==
policyMakerPasswordCharArray.length ) {
            for (int i = 0; i < passwordFieldCharArray.length; i++)
            {
                if ( passwordFieldCharArray [i] ==
policyMakerPasswordCharArray [i] ) {
                    passwordIsCorrect = true;
                } // End of if
                else { // If the password is wrong
                    passwordIsCorrect = false;
                    break;
                } // End of else
            } // End of for

        } // End of if

        if ( ( ( tempPolicyMaker.getLoginID() ).equalsIgnoreCase (
loginIDTextField.getText() ) ) && ( passwordIsCorrect ) ) {
            loginIDAndPasswordAreCorrect = true;
            break;
        } // End of if

    } // End of while

    if ( ! loginIDAndPasswordAreCorrect ) {
        JOptionPane.showMessageDialog (PasswordGui.this, "Your
loginID or password is wrong.", "Wrong login ID or password",
JOptionPane.ERROR_MESSAGE);
    } // End of if
    else { // Meaning that login ID and password are correct
        boss.setPolicyMakerID ( loginIDTextField.getText() ); //
set the policyMakerID in the boss class
        boss.launchGuiMenu(); // Since login ID and password are
correct, launch the main GUI
        PasswordGui.this.dispose();
    } // End of else

    } // End of method actionPerformed

} // End of class ButtonHandler

} // End of class PasswordGui

```



```

/*****
File: Path.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/25/2001
Description: In this file, network element path will be defined.
*****/

import java.io.Serializable;
import java.util.*;

public class Path extends Object implements Serializable {
    private String name;
    private Vector pathNodesVector;
    private float bwValue;
    private boolean hasBwParameter;
    private boolean hasDelayParameter;
    private boolean hasLossRateParameter;
    private boolean hasJitterParameter;
    private boolean hasUsedBwParameter;

    /**
     * Method      : Path
     * Purpose      : Default constructor for Path class
     * Parameters    : None
     */

    public Path () {
    } // End of default constructor

    /**
     * Method      : Path
     * Purpose      : Constructor for Path class
     * Parameters    : String n, Vector v, float b, boolean bwp, boolean
     * dp, boolean lrp, boolean jp, boolean ubwp
     */

    public Path (String n, Vector v, float b, boolean bwp, boolean dp,
boolean lrp, boolean jp, boolean ubwp) {
        setName ( n );
        setPathNodesVector ( v );
        setBwValue ( b );
        setHasBwParameter ( bwp );
        setHasDelayParameter ( dp );
        setHasLossRateParameter ( lrp );
        setHasJitterParameter ( jp );
        setHasUsedBwParameter ( ubwp );
    } // End of constructor

    /**
     * Method      : toString

```

```

    * Purpose      : to override toString () method of Object class
    * Parameters   : none
    */

public String toString () {
    return name;
}

/**
 * Method        : set and get methods of Path class
 * Purpose       : To get and set the data members of Path class
 * Parameters    : Each set method sets one data member of Path class.
 * get methods do not have parameters.
 */

public void setName (String n) {
    name = n;
}

public String getName () {
    return name;
}

public void setPathNodesVector ( Vector v ) {
    pathNodesVector = v;
}

public Vector getPathNodesVector () {
    return pathNodesVector;
}

public void setBwValue (float b) {
    bwValue = b;
}

public float getBwValue () {
    return bwValue;
}

public void setHasBwParameter (boolean bwp) {
    hasBwParameter = bwp;
}

public boolean getHasBwParameter () {
    return hasBwParameter;
}

public void setHasDelayParameter (boolean dp) {
    hasDelayParameter = dp;
}

public boolean getHasDelayParameter () {
    return hasDelayParameter;
}

public void setHasLossRateParameter (boolean lrp) {

```

```

        hasLossRateParameter = lrp;
    }

    public boolean getHasLossRateParameter () {
        return hasLossRateParameter;
    }

    public void setHasJitterParameter (boolean jp) {
        hasJitterParameter = jp;
    }

    public boolean getHasJitterParameter () {
        return hasJitterParameter;
    }

    public void setHasUsedBwParameter (boolean ubwp) {
        hasUsedBwParameter = ubwp;
    }

    public boolean getHasUsedBwParameter () {
        return hasUsedBwParameter;
    }
} // End of Path class

```

```

/*****
File: Policy.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/03/2001
Description: In this file, general data structure of a policy is
created. Policies are created after the network has been defined
(nodes, links, paths, classes, types, users).
*****/

```

```

import java.io.Serializable;
import java.util.*;

```

```

public class Policy extends Object implements Serializable {

```

```

    String policyID;
    String policyMakerID;
    Vector nodesInPolicyPathVector;
    Vector linksInPolicyPathVector;
    Vector pathsInPolicyPathVector;
    Target policyTarget;
    Condition policyCondition;
    Action policyAction;

```

```

    /**
     * Method      : copy
     * Purpose     : This method creates a copy of the policy object and
     * puts it into the policy object by which this method is called.
     * Parameters  : Policy p
     */

```

```

    public void copy ( Policy p ) {
        policyID = p.getPolicyID();
        policyMakerID = p.getPolicyMakerID();
        nodesInPolicyPathVector = p.getNodesInPolicyPathVector();
        linksInPolicyPathVector = p.getLinksInPolicyPathVector();
        pathsInPolicyPathVector = p.getPathsInPolicyPathVector();
        policyTarget = p.getPolicyTarget();
        policyCondition = p.getPolicyCondition();
        policyAction = p.getPolicyAction();
    } // End of method copy

```

```

    /**
     * Method      : toString
     * Purpose     : to override toString () method of Object class
     * Parameters  : none
     */

```

```

    public String toString () {
        return policyID;
    }

```

```

    /**
     * Method      : set and get methods of Policy class

```

```

* Purpose      : To get and set the data members of Policy class
* Parameters   : Each set method sets one data member of Policy
* class. get methods do not have parameters.
*/

public void setPolicyID (String n) {
    policyID = n;
}

public String getPolicyID () {
    return policyID;
}

public void setPolicyMakerID (String n) {
    policyMakerID = n;
}

public String getPolicyMakerID () {
    return policyMakerID;
}

public void setNodesInPolicyPathVector ( Vector v ) {
    nodesInPolicyPathVector = v;
}

public Vector getNodesInPolicyPathVector () {
    return nodesInPolicyPathVector;
}

public void setLinksInPolicyPathVector ( Vector v ) {
    linksInPolicyPathVector = v;
}

public Vector getLinksInPolicyPathVector () {
    return linksInPolicyPathVector;
}

public void setPathsInPolicyPathVector ( Vector v ) {
    pathsInPolicyPathVector = v;
}

public Vector getPathsInPolicyPathVector () {
    return pathsInPolicyPathVector;
}

public void setPolicyTarget ( Target t) {
    policyTarget = t;
}

public Target getPolicyTarget () {
    return policyTarget;
}

public void setPolicyCondition ( Condition c ) {
    policyCondition = c;
}

```

```
    }

    public Condition getPolicyCondition () {
        return policyCondition;
    }

    public void setPolicyAction ( Action a ) {
        policyAction = a;
    }

    public Action getPolicyAction () {
        return policyAction;
    }

} // End of class Policy
```

```

/*****
File: PolicyMaker.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/16/2001
Description: In this file, policy makers who can create network
policies will be defined.
*****/
import java.io.Serializable;

public class PolicyMaker implements Serializable {
    private String loginID;
    private String password;
    private int policyMakerPriority;

    /**
     * Method      : PolicyMaker
     * Purpose      : Constructor with 3 arguments. First argument is
     * login ID, second is password and third is policy maker priority.
     * Parameters   : String s, String p, int u
     */

    public PolicyMaker (String s, String p, int u ) {
        loginID = s;
        password = p;
        policyMakerPriority = u;
    } // End of constructor

    /**
     * Method      : toString
     * Purpose      : to override toString () method of Object class
     * Parameters   : none
     */

    public String toString () {
        return loginID;
    }

    /**
     * Method      : set and get methods of PolicyMaker class
     * Purpose      : To get and set the data members of PolicyMaker
     * class
     * Parameters   : Each set method sets one data member of PolicyMaker
     * class. get methods do not have parameters.
     */

    public void setLoginID (String s) {
        loginID = s;
    }

    public String getLoginID () {
        return loginID;
    }

    public void setPasswordOfPolicyMaker (String s) {
        password = s;
    }

```

```
    }

    public String getPasswordOfPolicyMaker () {
        return password;
    }

    public void setPolicyMakerPriority (int a) {
        policyMakerPriority = a;
    }

    public int getPolicyMakerPriority () {
        return policyMakerPriority;
    }

} // End of class PolicyMaker
```



```

/*****
File: Target.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 11/04/2001
Description: In this file, target element of a policy will be defined.
*****/
import java.util.*;
import java.io.Serializable;

public class Target implements Serializable {
    private boolean targetAllProperty; // This will be a check box item.
    When the user checks it, "*" will be written to file in the target
    element of a policy (meaning target all).
    private Vector policyTargets; // OneTarget class objects will be
    placed in this vector.

    /**
     * Method      : set and get methods of Target class
     * Purpose      : To set and get the data members of Target class
     * Parameters   : Each set method sets one data member of Target
     * class. get methods do not have parameters.
     */

    public void setTargetAllProperty (boolean tap) {
        targetAllProperty = tap;
    }

    public boolean getTargetAllProperty () {
        return targetAllProperty;
    }

    public void setPolicyTargets (Vector v) {
        policyTargets = v;
    }

    public Vector getPolicyTargets () {
        return policyTargets;
    }

} // End of class Target

```

```

/*****
File: Type.java
Name: Tufan Ekin
Course: Thesis (CS 0810)
Date: 10/30/2001
Description: In this file, user defined types that can be used in the
conditional element of a policy rule (items are AND'ed) will be
defined.
*****/

```

```

import java.io.Serializable;
import java.util.*;

```

```

public class Type extends Object implements Serializable {
    private String name;
    private Vector typeMembersVector;

```

```

    /**
     * Method      : Type
     * Purpose     : Default constructor for Type class
     * Parameters  : None
     */

```

```

    // Default constructor
    public Type () {

    } // End of default constructor

```

```

    /**
     * Method      : Type
     * Purpose     : Constructor for Type class
     * Parameters  : String n, Vector v
     */

```

```

    public Type ( String n, Vector v ) {
        setName ( n );
        setTypeMembersVector ( v );
    } // End of Constructor

```

```

    /**
     * Method      : toString
     * Purpose     : to override toString () method of Object class
     * Parameters  : none
     */

```

```

    public String toString () {
        return name;
    }

```

```

    /**
     * Method      : set and get methods of Type class

```

```

    * Purpose      : To get and set the data members of Type class
    * Parameters   : Each set method sets one data member of Type class.
get methods do not have parameters.
    */

    public void setName (String n) {
        name = n;
    }

    public String getName () {
        return name;
    }

    public void setTypeMembersVector ( Vector v ) {
        typeMembersVector = v;
    }

    public Vector getTypeMembersVector () {
        return typeMembersVector;
    }

} // End of Type class

```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Strassner, J. and Ellessen, E., “Terminology for describing network policy and services”, Internet Engineering Task Force, Internet Draft. Draft-strassner-policy-terms-01.txt, February 1999.
2. Stone, G.N., “A Path-Based Network Policy Language”, Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, September, 2000.
3. Naval Postgraduate School Department of Computer Science Report No: NPS-CS-00-003, “Network Policy Languages: A Survey and a New Approach”, by Stone, G.N., Xie, G. and Lundy, B., August 2000.
4. Shneiderman, B., “Designing the User Interface: Strategies for Effective Human-Computer Interaction”, Addison-Wesley, 1992.
5. Hix, D. and Hartson, H.R., “Developing User Interfaces, Ensuring Usability Through Product & Process”, John Wiley & Sons, Inc, 1993.
6. Sun Microsystems, “Policy Tool – Policy File Creation and Management Tool”, [<http://java.sun.com/products/jdk/1.2/docs/tooldocs/win32/policytool.html>], March 2002.
7. Xiao, X. and Ni, L., “Internet QoS: A Big Picture”, IEEE Network, v. 13 Issue 2, pp. 8-18, March/April 1999.
8. Vrable, D. and Yarger, J., “The SAAM Architecture: Enabling Integrated Services”, Master’s Thesis, Naval Postgraduate School, Monterey, California, September 1999.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, VA
2. Dudley Knox Library
Naval Postgraduate School
Monterey, CA
3. Professor Dan Boger
Naval Postgraduate School
Monterey, California
4. RADM Winsor Whiton
Naval Security Group Command
Fort Mead, Maryland
5. Deniz Kuvvetleri Komutanligi
Personel Daire Baskanligi
Bakanliklar
Ankara, TURKEY
6. Deniz Harp Okulu Komutanligi
Kutuphanesi
Tuzla
Istanbul, TURKEY
7. Chairman, Code CS
Naval Postgraduate School
Monterey, CA
8. Prof. Geoffrey Xie, Code CS
Naval Postgraduate School
Monterey, CA
9. Prof. James Bret Michael, Code CS
Naval Postgraduate School
Monterey, CA

10. Tufan Ekin, LTJG
Deniz Kuvvetleri Komutanligi
MEBS Baskanligi
Bilgi Sistem Daire Baskanligi
Bilgi Sistem Gelistirme Sube Mudurlugu
Bakanliklar
Ankara, TURKEY